FLAMENCO: A Programmatic Labeling and Sharing Framework for Internet Data Science

DRP Summer 2021

Yukhe Lavinia
University of Oregon
September 16, 2021

ABSTRACT

The success of Internet Data Science depends on the availability of high-quality labeled data (e.g., onset of a DDoS in NetFlow log). Equally critical is the ability to share the data with others, respecting the data owners' privacy concerns. Unfortunately, short of applying the data-to-code paradigm (i.e., actual sharing of data), researchers lack a systematic framework for working with or benefiting from data while being mindful of privacy concerns. As a result, Internet Data Science as practiced today is not amenable to leveraging the collective domain knowledge of the community for important ML-related activities such as (i) high-quality data labeling at scale, (ii) sharing of domain knowledge in a privacy-preserving manner, and (iii) creating a viable roadmap for their adoption by operators due to lack of capabilities to interpret the trained ML models.

We propose a novel code-to-data approach whose goals are to benefit data ownership, preserve privacy in collaborations, facilitate independent validation of each others' findings, and enable the interpretability of trained ML models. Here, code refers to *labeling functions* which we view as programmatic representations of operators' domain knowledge to identify events of interest in the network data. The key novelty of our approach is that it entails *only* the sharing of code and *no* sharing of any raw or curated data or trained ML models. We substantiate our approach by building FLAMENCO—a novel weak supervision-based framework to collaboratively label network data at scale while being mindful of data owner's privacy concerns. We demonstrate the efficacy of FLAMENCO by labeling diverse networking data programmatically, enabling privacy-preserving collaboration among researchers using those data, and facilitating the interpretability of models trained on those data.

1 INTRODUCTION

The success of *Internet Data Science*—defined as the application of machine learning or ML for network performance and security problems—depends crucially on the availability of high-quality labeled datasets collected from different real-world networks. Equally critical is the ability of researchers, other than those that own such data, to maximally benefit from the existence of these datasets while being mindful of critical privacy requirements that the data owners impose and that are unique to network data. However, except for following the "data-to-code" paradigm that involves the actual sharing of raw or curated datasets, today's researchers lack a systematic framework for working with or benefiting from datasets that exist outside their own networks or groups and that the owners of this data consider as being strictly off-limit when it comes to sharing with third parties.

To illustrate, consider an advanced volumetric DDoS attack scenario faced by two enterprise networks. In this scenario, the attackers targeting the two networks can be the same or different actors, and the types of DDoS attacks seen by the two networks can be the same or different (e.g., amplification attacks [36]). Moreover, the network infrastructures at the two networks are likely to be different. For example, the two networks may use firewall devices from different vendors, and even if the devices are from the same vendor, they can be expected to be configured differently. In any case, the combined telemetry information (e.g., Netflow data, firewall logs) collected at the two networks is likely to be substantially richer than the information that each enclave would collect separately. We posit that to obtain high-quality labels for these advanced attacks and to train/apply high-performance ML models to detect them, researchers should be able to fully benefit from and readily tap into such combined information sources.

Unfortunately, in view of how most of today's networks operate, collaboratively tackling such networking problems, be they security- or performance-related, is by and large impossible. For one, by functioning essentially as autonomous entities, today's network researchers work on such problems typically in isolation, mainly due to serious privacy concerns and liability issues surrounding the data under their control. Second, there exists no community-wide standard and/or agreed-upon features for identifying different events of interest. Finally, today's network researchers lack the capabilities to effectively leverage the collective domain knowledge of the community for important ML-related activities such as the scenario described above. Consequently, Internet Data

Science efforts lack (i) a rigorous evaluation of newly developed research artifacts such as learning models, (ii) an independent validation of reported research findings and effective means for their scientific reproducibility, and (iii) a viable roadmap for their adoption by network operators. In short, there is an increased need for a paradigm shift whereby the traditional but problematic "data-to-code" approach is replaced by an approach that is specifically designed for the sharing of domain knowledge among third parties as a means for more accurate identification/detection of more events of interest in the network data. This paradigm shift should benefit data ownership, preserve privacy, and facilitate collaborative learning and independent validation of each others' findings.

In this work, we intend to democratize Internet Data Science by arguing for and introducing a novel "code-to-data" approach. To motivate and illustrate our approach, we consider in the following the concrete use case of collaborative data labeling. Here, "code" refers to purposefully-defined *labeling functions* which we view as programmatic representations of network operators' domain knowledge to identify events of interest in the network data. By capturing and leveraging this domain knowledge, and together with other weak sources, we can generate "noisy" or "weak" labels for the unlabeled training data. The labels are expected to be inaccurate, hence the term "weak" labels. [11] introduced this type of learning and gave the term weak supervision, or weakly supervised learning. Popular form of weak supervision include distant supervision [8, 27] and crowd-sourcing with non-expert annotators [35, 48]. We developed our approach in a weak supervised setting. The key novelty of our approach is that it entails *only* the sharing of code and *no* sharing of any raw or curated data. To substantiate our approach, we build FLAMENCO, a new weak supervision-based framework capable of supporting all activities such as (i) labeling diverse networking datasets at scale, (ii) facilitating privacy-preserving collaborations among researchers/operators from different networks, and (iii) explaining the decision taken by downstream ML models that are trained using the labeled datasets.

With those activities in mind, we designed FLAMENCO to consist of four primary components: data preprocessing, label generation, label evaluation, and dataset-specific interpretability tree generation.

In the data preprocessing component, FLAMENCO (i) handles latency measurement data with various modality and also Netflow data, and (ii) facilitates feature engineering. In the label generation component, FLAMENCO generates training labels using either automatic labeling function generator or manual labeling function generator to facilitate user's preference and need. In the labeling evaluation component, FLAMENCO evaluates the quality of the training labels in a supervised learning setting. FLAMENCO trains a classifier using the newly labeled training set and tests the performance to determine the quality of the training labels. In the dataset-specific interpretability tree generation, FLAMENCO renders a tree similar to the decision tree structure to provide explanations on the framework's decision in determining labels. We built FLAMENCO upon the following points: (i) as a programmatic label generation, labeling function encapsulates domain expertise to determine events of interest and enables FLAMENCO to adapt to increased network data variation and size, (ii) combining labeling functions allows better learning to improve label quality and accommodate research collaborations without exposing privacy, and (iii) labeling functions can be mapped to a tree to visualize the decision making process in label assignment. In addition to these, we also compared the F1 scores of the manual labeling function generator, the trained classifier, and the tree.

We demonstrate the efficacy of FLAMENCO by evaluating it on two latency measurement datasets (CAIDA's Ark [2] and RIPE Atlas [5]) and two flow datasets (CIC DDoS NTP [3] and Worf [4]).

On the latency measurement datasets, we show how our framework can scale domain expertise and that the classifiers that are trained using our labels outperform those trained using the labels created by unsupervised learning methods. On the flow datasets, we show how our framework support collaboration without risking privacy, analyze the effects of various labeling function combinations,compare the labeling function F1 scores with that of trained classifiers, provide a level of interpretability on the results in a form of a tree, and compare the F1 scores of these interpretability trees with the F1 scores of the labeling functions from which the trees are generated.

This paper makes the following key contributions:

- **Design and implementation of framework FLAMENCO**. We propose a framework to create training labels for two different network data types: (1) latency measurements in the form of RTT (round-trip time) in ms, and (2) NetFlow data, which are traffic flow data with various features (*e.g.,* flow duration, average packet size in the flow, number of packets in the flow, number of bytes in the flow) of their own. We also designed this framework to be mindful of the privacy issue and easier to interpret.
- **Application of FLAMENCO to demonstrate the labeling of diverse datasets**, which will be done by evaluating our framework on four datasets: (1) CAIDA's Ark [2], (2) RIPE Atlas [5], (3) CIC DDoS NTP [3], and (4) Worf [4].
- **Application of FLAMENCO to enhance collaboration among researchers**, which will be done by showing

how different research groups can collaborate using the "code-to-data" approach without risking any privacy. The code that we use will be in the form of "labeling function," the primary element in the data programming paradigm [34].

- **Application of FLAMENCO to explain decisions**, which will be done by rendering trees to visualize the decision paths. We seek to improve the trust of network operators in ML by providing explanations on how our framework comes to a decision.

2 BACKGROUND AND MOTIVATION

In this section we will discuss the existing solutions that address the problem of training label scarcity in ML. We will also elaborate on why we deem these solutions inadequate when applied in the networking domain.

2.1 State-of-the-art

2.1.1 Ground Truth Labels. The scarcity of ground truth labels in Internet Data Science is primarily caused by the cost-prohibitive nature of data labeling efforts [28]. The scarcity problem is further complicated by the fact that networking data have a high variability as they are collected from different layers of the protocol stack (e.g., layer 3 traceroute measurements vs. layers 2 to 4 Netflow logs) and in different granularities (e.g., packets vs. flows vs. logs). Furthermore, there is a lack of consensus in the networking community about the features that should be found in the data to describe an event of interest. In addition to these, labeling networking data can only be done by domain experts (e.g, network operators, researchers) due to intimate knowledge of the data. This naturally obviates techniques such as crowdsourcing (e.g., [13, 47]) as potential candidates to address this problem.

Several works seek to address this label shortage, and one of the approaches is weak supervision. This approach assumes that the labelers are domain experts who act as a resource in providing ground truth labels. Since the primary problem in creating a labeled training set is the limited amount of labels that the domain experts can create, and thus only a subset of the data that can be labeled, weak supervision leverages this limited amount of ground truth labels to assign "weak" or "noisy" labels to the rest of the data to create a fully-labeled training set. For example, Snorkel [33] uses the data programming paradigm [34] to provide a way for users—usually domain experts—to write programs (also known as "labeling functions" or LFs) that describe the features of interest to categorize the data into different classes. Subsequently, Snorkel DryBell [7] was introduced to create training labels at a massive industrial scale (6 million data points). Since writing LFs is a burdensome task, especially when the amount of datasets increases, Snuba [43] was introduced to alleviate this burden to automatically generate LFs. Snuba requires users to provide only a small set of labeled data to learn from and create labels for the rest of the data. Note that these techniques are not specific to networking, as they are either domain-agnostic or specific to a certain domain, such as computer vision. This lack of specificity for the networking domain renders them unsuitable to handle the networking data and their peculiarities. A domain-agnostic framework need modifications to handle network data, as some network data, such as latency measurements, require different treatment for different source-destination pair datasets. Since the task of modifying the input space and data processing is non-trivial, these domain-agnostic frameworks cannot readily handle network data.

There are data labeling frameworks, however, that are specific to networking. NoMoNoise [28], a recently proposed framework to denoise latency measurements, leverages the benefits of Snorkel [33] and the data programming paradigm [34] in weak supervision setting to enable users to create high-quality labels by combining and learning noisy labels from many sources, without access to ground truth labels. NoMoNoise, however, is limited to only one type of networking data, that is, latency measurement data, and is also limited in scale, as it requires users—who are usually the few domain experts—to study the data pattern for an event of interest before writing the labeling functions; a task that could become labor-intensive as the amount and variability of datasets increase. Another framework to label networking data using weak supervised learning is EMERGE [24]. Extending from NoMoNoise [28] and inspired by Snuba [43], EMERGE generates labels to discriminate noise from good latency measurement data and does so without requiring users to find pattern in the data to write labeling functions, which makes EMERGE more scalable in handling a large amount and variety of networking data. However, EMERGE is specific only to latency measurement data, and thus it has limited versatility to handle other types of networking data.

To put it briefly, many have proposed solutions to address the problem of training label shortage. The solutions vary on the domain in which they apply; some are specific only to certain domain (*e.g.,* GOGGLES) or data type (*e.g.,* [13], NoMoNoise [28], EMERGE [24]), while some are too broad to readily handle network data (*e.g,* Snorkel [33], Snuba [43]).

2.1.2 Privacy Issues. While data privacy in other domains may not be a sticking point since not all data contain sensitive information (e.g., images of dog breeds, birds, or flowers), almost all networking data contain identifying and sensitive information. Except for following the "data-to-code" paradigm (i.e., actual sharing of raw or curated datasets), today's researchers lack a systematic framework for working with or

benefiting from datasets that exist outside their own groups and that the owners of this data consider as being strictly off-limit when it comes to sharing with third parties. Such privacy concern becomes a roadblock in collaborations among research groups in applying ML on networking, because they could not share the training data, the ML models, or the models' predictions.

Moreover, ML was also observed to have exposed security and privacy vulnerabilities in the software systems that adopted it [30]. Attackers who exploited these vulnerabilities usually are interested in recovering information on the ML algorithms used in the deployed models or on the data used in the training set, since these data are often proprietary or contain sensitive information protected by the data privacy laws. The types of attack depend on the level of information access that the attackers had on the systems. Access to ML models and their parameters allowed attackers to determine the statistical characteristics of the training set and generate datasets out of the information gained from the attack [6].

Allowing access only to the ML models and their output, as can be found in ML-as-a-service systems, can still wreak havoc. The information about ML models and the outputs still allows attackers to conduct myriads of other attacks. One of the attacks is membership inference [40]. In this type of attack, attackers could determine if a certain data point is part of the training set. Another attack that can be launched from ML model and output information is model inversion [15, 16]. This vulnerability allowed attackers to extract the training set and gain sensitive information contained in the data. Finally, ML models and output exposure can be exploited for model extraction [41], which could potentially leak sensitive information in the training set, as this attack gave attackers access to the model, on which they could further exploit by conducting model inversion. Considering all these risks, the safest way to prevent privacy leaks is to withhold access to the ML models, the training data, and the output of the ML models.

While approaches like federated learning and multi-party cryptographic collaborative learning are proposed to tackle such data and privacy issues, we posit that both those approaches are susceptible to new types of attacks. For example, federated learning in its current form has problems with leaking information (e.g., membership inference attack) and may therefore not be suitable for the collaborative learning scenarios we envision in this project [32]. Similarly, multi-party cryptographic collaborative learning is prone to contamination attacks by the one or more of the participating networks [18], dealing which is beyond the scope of this work.

To summarize, the existing solutions are not designed with privacy protection in mind, and this renders them unfit to handle network data where privacy is a major concern.

2.1.3 Model Interpretability. Lastly, the lack of transparency in ML models, which has eroded the trust that network operators have in ML models, is a significant issue in networking. Since these operators are accountable for their actions and decisions, it is unsurprising if they require interpretability in the ML models that they plan to adopt.

Recent works have also brought into attention the growing concern over the blackbox nature of many ML models, particularly when the models' predictions have a significant impact on people's lives. In 2016, a recidivism prediction algorithm called COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) caused a controversy when it was claimed to be racially biased [17, 22], although it seemed that the claim was not fully validated [14, 39]. The fact that such claim could have arisen, coupled with other instances where ML predictions have shown to be wrong [25, 46] while affecting the safety and well-being of many [29, 44], has led to a rising hesitancy in blind acceptance of blackbox ML predictions, and even to a growing voice to avoid the use of blackbox ML models when interpretable models would perform equally well [37, 38, 45]. This voice of concern seems to not fall of deaf ears, as efforts to make ML model's predictions explainable have gained traction recently, such as FICO's Explainable Machine Learning Challenge in 2017 [1], DARPA's program of Explainable AI (XAI) [42], Google Cloud's XAI tools [10], and IBM's [21], to name a few. The goal of these efforts is to gain understanding as to how the model makes a prediction, including what variables it considered and the threshold values it compares to. In achieving such a level of interpretability, trust in the ML models would hopefully grow.

To summarize, the solutions still use ML as a black box, and given the rise of hesitation to trust ML blindly, it presents a problem of slow adoption of ML in the networking domain.

2.2 Limitations of State-of-the-art

These prior efforts fall short in handling the networking data in their entirety due to the following limitations (**L**s):

L1: Inability to handle diverse networking data, which in turn leads to a general lack of agreed specificity among the networking community on describing events of interest. Hence, GOGGLES [13], as a framework to address labeling problems in computer vision, would be unsuitable. The other frameworks (Snorkel [33], Snorkel DryBell [7], Snuba [43]) are domain-agnostic, which, some could argue, would be suitable to solve networking problems. However, the task of data preprocessing and finding the patterns in the networking data is non-trivial, and thus we need a weak supervision framework that is *specific* to networking and ready to handle data collected from different layers of the network stack (TTL-limited, NetFlow) and at different granularities

(packets and flows). On the other hand, frameworks that are *too* specific to only one kind of networking data would also fall short, as in the case of NoMoNoise [28] and EMERGE [24].

L2: Inability to scale the domain expertise to cover larger and more datasets. Since labeling networking data requires domain expertise, frameworks such as CROWDGAME [47], which employs crowdsourcing, would be ill-suited. It is also important to note that many networking data, especially latency measurement data, are separated into different datasets based on their source and destination location, as different source and destination datasets have different statistical profiles (e.g., RTTs in the data between Seattle and Los Angeles would be markedly different in values from RTTs in the data between Seattle and Shanghai), and thus each dataset requires different processing. In light of this, while frameworks such as Snorkel DryBell [7] are developed to handle a massive amount of data, it is still not a readily available framework to handle a large amount of networking data that consist of unique source-destination data points.

L3: Lack of support for privacy-preserving collaboration. The frameworks mentioned above made no mention of data privacy as one of their chief concerns in design. As research advances from a community effort in learning from each other, in which collaboration is encouraged, most of the frameworks above are ill-suited to support research collaboration in the networking domain, as most of the networking data contain sensitive information.

L4: Lack of capabilities to facilitate interpretability in the ML-based methods. There is a general hesitancy among network operators to adopt machine learning-based tools due to their lack of interpretability since many of the complex machine learning techniques, especially deep learning, are treated as blackbox and thus their inferences and how they arrive at the results are left without explanation. Most of the frameworks above use complex ML methods that are hard to interpret.

3 FLAMENCO: DESIGN AND IMPLEMENTATION

In this section we will present the requirements that a solution must meet to address the limitations discussed in §2.2. We will also present the key intuitions upon which we build our solution. Finally, we will describe the design and implementation of our proposed solution.

3.1 Requirements

We posit that an "ideal" framework for democratizing AI/ML for networking should meet the following requirements (**R**s):

```
def LF_Packets_75percentile_Benign(x):

    val = x.number2.get_attrib_tokens()
    if float(val[0]) <= df_stats_features.iloc[idx['75%']]['Packets']:
        return 1
    else:
        return 0
```

Figure 1: Example of labeling function.

R1: Versatility to handle various networking data. An ideal framework to label networking data should be versatile enough in its input space to accept data from different layers of the networking stack, which often are in the form of time series with most variables containing numerical values, but also stringent enough to only accept certain types of data to ensure a smooth flow with the minimum time needed on the data preprocessing step.

R2: Ability to scale the domain expertise to label data. The ideal framework should have the ability to broaden the coverage of the available domain expertise over more and larger datasets at a low cost. This could be done by replicating the labeling process that the domain experts perform without them being directly involved; a system that can mimic the human reasoning involved to assign labels and apply them to a large amount of data.

R3: Ability to preserve privacy. The framework should serve as a platform that promotes collaboration between research groups in the overall effort to democratize the use of ML in networking research, while also preserving privacy. The framework shares neither raw data nor ML models, but instead only labeling techniques regarding the features and threshold values. The labeling information can be shared among different research groups working on the same granularity of network data (*e.g.,* flows) to solve the same problem (*e.g.,* DDoS attack detection) without exposing any personally identifiable information.

R4: Interpretability. The framework should offer transparency in its decision-making. This can be done by providing the users a visualization in the labeling process, such as the patterns that the system use, and why it categorizes certain value in one class, while others in another class. The ability to explain the logical steps in the model's decision-making process improves the framework's accountability, which hopefully would inspire a more trusting stance among newly ML adopters in the networking community.

3.2 Key Insights

As described in § 2.1.1, it is well known that the data programming paradigm employs labeling functions on the unlabeled data and learns the joint probability distributions of the noisy labels to produce ground truth labels in a scalable way [28]. We build on this paradigm and use the following key intuitions (**I**s) in this work:

- **I1: Labeling functions can capture the operator's domain knowledge in identifying events of interest in the data, and enable a scalable way to programmatically label diverse networking data.** This programmatic approach answers the issue of scalability §2.2 **L2** and fulfills §3.1 **R2**. Figure 1 shows an example of a labeling function. This labeling function encodes a heuristic, or rule, that considers data with the number of packets less than or equal to its 75th percentile to be benign (returning 1) and abstains from determining the data that do not meet the condition (returning 0).

  This example shows that labeling functions are basically heuristics encoded in a program. Users are given the ability to write labeling functions to identify *multiple* events of interest on the *same* dataset. For instance, a network experiences two types of DDoS attacks: an NTP attack and a SYN flood attack. In a flow dataset the SYN flag count field would play a major role to detect a SYN flood attack, while this field may not be important to detect NTP attack. On this same network from which the flow dataset is formed, the first domain expert can write labeling functions to identify NTP attacks, while another domain expert can write labeling functions to identify SYN flood attacks. In this scenario, instead of putting permanent labels on the dataset, which can create a *fixed* labeled dataset to identify only *one* event of interest, we can actually identify *two* events of interest (NTP attacks and SYN flood attacks) on the same dataset. In this approach, our proposed labeling technique is *flexible*.

  When it comes to scalability, although labeling functions are unique to specific networks, labeling functions can still capture the network's statistical or universal (*e.g.,* ports) thresholds to be applicable to other networks. For instance, one dataset can have a flow size average of 1000 bits, while another dataset 5000 bits. By using statistical characteristics (*e.g.,* mean, standard deviation, median and their derivation) of the first dataset to write the labeling functions, users can apply these labeling functions to the second dataset. This allows the labeling functions to be applicable to many datasets, and thus solves the scalability issue §2.2 **L2**.
- **I2: Joining two or more labeling functions enables better learning and provides an opportunity to collaborate in a privacy-preserving fashion where only "code" is shared across groups instead of the data or the trained models.** This approach provides a solution to the issue of privacy in research collaboration §2.2 **L3** and meets §3.1 **R3**.

  We aim to address two issues that are found to be the causes of the label scarcity in the networking data: (1) lack of agreement in the community on features to identify certain events of interest, and (2) the risk of privacy leaks in the research collaborative setting. Since reaching a consensus involves the whole community, one way to move towards a consensus is through collaboration between research groups. To illustrate, a research group working on a problem of identifying characteristics of an HTTP flood attack can collaborate with another group who are also working on the same problem.

  Recall that labeling functions can overlap and conflict with each other, and by combining multiple labeling functions, we can estimate their accuracy by counting their frequency of agreement and disagreement between them. Labeling functions with a higher frequency of agreement with others will be considered more accurate, while those with low frequency of agreement with others will be considered less accurate. In generating labels, labeling functions will vote on a label (-1 or 1), with the votes weighted by their respective estimated accuracy. [34] showed that this combination generated quality labels without the labor-intensiveness of hand-labeling approach. With this benefit of combining labeling functions, collaboration between different research groups by sharing their respective labeling functions would form a richer learning material. Since more knowledge can be shared and accumulated into a community knowledge, we could move closer towards a consensus.

  The privacy leaks issue can be avoided by sharing *only* the labeling functions between the groups; no data or ML models are shared. Since labeling functions practically are heuristics (or rules), sharing these labeling functions resembles sharing rules with others, which allow a greater community to assess and study the findings.
- **I3: Labeling functions that are used to create data labels and train ML models can also be expanded to dataset-specific interpretability trees.** Using these trees, operators can run the data points along different branches to "explain" the classification/prediction decisions made by the trained ML models. We acknowledge that the labeling functions produce noisy labels that might not be as accurate as those created by hand. Our attempt is to offer a degree of transparency in the decision-making process. One may argue that if the tree merely visualizes the labeling functions that the user wrote, then the tree does not add value to the understanding on the decision-making process. To respond, we posit the following:
  - The hierarchy of the nodes are based on feature importance, which can offer the user an understanding on why the path goes in a certain direction and not the other. A user who writes labeling functions that are based on two different features on the same dataset might not always be aware of the order of importance,

and thus the visualization of these two labeling functions and the path taken to arrive at the conclusion can still add a degree of explanation.

- Since a tree represents one dataset at a time, and since a dataset represents a network, the structure of the tree is determined by the features and the values of those features. This is because these features and their values affect the threshold calculation in the labeling function and also affect the feature importance score. The order of importance of the features largely determines the feature and the threshold on the tree nodes. When generating the dataset-specific interpretability tree that is based on a labeling function combination from two different groups, the user of each group can readily trace the logical path without having to remember or recalculate feature importance in order to gain understanding of the decision-making process.

We posit that the dataset-specific interpretability tree offers a solution to the issue of interpretability in §2.2 **L4** and fulfills requirement §3.1 **R4**.

3.3 Overview and Design

Building on intuitions §3.2 **I1**, §3.2 **I2**, and §3.2 **I3**, we intend to revolutionize Internet Data Science by arguing for and introducing a novel approach we term "code-to-data". To motivate and illustrate our approach, we consider in this work the concrete use case of collaborative data labeling. In this case, "code" refers to purposefully-defined *labeling functions* that encapsulate domain expertise to identify events of interest. The key novelty of our approach is that it entails *only* the sharing of code and *no* sharing of any raw or curated data. Our high-level objective is to facilitate the sharing of domain knowledge across research groups in pursuit of Internet Data Science in a way that benefits data ownership, preserves data privacy, and facilitates collaborative learning and independent validation of each others' findings. The overall framework pipeline is shown in Figure 2.

It is well known that data programming facilitates scaling by applying labeling functions on unlabeled data and that weak supervision produces ground truth via learning of joint probability distributions. So we build on data programming and use the intuitions discussed in §3.2, which are: (**I1**) Labeling functions capture operator's domain knowledge and enable scalable data labeling, (**I2**) Joining labeling functions enables better learning and opens up new ways to collaborate in a privacy-preserving fashion where only "code" is sent instead of the data or the trained models, and (**I3**) Labeling functions can be expanded to interpretability trees and operators can run the data points on those trees to "explain" the decisions made by the ML models. The design of FLAMENCO follows these intuitions and aims to provide a framework that supports (1) generating labeling functions, (2) improving labeling functions, and (3) understanding the results.

To achieve this goal, we propose the design of FLAMENCO (shown in Figure 2), a novel framework capable of supporting all the requirements (described in § 3.1) to address training label shortage, support collaboration that avoids privacy leak, and provide explanations on the decision-making in the labeling process. At its core, the FLAMENCO framework consists of the following four components:

- a **data preprocessing** (§ 3.3.1) component that takes the input data and prepares them for the next component, consisting of (i) a modality detection module, (ii) a synthetic data creation module, (iii) a threshold calculation module, (iv) a data loader module in which the data's statistical features are extracted and the data are split into training, validation, and test sets, and (v) a data preparation module in which the data features are selected and the data split into training, validation, and test sets;
- a **label generation** (§ 3.3.2) component in which the probabilistic labels for the training set are generated, consisting of (i) an automatic labeling function generation module, and (ii) a manual labeling function generation module;
- a **label evaluation** (§ 3.3.3) component consisting of (i) an end-classifier training using the training set and the newly generated probabilistic labels in a supervised learning setting, and (ii) platform to combine labeling functions to improve label quality; and
- **dataset-specific interpretability tree generation** (§ 3.3.4) component that will generate trees for the datasets.

We describe each one of these components below.

3.3.1 **Data preprocessing.** This component takes the input dataset in the CSV format and prepares the data in a format acceptable for label generation. The component consists of the following modules:

Modality detection. This module is developed to analyze a dataset with continuous numerical values that does not conform to the normal distribution with the goal of splitting the bimodal or multimodal data into separate subsets. Although the module was developed with latency measurement data (e.g., RTT) in mind, it can work with any continuous numerical data. The module works by finding the number of peaks in the data distribution using KDE computation, finding the lowest points between the peaks, and using these lowest points to split the peaks into different subsets. The purpose of splitting bimodal and multimodal data into a unimodal data subset is to extract the statistical characteristics of each peak in isolation, since the statistical characteristics of bimodal or multimodal data are different than those of the

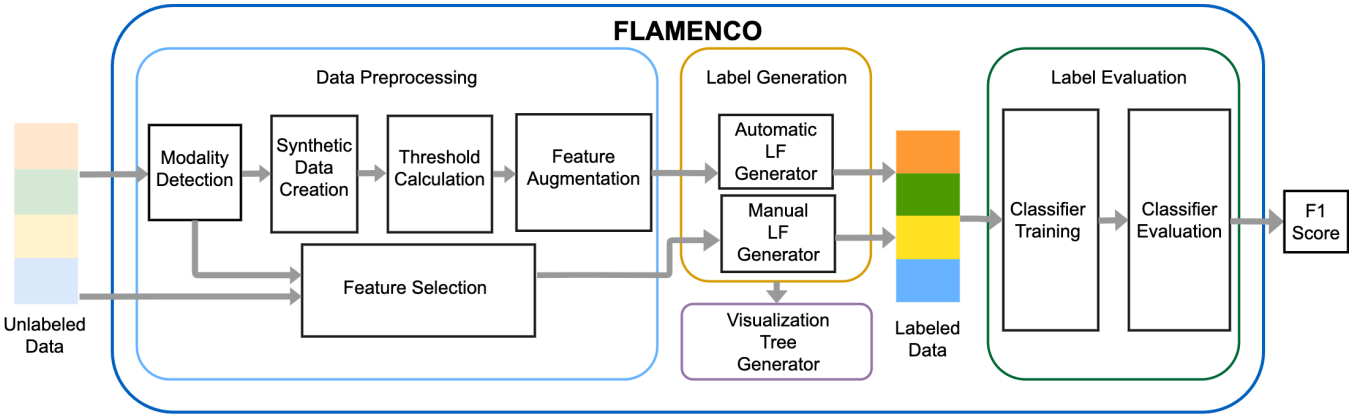


Figure 2: Overview of FLAMENCO. The four main components are data preprocessing, label generation, visualization tree generator, and label evaluation. After labeling functions are generated by either automatic or manual method, the labeling functions are passed to the visualization tree component. We use only one arrow from "Label Generation" to "Visualization Tree Generator" to indicate that labeling functions from either method can feed into the "Visualization Tree Generator.

individual peaks. These statistical characteristics can then be used for further analysis by other modules or components.

Synthetic data creation. This module is created to facilitate oversampling, which the user might use in dealing with imbalanced data. The module works by finding the size of the synthetic data, creating random values according to user's specified lower and upper bounds, and injecting the synthetic data into the original data.

Threshold calculation. This module is developed to facilitate user in determining the threshold value with which the user might want to use to separate the negative class from the positive class. The module assumes unimodal distribution in the data and uses the statistical characteristics of the data to calculate the threshold value.

Feature augmentation. This module is designed to handle time series data with only one feature. The module will add statistical features (count, minimum, mean, standard deviation, first quartile, median, third quartile, and maximum) to the feature column. Then the module will split the data into train, validation, test sets, and label the validation and test sets. So in this module, the data undergo the following process: (1) statistical feature augmentation, (2) creation of a new column containing the calculated features, (3) training, validation, and test split, (4) validation set labeling, and (5) test set labeling. The outputs of this module are training feature matrix, training set, validation feature matrix, validation set, validation ground truth labels set, test feature matrix, test set, test ground truth labels.

Feature selection. While the feature augmentation component is used to *add* the number of features, the feature selection module is used to *reduce* the number of features. The goal of this module is to select the features that can benefit users in writing labeling functions. We realize that some users may already know exactly what features they want to include. In this case, they merely need to proceed and select the features they wanted. However, we also realize that there are users who would want to explore and study the features before selecting any. This module facilitates this latter group of users by computing the feature correlation, which the users can use to select the features. After feature selection, this module will split the data into training, validation, and test sets, and allow users to create ground truth labels for the validation and test sets. Next, the module will write the training, validation, and test sets into separate TSV files. Lastly, the module creates a pandas DataFrame [26] containing the statistical characteristics of the selected features and writes these data into a CSV file. This last step is to facilitate users in deciding threshold values for their labeling functions.

3.3.2 **Label generation.** The label generator component operates in a weakly supervised setting. This component creates the training labels by first generating labeling functions in a programmable fashion. The labeling functions can be generated in two ways: (1) automatically, and (2) manually. This component is the essence of our framework. The generated labeling functions will capture the experts' domain knowledge and will be scalable, following intuition §3.2 **I1** (capture the operator's domain knowledge to identify events of interest, done in a programmable and scalable way). The labels are expected to be weak or inaccurate. However, with a limited availability of hand-labeled ground truth data created by domain experts as input, a generative model will learn and improve the label accuracy. Thus, learning occurs

from agreement and disagreement of various labeling functions and also from the limited hand-labeled ground truth data [34]. In addition to that, the component is built upon intuition §3.2 **I2** (joining multiple labeling functions enables better learning and promote privacy-preserving collaboration).

This component consists of the following modules:

Automatic labeling function generation. This module takes the training data and their features, validation data and their features, and also the validation ground truth labels, and then assigns a simple classifier (*e.g.,* decision stump) to each row of the training set. Each classifier learns from the feature row associated with the data point, identifies heuristics or rules of thumb (*e.g.,* RTTs with value greater than or equal to 150 ms is considered as noise), and generates a labeling function for that data point based on the heuristics. The process is applied to the entire rows of the training set, creating probabilistic label for each data point.

Note that this module expects users to provide ground truth labels for a subset of the data, which then would become the validation and test sets. The ground truth labels for the validation set is for guiding the simple classifiers in learning the features and associating them with the classes and also for evaluation, while the ground truth labels for the test set is used for evaluation purposes.

Manual labeling function generation. This module provides an interface where users can write labeling functions. It will load the CSV file containing the statistical characteristics of the selected features. In this module the users can define their labeling functions and combine them. The module will take the labeling functions and calculate the dependencies, overlaps, and even any conflict in the labeling functions. The module then uses this information to generate the probabilistic labels for the training set.

The ability to combine labeling functions suits the need for privacy-preserving research collaboration since (a) these labeling functions act as the "code" in the "code-to-data" approach and prevent data leaks, and (b) it opens an avenue to promote research collaboration, as different research groups working on the same type of dataset (*e.g.,* flows, latency measurements) can share their labeling functions to improve their label quality and move closer to reaching a community consensus on features for various events of interests (*e.g.*, detecting noise, DDoS attack). This component is designed to meet requirement §3.1 **R3** (ability to preserve privacy) and follow intuition §3.2 **I2** (joining multiple labeling functions enables better learning and promote privacy-preserving collaboration).

We included the automatic labeling functions generation component alongside the manual component because we realized that as the number of datasets and the diversity of the networking data input increase, experts might still get overwhelmed at the repetitive and laborious process of analyzing and studying new datasets. Thus, the automatic labeling function generation component is meant to substitute the human logic and repetitive process in studying the data and forming the heuristics (patterns, or rules of thumb; *e.g.,* flow packet size greater than or equal to 100 bits is considered a DDoS attack). The addition of the automatic labeling function generation component sets our framework to generate labeling functions in a programmable and scalable fashion. Thus, the automatic labeling function generator seeks to meet requirement §3.1 **R1** (versatility to handle various networking data), and §3.1 **R2** (ability to scale the domain expertise to label data).

We realize that a single labeling function from one feature would produce noisy labels, as these labeling functions would often apply to only *some* data points and not others. For instance, in detecting DDoS attack, one labeling function considers data with packet size less than their 75th percentile to be benign. While this may apply to some data, this categorization cannot apply to *all* data. To get a more generalized conclusion, we need multiple labeling functions.

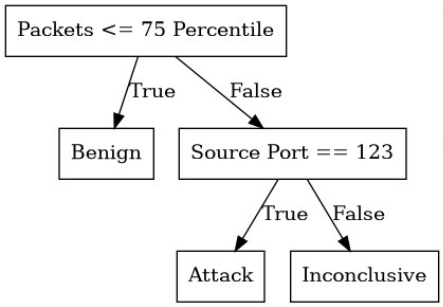
Combining multiple labeling functions (from the same feature and different features) is a useful if not a necessary exercise, as different labeling functions capture different patterns and different angles in analyzing the data. These labeling functions might agree or disagree. The accuracy of the labeling functions are estimated based on the frequency of their agreement and disagreement with other labeling functions. The higher the frequency of agreement, the higher the estimated accuracy, and thus the more weight assigned to this labeling function when it comes to voting for label assignment.

3.3.3 **Label evaluation.** This component takes the training set and the newly generated labels, trains a classifier on this training set, and evaluates the classifier's performance. The component consists of:

End-classifier training. At this stage, the framework trains a classifier on the training set and its probabilistic labels in a supervised learning setting. The default classifier is LSTM, since the majority of the networking data is time series, and we want to preserve the time order of the data as this sequence may carry important information.

End-classifier evaluation. We use F1 score as the evaluation metric. This is because we are interested in knowing our label quality by measuring the number of correct and incorrect classification.

3.3.4 **Dataset-specific interpretability tree generation.** To meet the fourth requirement §3.1 **R4** (interpretability) and to built upon intuition §3.2 **I3**(labeling functions

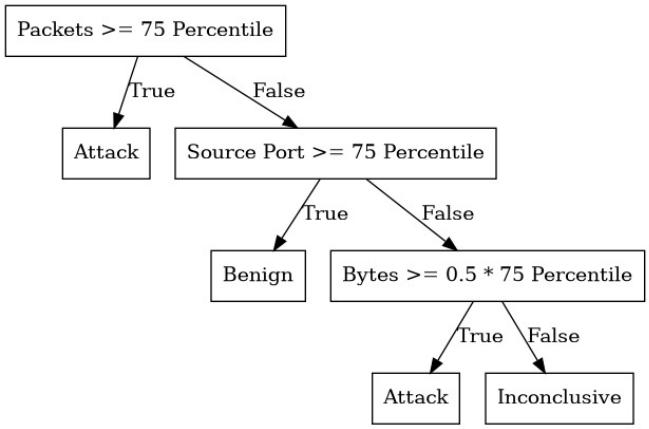



```python
def LF_Packets_75percentile_Benign(x):

    val = x.number2.get_attrib_tokens()
    if float(val[0]) <= df_stats_features.iloc[idx['75%']]['Packets']:
        return 1
    else:
        return 0
def LF_Source_Port_123_Attack(x):

    val = x.number1.get_attrib_tokens()
    if float(val[0]) == 123:
        return -1
    else:
        return 0
```

Figure 3: Example decision tree generated from a combination of two labeling functions for the Worf dataset.




```python
def LF_Packets_75percentile_Attack(x):

    val = x.number2.get_attrib_tokens()
    if float(val[0]) >= df_stats_features.iloc[idx['75%']]['Packets']:
        return -1
    else:
        return 0
def LF_Source_Port_75percentile_Benign(x):
    val = x.number1.get_attrib_tokens()
    if float(val[0]) >= df_stats_features.iloc[idx['75%']]['Source Port']:
        return 1
    else:
        return 0
def LF_Bytes_Half_75percentile_Attack(x):

    val = x.number3.get_attrib_tokens()
    if float(val[0]) >= (0.5 * df_stats_features.iloc[idx['75%']]['Bytes']):
        return -1
    else:
        return 0
```

Figure 4: Example decision tree generated from a combination of three labeling functions for the Worf dataset.

can be expanded to trees), FLAMENCO will map the user-defined labeling functions to trees, which will serve as visual representations of the user-defined labeling function(s). The tree's nodes contain a boolean statement of a feature and a threshold (*e.g*, packet length >= 150, bytes <= mean + sd) and the edges are the "True" and "False" answers to the boolean statements. With these trees, users can trace how our framework analyzes a data point and categorizes it to a certain class. The decision-making process displayed by the tree is meant to provide transparency on why the data points with certain feature and value are categorized in one class, while other data points in another class.

Note, however, that these trees are *not* the decision tree classifiers. These dataset-specific interpretability trees simply visualize the path that a data point takes to arrive at a classification decision *according* to the user-defined labeling functions. In contrast, the decision tree classifiers follow no user-defined functions to arrive at a decision, as the path taken is determined by the decision tree algorithm. Thus, our framework's dataset-specific interpretability trees are user-driven, while the decision tree classifiers are algorithm-driven.

Examples of the dataset-specific interpretability trees are shown in Figures 3 and 4. In both figures, we see the labeling functions and the mapped features and values on the tree nodes. Class 1 denotes benign cases, while -1 denotes attack cases. Each level of the tree represent one feature and the node hierarchy is determined by feature importance. In the example depicted in Figure 3, "Packets" is at the root because it has the highest feature importance score, while "Source Port" is placed at level 2 in the hierarchy, since its feature importance score is less than that of "Packets". In Figure 4, we see that the tree contains three decision nodes that are located at different level. Here, "Bytes" occupies level 3, since it has a lower feature importance score than "Source Port".

3.4 Implementation

This section describes the end-to-end workings of the framework. While §3.3 describes the components and the modules, this section describes how the data move from the input for preprocessing to the end-classifier evaluation. We will present the data flow in two paths: (a) when the user opted for the automatic labeling function generation, and (b) when the user opted for the manual labeling function generation.

Also, note that the current implementation of FLAMENCO only supports binary classification.

3.4.1 **Automatic labeling function generation.** This path starts with **data preprocessing** that includes modality detection, synthetic data creation, threshold calculation, and feature augmentation. The current implementation of the component expects numerical data with *one* feature column. In data preprocessing, the data will undergo ***modality detection*** where the framework will either split the data into different subsets, or output the same dataset without splitting. Note also that for latency measurement data, the framework will separate the data into different source-destination (SD) pairs datasets.

After this, the user can ***calculate the threshold value*** that can be used to separate the positive class from the negative class. The framework will then check the size of the two classes in the dataset (or each subset). If the classes are imbalanced, the user can then ***create synthetic data*** to balance the two classes and combine the synthetic data with the original data. We used *pandas* and *numpy* for the entire data preprocessing stage.

Next, the data are passed to the data loader module where they undergo ***feature augmentation***. At this step, the framework will calculate the data's statistical features using *tsfresh* [9], and create a matrix for these features. Following this, FLAMENCO will split the data and the features into test (10 percent), validation (25 percent), and training sets (the rest), and create ground truth labels for the validation and test sets.

Since this pipeline is an ***automatic labeling function generation***, FLAMENCO will employ simple classifiers, which, in the current implementation are decision stump, and take the training data and the features, and have the classifiers learn from the validation set, the validation set features, and also the validation ground truth labels (that the user needs to provide). After learning, the simple classifiers will generate heuristics that are turned into labeling functions, and finally produce probabilistic labels for the training data. We implemented the automatic labeling function generator using Snuba [43].

Finally, with the labeled training set, the framework will ***train a classifier***. We used LSTM due to the time-series nature of our data and we wanted to preserve the time order. We used Adam optimizer (beta1=0.9, beta2=0.999) and ***fine-tuned the hyperparameters*** by trying out different values for learning rate (0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01, 0.03, 0.05), number of epochs (20, 25, 30), batch size (32, 64, 128, 256), and number of LSTM cells (32, 64, 128). We apply the following regularization methods to guard against overfitting: L2 regularization (between 0.2 and 0.6), dropout (between 0.2 and 0.6), and early stopping (minimum delta=0.01, patience=2) that monitors the validation loss. Following training, FLAMENCO will ***evaluate the performance*** by measuring the F1 score. We used F1 score as our evaluation metric because we are interested in knowing the balance between precision and recall.

At any point after labeling function generation, the user can ***plot a dataset-specific interpretability tree*** that is based on the labeling functions that the simple classifiers formed. The tree is unique for each dataset, *e.g.,* one SD-pair dataset will generate one tree, one flow dataset will generate one tree.

3.4.2 **Manual labeling function generation.** For the manual labeling function generation pipeline, the process also begins with **data preprocessing**. After this, the framework will calculate the correlation coefficient of the features, and let the user decide on which ***features to select***. FLAMENCO will then create a new pandas DataFrame with only the selected features, and calculate the statistical profiles of each feature column.

The new data are then split into test (20 percent of total data), validation (20 percent of the 80 percent of data after test data are taken), and training (the rest of the data after taking the data for validation set). We used the stratification method to split the data so the ratio of the classes is the same in the training, validation, and test sets, and then order the data based on time. We then label the validation and test sets and create CSV files for each set, and also write the training set into a TSV file. We employed NoMoNoise [28] to implement our manual labeling function generation. Based on these user-written labeling functions, NoMoNoise will generate weak or low-quality labels and connect to Snorkel [33] library to produce generative models with accuracies in the form of probabilities or confidence values.

Following this, the process is the same as in supervised learning. The user can ***train a classifier*** using the training set and the newly generated labels, and ***evaluate the performance*** using the F1 score. As in the automatic labeling function generation pipeline, the default classifier is LSTM, although users are at liberty to choose other classifiers. We fine-tuned the hyperparameters by trying out different values of learning rate (0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01, 0.03, 0.05), number of epochs (20, 30, 50), number of LSTM cells (128, 256), and batch size (64, 128, 256). We applied regularization methods to guard against overfitting: L2 regularization (between 0.2 and 0.6), and dropout (between 0.2 and 0.6).

As in the case of automatic labeling function generation, at any point after labeling function generation, the user can **render a tree** that visualizes the decision-making process.

The tree will be based on the labeling functions that the user writes. Each dataset will produce its own tree.

4 EVALUATION

The overall goal of this evaluation is to demonstrate how our framework solve the problem of training label scarcity in network data. To achieve this overall goal we will show that our framework has the ability to do the following:

- Handling network data types (latency measurements and NetFlow) that differ in methods of collection and granularity, and thus fulfilling §3.1 **R1**,
- Creating training labels programmatically and in a scalable way, fulfilling §3.1 **R2**,
- Promoting research collaborations while avoiding the risk of privacy leak, fulfilling §3.1 **R3**,
- Providing explanations on the decision making process, fulfilling §3.1 **R4**.

4.1 Case Study 1: CAIDA's Ark

Overview. The goal of this experiment is to demonstrate that our framework readily process latency measurement data and create high quality training labels in a scalable process that can cover more datasets. In §2.2 recall that ***L1*** and ***L2*** discuss the limitations of the state of the art approach; they are unable to handle diverse networking data and they are unable to scale the domain expertise. Corresponding to these limitations, in §3.1 we propose that the solutions to address ***L1*** and ***L2*** should meet Requirements ***R1*** and ***R2***. ***R1*** describes that the solution must be versatile to be able to process various networking data, while ***R2*** describes that the solution must have the ability to scale the domain expertise in creating labels. These experiments are designed to meet these two requirements.

Dataset. Our traceroute data consists of a day's worth of measurements from five different vantage points divided into 25 source-destination (SD) pairs datasets (five SD pair datasets from each vantage point) and amounting to 466,061 data points. The size of the SD pair datasets ranges from 2,000 to 110,000. We intentionally look for traceroute data with diverse statistical characteristics such as different modalities, various values of mean, standard deviation, and quartiles. Figure 5 shows the boxplots of two SD pairs datasets, which represent some of the unusual statistical characteristics of the RTT values in the 25 SD pairs datasets. On the left image, we see that the minimum value, the 25th percentile, the median, the 75th percentile, and the maximum value are squashed together, and the outliers have a wide variety of values. The middle image shows a dataset where the minimum value and the 25th percentile overlap, a median value not far from the minimum and the 25th percentile, and a large distance between the median and the 75th percentile.

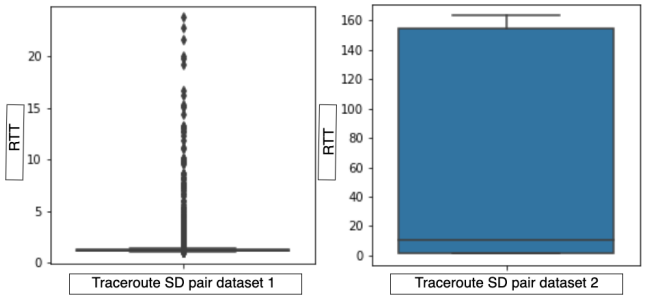


Figure 5: Unusual characteristics among our 25 SD pairs CAIDA's Ark datasets.

From these 25 SD pairs datasets, 12 datasets are either bimodal or multimodal, with the total size of 58,201 data points, and the other 13 datasets are unimodal. Figure 6 shows the bimodality and multimodality that we found in our data.

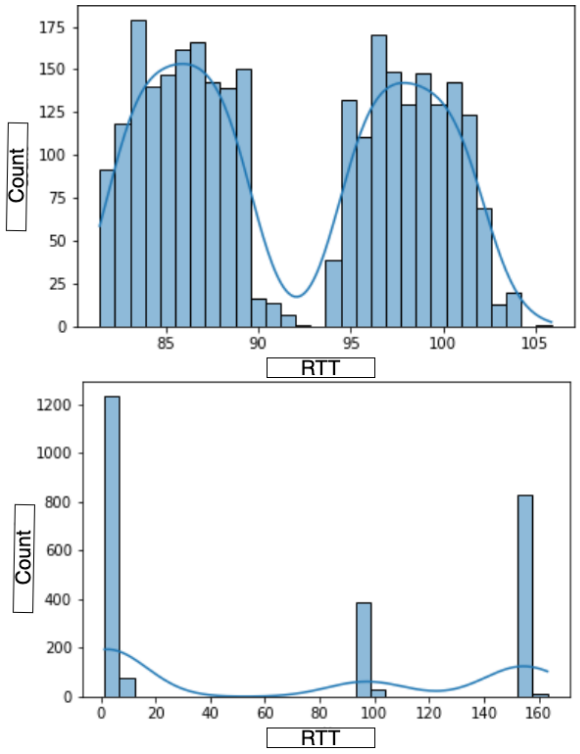


Figure 6: Bimodality and multimodality in our CAIDA's Ark datasets.

Experiments. The goal of the experiment is to demonstrate the efficacy of the framework in generating training labels. Our event of interest is noise detection. The good data class is the positive class, while the noisy data class is the negative class. Since the data are unlabeled, and since noise

in the RTT measurement data can be considered outliers or anomalies, we chose to compare the quality of the labels generated by our framework with those generated using unsupervised learning methods commonly used in outlier detection and anomaly detection. The methods that we considered are: (1) Local Outlier Factor (LOF), a method that measures the local density of a data point, compares this density with the local density of its neighbors, find density similarities and differences in the regions, and identify the lower density data points as outliers; (2) Elliptic Envelope (EE), an outlier detection method that analyzes the whole dataset, draws an imaginary ellipse around the values based on specified criteria, and identifies the values outside the ellipse as outliers; (3) Overly-Robust Covariance Estimation, an outlier detection method that identifies data points whose covariance has the smallest determinant to be put in a "pure" data subset; (4) Isolation Forest (IF), an anomaly detection method that employs random forests to perform recursive partitioning of the randomly selected features and identifies the shortest paths from the root to the terminal node as anomalies.

Label generation. In this experiment we used the automatic labeling function generation pipeline. We preprocessed, oversampled, and split the data into test (10 percent), validation (25 percent), and training (65 percent) sets. We then calculated statistical features and labeled the validation and test sets. Next, we passed the data to the automatic labeling function generator and produced training labels. Next, we also produced training labels using the four unsupervised methods (LOF, EE, ORCE, IF). At this stage, we had five different training label sets that were produced by five different methods (our framework and the four unsupervised methods.)

End-classifier training and evaluation. Following label generation, we trained LSTM models on the training set, and evaluated them first on the validation set to find the optimal hyperparameter setting, and then evaluated them on the test set. We obtained the F1 score from this process. This F1 score represents the label quality that our framework generated. Using the same training set, we employed the unsupervised learning methods to generate labels, and used these labels as the training labels. We then trained LSTM models on these training data and the generated labels, evaluated them on the validation set and fine-tuned them, and lastly we evaluated the LSTM models on the test set. The hyperparameter setting differs for each method and for each SD pairs dataset. We used Adam optimizer (beta1=0.9, beta2=0.999) and tried out various values to fine-tune our hyperparameters: learning rate (0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01, 0.03, 0.05), number of epochs (20, 25, 30), batch size (32, 64, 128, 256), and number of LSTM cells (32, 64, 128). We applied the following regularization methods to avoid overfitting: L2 regularization (between 0.2 and 0.6), dropout (between 0.2 and 0.6), and early stopping (minimum delta=0.01, patience=2) that monitors the validation loss. We did not include the exact hyperparameter setting here because each dataset has different hyperparameter setting. Lastly, we averaged the F1 scores of the 25 SD pairs datasets for each labeling method. At this step, we obtained average F1 scores of LOF, EE, ORCE, IF, and FLAMENCO.

Results. Figure 7 shows the F1 scores of the LSTM trained using the labels generated by LOF, EE, ORCE, IF, and FLAMENCO. From this result, we see that the unsupervised labeling methods achieved less than 0.5 score, while our framework achieved more than 0.9 score. The unsupervised methods work by finding similarities and differences between the data points, and grouped the data based on these similarities. Since these methods rely on finding similarity and difference between data points to classify, and make prior assumption that the data conforms to a normal distribution, they might have found a lot of variability in the data and formed many groups based on the similarity, which would be unsurprising since the data had high statistical variability and non-normal distribution. On the other hand, our framework makes no assumption about the data distribution, and instead analyzes the data first to detect any multimodality that can skew the profile, and, with the simple decision stump algorithm embedded in the automatic labeling function generator, our framework was able to recognize the patterns in the data and determine the decision boundary to classify the data.

Dataset-specific Interpretability Tree. The root of the tree (not shown) is a condition checking if the RTT value of the incoming data point is greater than a threshold value. This value is taken from the decision boundary that the automatic labeling function generator discovers. Any RTT value greater than this threshold is considered noise, while those less than or equal to this value is considered good data.

Summary. This experiment goal is to show the efficacy of our framework in handling traceroute data to create training labels for noise detection. We compared our label quality with labels created using unsupervised methods. Our framework is designed to handle data with multimodal distribution. The LSTM models that are trained using the labels created by our framework achieved an F1 score that is up to 0.48 points higher than the F1 scores of the LSTM trained using unsupervised methods' labels.

4.2 Case Study 2: RIPE Atlas

Overview. The goals of this experiment are similar to those of the experiments on CAIDA Ark. We seek to meet the requirements §3.1 ***R1*** (versatility) and §3.1 ***R2*** (employing scalable methods to create training labels). However, in this

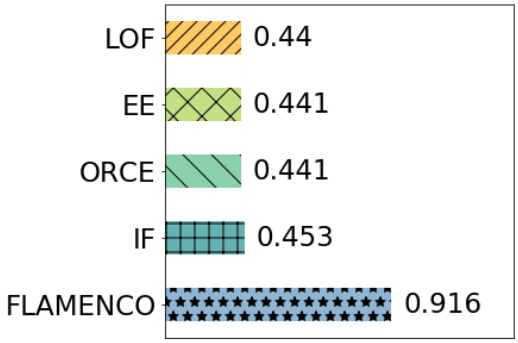


Figure 7: The average F1 scores of the LSTM models trained using five methods on CAIDA's Ark's 25 SD pairs datasets.

experiment we used a different type of latency measurement data, that is, ping data.

Dataset. Our RIPE Atlas dataset [5] is collected between the time period 2017 and 2021 from 25 ping measurements raw datasets, with each dataset containing 10,000 probes. From each dataset, we grouped the RTT data based on the source-destination addresses and include the RTT data with the same source-destination address into one dataset. While there are over 7,000 unique source-destination pair, we found that the most RTT values between this source-destination addresses are below 100 data points. We took only the source-destination pairs that have over 100 data points, which are only 25 pairs. These 25 pairs become their own dataset with size ranging from 100 to 450 RTT data points. Most of the data follow unimodal normal distribution, although some exhibit multimodalities. Figure 8 shows two multimodalities found amoung our RIPE Atlas datasets.

Experiments. Like in the case of CAIDA Ark's, the goal of this experiment is to demonstrate that our framework produces high quality training labels. The experiment procedure is similar to CAIDA Ark's, with a slight variation in the end-classifier training. Our event of interest is to detect noise in the ping measurement data, with noisy data labeled as the negative class and good data labeled as the positive class. Also, our framework's generated training labels are compared with the labels created by the EE, ORCE, IF, and LOF methods.

Label generation. The experiment is done using the framework's automatic labeling function generation pipeline. Recall that "automatic" in this context refers to the substitution of human logic with simple classifiers to identify rules for labeling function. The first step in this pipeline is preprocessing the data, followed by checking the modality. We set our modality detector to look for peaks with minimum height of 10 percent of the maximum height. We found that although our ping datasets appeared to have multimodalities

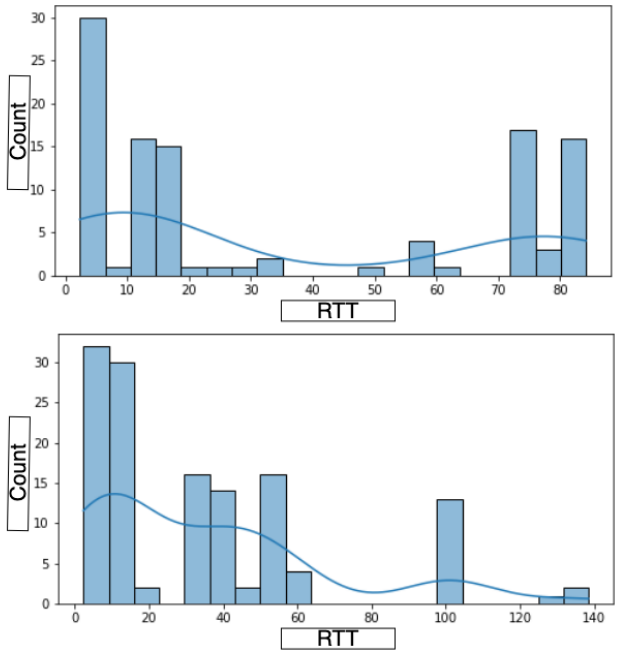


Figure 8: Multimodality in our RIPE Atlas ping dataset.

such as depicted in Figure 8, many of those smaller peaks had heights less than 10 percent of the maximum height, which in many SD pairs datasets, reduced the modalities to either two (bimodal) or one (unimodal). We separated the datasets into subsets (in the case of non-unimodal datasets), and oversampled the data due to the large imbalance between the two classes. We split the data into test (25 percent), validation (20 percent), and training (55 percent). The reason for the different split on this dataset is because our ping measurements dataset is small, with each SD pair dataset amounting to around 100 to 500 data points. When the framework found bimodality or multimodality in the data, it will split the dataset into subsets before splitting for training, validation, and test sets. The framework treats each subset as a unimodal dataset and will split it into training, validation, and test data. This process causes the size of the subsets to be even smaller, with some subsets having the size of 8 data points.

After splitting the data into training, validation, and test sets, we calculated the statistical features of the data, and labeled the validation and test sets. Then we passed the data to the automatic labeling function generator, which then produced training labels for our training sets. We also produced training labels using the EE, ORCE, IF, and LOF methods, so now we had the training labels produced by five different methods.

End-classifier training and evaluation. Since the size of our ping measurement SD pairs datasets is extremely small, we employed a different end-classifier training strategy. While we used LSTM for our end-classifier in the CAIDA Ark's experiments, we found that deep learning models are too complex for small datasets such as our ping SD pairs datasets, and this caused a major overfitting issue. To avoid overfitting, we decided to use a simpler classifier. After trying k-nearest neighbors, logistic regression, and random forest, we settled on using random forest because it generated the best results.

We trained the data using cross validation since it would allow us to utilize as many data points as possible for training, since we had so few data points. Before we performed cross validation, we combined the training and validation sets and also their corresponding labels, and left the test set and the test ground truth labels intact, since we did not do cross validation on the test set. This resulted in our combined training and validation set having 75 percent of the total data points, and the test set having 25 percent. For each dataset (or subset), we had five different training label sets, one for each method by which they were generated.

Since cross validation techniques are rarely used on time series data due to their rearrangement of the row order, we used *scikit-learn*'s *TimeSeriesSplit* method [31], which allows the cross validation approach while preserving the time order. The *TimeSeriesSplit* works like KFold cross validation; however, instead of shuffling the indices when creating the folds, *TimeSeriesSplit* created validation indices that are always greater than the training indices, thus preserving the time order. In each cross validation iteration, we also employed hyperparameter tuning using randomized grid search, which sought the optimal values for the number of trees (50, 100, 150, 200), maximum depth (2, 5, 10, None), and whether or not to use bootstrapping (True or False). We used the best parameters as the hyperparameter setting when we evaluated the model on the test data. Lastly, for each method, we averaged the F1 scores of the subsets of the same dataset to represent the F1 score of that dataset, and then averaged the F1 scores across the 25 SD pairs RIPE Atlas ping datasets.

Results. The final F1 scores are shown in Figure 9. In this experiment, the F1 score of the classifiers trained using our labels is the highest, followed by EE and IF, whose F1 scores are also above 0.9. The gap between the F1 score associated with our framework and those with the unsupervised methods is around 0.05. The reason for the much better performance of the unsupervised methods might be the size of the datasets, in which the number of multimodal data is not high enough compared to the unimodal data, which these four methods would be able to handle well.

Dataset-specific Interpretability Tree. As in the case of the previous experiments on CAIDA's Ark, each tree (or stump) represents one SD pair dataset. Since in this latency measurement data we only consider the RTTs, the visualization tree has only one level, with the root of the tree (not shown) is a conditional statement and the children are the next node when the condition is met. The conditional statement at the root checks if the RTT value of the incoming data point is greater than a threshold value. This value is taken from the decision boundary that the automatic labeling function generator discovers after learning. Any RTT value greater than this threshold is considered noise, while those less than or equal to this value is considered good data.

Summary. The goal of this experiment is to show the efficacy of our framework in handling ping datasets. We compared the label quality generated by our framework with the labels generated by four unsupervised methods (ORCE, LOF, IF, EE). The F1 score of the classifiers trained using our framework's labels is 0.996, which is higher than the F1 scores associated with the other four methods, although the gap between our framework's F1 score and that of the best performing unsupervised method is small.

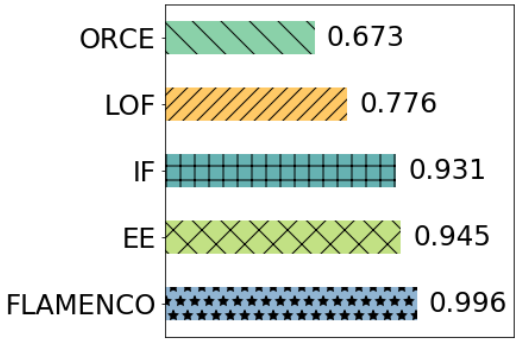


Figure 9: F1 score comparison of classifiers trained using labels generated by ORCE, LOF, IF, EE, and FLAMENCO on RIPE Atlas' 25 SD pairs datasets.

4.3 Case Study 3: CIC-DDoS2019

Overview. The goal of the experiments on this dataset is to meet requirement ***R3***, that is, ability to preserve privacy. As discussed, we avoid the risk of privacy leak by using the "code-to-data" approach, with labeling functions being the "code" in this context. The experiments on this dataset, then, aim to demonstrate how our framework supports user-defined labeling functions and the effects of combining labeling functions with different features and feature correlations on label quality. Moreover, since the data in this dataset are flow data, we also demonstrate that our framework is readily able to handle another network data type besides latency measurement.

Dataset. In the last six months of 2020, it is reported that the DDoS attack frequency sharply increased by 22 percent, with the total of 10,089,687 attacks in 2020, and more than 800,000 attacks per month [20]. The same report, coupled with another observation in the first quarter of 2021 [19], stated that the increase of the attacks was incited by several new UDP-based reflection/amplification DDoS vectors. With this DDoS vector trend in mind, we decided to conduct our experiments on a UDP-based amplification dataset, which in this case is an NTP dataset, one of the 12 different DDoS vectors available in the CIC-DDoS2019 dataset.

The CIC-DDoS2019 dataset was collected on January 12th, 2019 and included results from CIC FlowMeter-V3 [23], a network traffic flow generator and analyzer. The dataset has 80 features including timestamp, source and destination IPs and ports, protocols, statistical characteristics of the forward and backward packet lengths, and ground truth labels indicating benign and attack cases. This dataset is a network traffic flow dataset that provides examples for benign data and the 12 different types of DDoS attacks that can be carried out on the application layer protocols using TCP/UDP, with each attack type stored in different CSV file. The NTP dataset we chose has a size of 645 MB and 1,217,007 data points.

Considering the lack of agreement in the community to describe certain events of interest (§2.2 L1), which in this case is DDoS NTP attacks, we submit that collaboration between research groups to find common features of DDoS NTP attack is crucial in the effort of reaching certain level of agreement. To account for the privacy leak risk in a collaboration setting, we design this experiment to (1) show that FLAMENCO can support privacy preserving collaboration and (2) investigate the combination of various labeling functions. Concretely, we propose a scenario where two different research groups are working together to find features of DDoS NTP attacks that are common in their respective NetFlow data. We investigated the kind of labeling function combination that can benefit both groups and also the apparent effects of correlation coefficient on the label quality.

To achieve the experiment goal, we divide the experiment into two parts. In the first part, our scenario assumes that both research groups are using the same feature (*e.g.,* source port, packet size) to write a labeling function, but each group implements a different threshold to determine whether an event is an attack or not. In most cases, each group would apply different threshold values, as their datasets are more likely to have different statistical features.

In the second part, the two research groups use two different features with two different thresholds in their respective experiments. In this scenario, the first group uses two features that are correlated with each other, while the second group uses two features that have little or no correlation with each other. Here, we aim to investigate the impact of correlation coefficients on label quality.

In both of the two experiments, we assess the labeling function quality directly as well, before using the labels with the training set and train a classifier with them.

Data preparation. To clean the data, we take the DDoS_NTP dataset from CIC DDoS 2019, order the data based on the timestamp, and remove the NaN, infinity, and duplicate values. Since the data are imbalanced, we split the data into training, validation, and test sets using the stratification method, ensuring that all three sets have the same ratio of positive and negative classes. The total number of examples in the NTP dataset is 228,340.

Feature selection. We selected three out 80 features according to two sources: (1) a previous study on the DDoS NTP victim profiles [12], and (2) the correlation coefficients. Among the features investigated in [12], we calculated the correlation coefficients between each feature, and chose three features; the first feature is correlated with the second feature, while the third feature is not correlated with the first or the second feature. Table 1 shows the feature name, description, and correlation coefficents of the three features. “Source Port" and “Average Packet Size" have a correlation coefficient of -0.477, “Source Port" and “Flow IAT Min" have correlation coefficient of 0.001, and “Average Packet Size" and “Flow IAT Min" have correlation coefficient of 0.003. The correlation values indicate that “Source Port" and “Average Packet Size" are moderately correlated, while both these two features have little correlation with “Flow IAT Min". The statistical characteristics of the features are listed on Table 2.

Label generation. According to the study done by [12], 36.2% of NTP attacks targeted port 80, 23.8% targeted port 123, 7.9% targeted port 3074, and the other 32% targeted various ports ranging from 19 to 50557. Attacks on ports less than 80 amount to 4.8%, between ports 80 and 123 amount to 0.4%, between ports 123 and 3074 amount to 0.7%, and greater than port 3074 amount to 11.1%. Since the datasets used in that study are different than our DDoS2019 synthetic data, we did not take the exact port numbers as threshold values. Although port number is a universal feature, the percentage of attacks that targets certain ports in [12] would differ than the percentage of attacks on those ports that might occur in our data. This is because the data used in [12] are different than our data. Thus, the frequency of the attack, and also the percentage of attacks that target certain ports would also be different. Instead of taking the exact values that [12] listed, we started with those values and experimented with the values around the thresholds in the paper. We settled with port number <= 6600 to detect attack events, and port number >= 9000 to detect benign events. We used 100 as the threshold for our second feature (“Average

Selected Feature	Description	Corr. with Source Port	Corr. with Average Packet Size	Corr. with Flow IAT Min
Source Port	Source port	1.0	-0.477	0.001
Average Packet Size	Average size of packet	-0.477	1.0	0.003
Flow IAT Min	Minimum time between two packets sent in the flow	0.001	0.003	1.0

Table 1: Selected DDoS NTP features, description, and correlation.

Statistical Feature	Source Port	Average Packet Size	Flow IAT Min
count	228340.000000	228340.000000	2.283400e+05
mean	3858.588740	424.622116	3.059715e+03
std	12607.206688	110.243079	2.778376e+05
min	0.000000	0.000000	0.000000e+00
25%	644.000000	436.666667	0.000000e+00
50%	781.000000	442.558140	0.000000e+00
75%	918.000000	447.600000	1.000000e+00
max	65532.000000	4025.778588	6.553614e+07

Table 2: Statistical characteristics of the selected features from the DDoS NTP dataset

Packet Size") to detect benign events, as 100 is much smaller than the mean of "Average Packet Size", which is 424.62. Lastly, we also used the statistical characteristics of "Flow IAT Min" to determine the threshold value. In this case, we used the mean as the threshold to determine attack events. We coded these features and their corresponding thresholds to distinguish the attack cases from the benign cases in our labeling functions. Using these labeling functions (individual and combined), we generated the training labels. We also directly measured the label quality of these labeling functions using the F1 score.

End-classifier training and evaluation. We chose LSTM as our end-classifier since our dataset is a time-series dataset and we wanted to preserve the time order in the data. We trained the LSTM on the training set using the probabilistic labels that were generated in the previous step.

Since our data are extremely imbalanced, we place more weight for the minority class in the loss function of our LSTM model. The weight is calculated as the ratio of the number of the majority class to the number of the minority class. We fine-tuned the hyperparamters by trying out different values of learning rate (0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01, 0.03, 0.05), number of epochs (20, 30, 50), number of LSTM cells (128, 256), and batch size (64, 128, 256). We also applied regularization methods to avoid overfitting: L2 regularization (between 0.2 and 0.6), dropout (between 0.2 and 0.6). We did not include the exact hyperparameter setting here because each labeling function and their combinations have different hyperparameter setting.

Results. The following listed the results of both experiments.

4.3.1 ***Experiment 1: Combination of labeling functions that are based on one feature.*** Table 3 shows the individual F1 scores and the combined F1 score for the first experiment. We chose "Source Port" to be the feature on which we write labeling functions because "Source Port" has the highest feature importance score. Inspired by a previous study [12], we chose the values 9000 and 6600 as the thresholds to determine the classes; any "Source Port" value greater than 9000 will be considered as a benign case, while any "Source Port" value less than 6600 will be considered an attack case.

In Table 3, we see that the F1 score of the classifier trained with the first labeling function is 0.893, while the second one is 0.849, and the combination is 0.897, which is higher than both the individual scores. Note that although the labeling functions are based on the same feature, each classifies for a different class; the first labeling function classifies for the attack cases, while the second classifies for the benign cases. This could happen because by using different thresholds on the same feature to detect attack and benign cases, we have expanded the coverage of the labeling functions; covering values less than 6600 and also greater than 9000, although admittedly there are some values between 6600 and 9000 that are not covered, and the values that fall in this range are

considered inconclusive. The improved F1 score of the combined labeling function suggests a benefit of two research groups working to detect DDoS attacks using the same feature. In this example, the labeling functions have different thresholds that expand the labeling coverage of both functions. While we are far from claiming that our proposed privacy-preserving collaboration platform for "code-to-data" approach *will* result in improved F1 score, we have demonstrated that our proposed solution *can* improve the F1 score, and by extension, can improve the label quality and thus moves us closer to a consensus on agreed DDoS NTP features.

Here we also present the F1 score of the labeling functions (referred to as LF F1), which is different from the classifier's F1; while the F1 score of the labeling functions directly assesses if *the labels generated by the labeling functions* match the ground truth, the F1 score of the classifier assesses if *a classifier trained using the labels and the training set* matches the ground truth. "LF F1" indicates the F1 score of the labeling functions, and "Classifier F1" indicates the F1 score of the trained classifier. Note also that the labeling function's direct performance and the trained classifier's performance are evaluated using the test set.

In Table 3, we see that LF F1 of the first and second labeling functions are higher than the classifier's F1 score. While the difference for the first labeling function is only 0.016, the difference for the second labeling function is 0.7977. The much higher LF F1 compared to the classifier's indicates that LF does not generalize well. This is because when the labels are used on the training set to train a classifier and are evaluated on the test set, the F1 score is lower. This difference may be because the two labeling functions are written to detect only one class; one is for attack cases, while another is for benign cases, making the labeling functions one-sided. This could be the cause of the bias in LF. This explanation also fits the result of the combined LF F1, which is 0.899, compared to the classifier F1, which is 0.897. As the combination of these two labeling functions gives a perspective from both the attack side and the benign side, it generates a more generalized result.

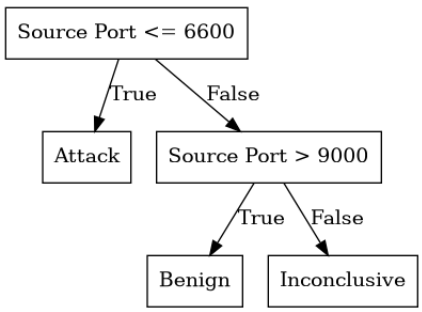
LF	LF F1	Classifer F1
LF_Source_Port_Less6600_Attack	0.899	0.877
LF_Source_Port_Greater9000_Benign	0.899	0.0963
Combined	0.899	0.897

Table 3: The LF F1 and classifier F1 scores of the labeling functions that are based on one feature of the DDoS 2019 NTP data. The values 9000 and 6600 are inspired by a study done by [12] with some adjustments.

Dataset-specific Interpretability Tree. Based only on these labeling functions, FLAMENCO can generate a tree as shown in Figure 10. The feature and value in each labeling function are mapped to the decision nodes of the tree. In most cases, FLAMENCO will generate a tree where one level represents one feature, but in this case, since the labeling functions are for different classes, we see two decision nodes located at different levels of the tree. By inspecting the decision tree, a data point with "Source Port" value of 123 (the dedicated port for NTP), will be categorized as an attack, a data point with "Source Port" 10000 will be categorized as benign, and a data point with "Source Port" 8888 will be considered inconclusive.

In our effort to directly assess the label quality, we also compared the labels generated by the interpretability tree (Tree F1) and the labels generated by the manual labeling function generator (LF F1). We calculated the Tree F1 score by taking each data point from the test set and traversing the tree until the data point reaches a terminal node, which determines the label for that data point. We compared this label with the ground-truth label from the test set. Then we counted the number of true positives, true negatives, false positives, and false negatives, calculated precision and recall, and finally the F1 score. As we see in Table 3, the F1 score of the combined "Source Port" labeling functions is 0.899. This F1 score matches the calculated F1 score of the interpretability tree, which is 0.894. This result seems to be encouraging that the tree's F1 score is consistent with the manual labeling functions's F1 score.

4.3.2 ***Experiment 2: Combination of labeling functions that are based on two features.*** Table 4 shows the individual F1 scores and the combined F1 scores of the labeling functions written using two features. The first row shows the classifier F1 score of LF_Avg_Pkt_Less_100, which is 0.09. Note that 100 here is not based on any statistical feature of the data since it is arbitrarily chosen so its F1 score is low. The second row shows the classifier F1 score of LF_Source_Port_Less_6600_Attack, which is 0.877. We chose these two labeling functions (one is low and another is high) because we wanted to see the effect of combining labeling functions with a vast difference of classifier F1 scores. The third row shows the F1 score of the combined labeling functions, which is higher than the individual F1 scores. Notice that the correlation coefficient between "Average Packet Size" and "Source Port" is -0.477, which is considered moderate. These results suggest that combining two labeling functions that are based on correlated features can improve the classifier F1 score.




```
def LF_Source_Port_Less6600_Attack(x):
    val = x.number1.get_attrib_tokens()
    if float(val[0]) <= 6600:
        return -1
    else:
        return 0
```

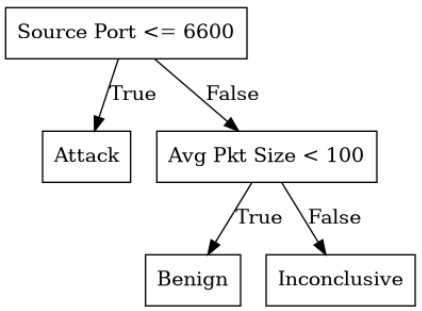
```
def LF_Source_Port_Greater9000_Benign(x):

    val = x.number1.get_attrib_tokens()
    if float(val[0]) > 9000:
        return 1
    else:
        return 0
```

Figure 10: The dataset-specific interpretability tree generated from one-featured labeling functions (“Source Port") of the DDoS2019 dataset. The F1 score of the tree is 0.894, and the LF F1 is 0.894.

First LF F1	Second LF F1	LF F1	Trained Classifier F1	Correlation
LF_Avg_Pkt_Less100_Benign	-	0.00883	0.09	-
LF_Source_Port_Less6600_Attack	-	0.893	0.877	-
LF_Avg_Pkt_Less100_Benign	LF_Source_Port_Less6600_Attack	0.893	0.893	-0.477
LF_IAT_Min_Less_Mean_Attack	-	0.893	0.892	-
LF_Avg_Pkt_Less100_Benign	LF_IAT_Min_Less_Mean_Attack	0	0.249	0.003
LF_Source_Port_Less6600_Attack	LF_IAT_Min_Less_Mean_Attack	0	0.863	0.004

Table 4: The F1 scores of the labeling functions and the trained classifiers based on two features of the DDoS 2019 NTP data.




```
def LF_Source_Port_Less6600_Attack(x):
    val = x.number1.get_attrib_tokens()
    if float(val[0]) <= 6600:
        return -1
    else:
        return 0
```

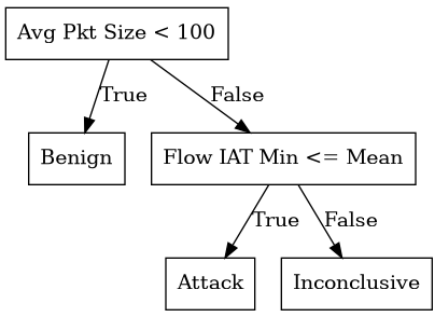
```
def LF_Avg_Pkt_Less100_Benign(x):

    val = x.number2.get_attrib_tokens()
    if float(val[0]) < 100:
        return 1
    else:
        return 0
```

Figure 11: The dataset-specific interpretability tree generated from labeling functions based on “Source Port" and “Average Packet Size". The F1 score of the tree is 0.897, while the LF F1 is 0.893.

Looking at the LF F1 of the labeling functions and comparing them to the classifier F1, we see a difference of 0.08 for LF_Avg_Pkt_Less_100_Benign, a difference of 0.016 for LF_Source_Port_Less_6600_Attack, and a difference of 0.01 for LF_IAT_Min_Less_Mean_Attack. The combination of LF_Avg_Pkt_Less_100_Benign and LF_Source_Port_Less_6600_Attack generates the same LF F1 and classifier F1, and the correlation between the features is -0.477. This could indicate that correlation affects the labeling function quality, which is consistent with the results of the classifier F1 scores.

Next, we investigated the effect of combining labeling functions that are based on two uncorrelated features. The fourth row of Table 4 shows the classifier F1 score of LF_Flow_IAT_Min_Less_Mean, which is 0.892. The correlation coefficient between “Flow IAT Min" and “Average Packet Size" is 0.003, which is so low that the two features can be considered uncorrelated. Combining LF_Flow_IAT_Min_Less_Mean with LF_Avg_Pkt_Less_100 shows a classifier F1 score of 0.249, as shown in second to the last row of Table 4. This score is a major decrease compared to the classifier score of LF_Flow_IAT_Min_Less_Mean (0.892), although it is slightly




```
def LF_Avg_Pkt_Less100_Benign(x):

    val = x.number2.get_attrib_tokens()
    if float(val[0]) < 100:
        return 1
    else:
        return 0
```

```
def LF_Flow_IAT_Min_Less_Mean_Attack(x):

    val = x.number3.get_attrib_tokens()
    if float(val[0]) <= df_stats_features.iloc[dict_stat_features_index['mean']][' Flow IAT Min']:
        return -1
    else:
        return 0
```

Figure 12: The dataset-specific interpretability tree generated from labeling functions based on “Average Packet Size" and “Flow IAT Min". The F1 score of the tree is 0.728, while the LF F1 is 0.

higher than that of LF_Avg_Pkt_Less_100 (0.09). For this one case, the results seem to suggest that combining two labeling functions that are based on two uncorrelated features could improve the lower-scored labeling function, while weakening a higher-scored one.

However, when we look at the LF F1 of the combination of LF_Avg_Pkt_Less_100_Benign and LF_IAT_Min_Less_Mean_Attack and also the LF F1 of the combination of LF_Source_Port_Less_6600_Attack and LF_IAT_Min_Less_Mean_Attack, we see a stark difference between LF F1 and classifier F1. The LF F1s for both combinations are zeros. We propose that this is caused by the true positives that are so low or even zero, which could result in zero precision or zero recall, which in turn would result in a zero F1 score. Regarding the classifier F1, both the precision and recall may not be zero, which leads to a non-zero F1 score. The reason for a zero true positive could be because the labeling functions are one-sided (only detect one class), or could be because the coverage of the labeling functions almost completely overlaps, which render one of the labeling function redundant.

In addition to the above experiments, we also present a case where combination of two uncorrelated labeling functions do not affect the combined F1 score that much. On the last row, we see that the classifier F1 score of the combination between labeling functions with two uncorrelated features (LF_Source_Port_Less6600_Attack and LF_IAT_Min_Less_Mean_Attack) results in a slight decrease of the classifier F1 score of LF_Source_Port_Less6600_Attack (from 0.877 to 0.863, and also a decrease of the F1 score of LF_IAT_Min_Less_Mean_Attack (from 0.892 to 0.863).

Finally, the results of this experiment did not present a clear conclusion on the effects of feature correlation in the classifier F1 score of the combined labeling functions. Thus, we refrain from making a definite conclusion and proceed to the experiments on the next dataset.

Dataset-specifc Interpretability Tree. The visualization trees generated from labeling function combinations of row 3 and row 5 in Table 4 are shown in Figures 11 and 12. In the tree on Figure 11, an example of flow data point with a source port of 5555 would be considered an attack, since it is less than or equal to 6600. A data point with source port 8888 would not be considered an attack immediately, but would go to the right child node where its average packet size would be checked if it is less than 100. Supposing that the average packet size is 400 bits, the data point’s class would be inconclusive. However, if the average packet size is 80 bits, the data point would be considered benign.

Suppose that we have a new data point with a source port of 8888 and average packet size of 80. According to the tree on Figure 11, the data point will go to the root’s right child, where its average packet size will be checked if it is less than 100. Since the new data’s average packet size is 80, it will be considered inconclusive. Note that the hierarchy of the nodes are determined by feature importance, which in this case the source port is of higher importance than the average packet size. However, using the same data point, suppose that average packet size is more important than source port. When the average packet size is at the root, it will check if the average packet size of the new data point is less than 100. Since the new data points’ average packet size is 80, the data point will go to the left child of the root, at which the data point is considered benign.

This example shows that feature importance plays a major role in the decision-making process. The tree offers a degree of explanation by figuring out the tree structure and portraying the nodes where the features and their thresholds assess a data point and the paths to the results.

As in the case before, we also investigated the label consistency between the intrepretability tree and the manual labeling function generator. In Table. 4, the F1 score of the combined LF_Avg_Pkt_Less_100_Benign and LF_Source_Port_Less_6600_Attack (LF F1) is 0.893. The F1

score of the interpretability tree is 0.783, which is actually slightly lower than the F1 score of the manual labeling functions generator. This is not surprising since the interpretability tree is an approximation, which usually performs worse.

The tree's F1 score, however, is higher when we investigated Figure 12. In Table 4, the F1 score of the manual labeling function generator (LF F1) of the combined LF_Avg_Pkt_Less_100_Benign and LF_Flow_IAT_Min_Less_Mean is 0. As discussed above, this could be because the true positive is extremely low or even zero, which could lead to a zero precision or recall. The F1 score of the interpretability tree, however, is 0.749. This is a showcase where the interpretability tree generates a much more optimistic F1 score, and thus require further work to improve it.

F1 and prediction comparison between manual labeling function generators, trained classifiers, and dataset-specific interpretability tree. We present the F1 scores of the three classifiers: a) the manual labeling function generator in the generative model, b) the trained classifier in the discriminative model, and c) the interpretability tree. We consider these three as classifiers since they produce labels for negative or positive classes given a data point. Evaluation of the F1 scores are done on the test set. The F1 scores comparison is shown in Table 5. Each row shows the F1 scores of the three classifiers for the labeling function combination.

In this Table, we see that the combination of LF_Source_Port_Less_6600_Attack and LF_Source_Port_Greater9000_Benign generates 0.899 LF F1 score, 0.897 trained classifier F1 score, and 0.894 tree F1 score. In this case, LF produces the highest label quality, followed by trained classifier, and then tree. Since the F1 scores of the three classifiers are close to each other, we wanted to investigate how different their predictions are. We conducted the prediction comparison by taking 20 percent of the test data, pairing up the classifiers (e.g., LF and Cls, LF and tree, Cls and Tree), treating one of them as ground truth, and measure the prediction difference between the classifiers in each pair. Thus, we define prediction difference as the fraction of predictions that are different from the methods in each pair.

Note that since we compared the prediction difference only when the F1 scores of the classifiers are close to each other, we did not compare the predictions of labeling function combinations on the second and third row. We notice that the LF F1 of the labeling function combinations on the second and third row are zero, which indicate that their true positives are zero. This implies that the manual labeling function generators performed poorly in predicting the positive class. However, on the second row, the trained classifier and the tree performed moderately well, and on the third row, the trained classifier performed rather poorly while the tree performed moderately well. Since there is a large gap between the F1 scores of the classifiers on the second and third rows, we did not investigate the prediction difference between the three classifiers.

The prediction comparison results of LF_Source_Port_Less_6600_Attack and LF_Source_Port_Greater9000_Benign are shown in Table 6. Each row in the second column corresponds to the the rows in the third, fourth, and fifth columns. On the first row we see that the prediction difference between LF and LF is 0.0, between LF and Cls is 0.0036, and between LF and Tree is 0.00032. On the second row, we see that the prediction difference between Cls and LF is 0.0036, between Cls and Cls is 0.0, and between Cls and Tre is 0.0036. On the third row, the prediction difference between Tree and LF is 0.00032, between Tree and Cls is 0.0036, and between Tree and Tree is 0.0.

Considering both Tables 5 and 6 for the combination of LF_Source_Port_Less_6600_Attack and LF_Source_Port_Greater9000_Benign, we see that there is only a slight prediction difference between the three classifiers, reflecting the F1 difference between them.

We also notice a symmetric relationship between the comparison pairs. The prediction difference between LF and Cls is the same as that of Cls and LF, the prediction difference between LF and Tree is the same as that of Tree and LF, and the prediction difference between Cls and Tree is the same as that of Tree and Cls.

Summary. The goal of these two experiments is to demonstrate how our framework can support privacy-preserving collaboration, analyze factors that can affect label quality, and provide a level of interpretability of the results. We used the CIC DDoS NTP dataset, which is a synthetic dataset created for DDoS NTP detection. We combined multiple labeling functions to show how multiple research groups can contribute their labeling functions in the community effort to improve label quality while avoiding privacy leaks. We combined labeling functions that are based on various features to examine the effects of the combination on label quality. In the first experiment, we found that combining two labeling functions that are based on one feature can improve label quality if the combination extends the coverage of the labeling functions. In the second experiment we examined the effect of feature correlation on label quality, and the results show that labeling function combination based on highly correlated features can improve label quality, while the combination of the labeling functions that are based on weakly correlated features can decrease label quality. We also observed a case that is similar to the latter combination, where labeling function combination benefited one labeling function but not the other. However, the decrease and increase

First LF	Second LF	LF F1	Cls. F1	Tree F1	Figure Number
LF_Source_Port_Less_6600_Attack	LF_Source_Port_Greater9000_Benign	0.899	0.897	0.894	10
LF_Source_Port_Less_6600	LF_Avg_Pkt_Less_100	0	0.893	0.783	11
LF_Avg_Pkt_Less_100	LF_IAT_Min_Less_Mean_Attack	0	0.249	0.749	12

Table 5: LF F1, trained classifier F1, and tree F1 of the labeling functions and the trees in Figure 10, 11, and 12 from the DDoS NTP dataset and evaluated on the test set. LF F1 refers to the F1 score of the manual labeling function generator, Cls F1 refers to the F1 score of the trained classifier, and Tree F1 refers to the F1 score of the interpretability tree.

Labeling function combination	Prediction Difference	LF	Cls	Tree
DDoS LF_Source_Port_Less_6600_Attack	LF	0.0	0.0036	0.00032
	Cls	0.0036	0.0	0.0036
DDoS LF_Source_Port_Greater_9000_Benign	Tree	0.00032	0.0036	0.0

Table 6: Prediction difference between LF, trained classifier, and tree on the DDoS NTP dataset. Each row in the second column corresponds to the third, fourth, and the fifth columns (e.g., the percent of prediction difference of LF with LF, Cls, and Tree are 0.0, 0.0036, and 0.00032, respectively.)

of the performance are small. With these results, we cannot make a conclusion with confidence on the effect of feature correlation on combined labeling function performance. Following that, we presented the summary of the LF F1, Cls F1, and the Tree F1 in Table 5. We see that these tree F1 results rivals those of LF and Cls. Finally, we compared the predictions of the labeling function combinations if their F1 scores are close to each other. The results of our comparison shows a symmetric relationship between the comparison pairs.

4.4 Case Study 4: Worf

Overview. The goal the experiments on this dataset is similar to that of the experiments on CIC-DDoS NTP dataset. We aim to show that our framework provides platform for user-defined labeling functions while maintaining privacy, to show that our framework is able to readily handle NetFlow data, and to provide support in writing labeling function by analyzing the effects of various labeling function combinations on label quality. The difference between the CIC-DDoS NTP dataset with the Worf dataset is that the CIC-DDoS NTP dataset consists of synthetic data, while the Worf dataset consists of real data that capture actual DDoS NTP attacks.

Worf dataset. Our Worf dataset comprises traffic traces from the Front Range GigaPop (FRGP) network [4], a medium-size academic and research network exchange in the United States that connects federal research labs, nonprofit, universities, and other educational institutions. The dataset consists of NetFlow data capturing the events occurring on May 24, 2020 from 06:46am - 06:51am UTC, on September 13, 2020 from 10:50pm - 11:01pm UTC, and on November 13, 2020 from 4:08pm - 04:13pm UTC, during which the network system recognized NTP DDoS attacks. We collected traffic traces from these three time periods for the attack profile and also traces on November 8, 2020 from 02:00pm - 02:05pm for the benign network profile. We chose such time period for our benign profile because we found many other DDoS attack types occurring around the same time as our selected NTP DDoS attack periods. From these flow data, we sampled 300,000 data points with an equal number of positive and negative classes.

The goals and experiment procedures are the same as in the DDoS 2019 case (§4.3). Our first goal is to demonstrate our framework's support for research collaboration that preserves data privacy, and our second goal is to investigate the effects of LF combinations based on the feature correlation, in order to promote a more fruitful collaboration. We divided our experiments into two parts. The first experiment used only one feature to write LFs, and then combined these LFs and assessed the results. The second experiment used three features for LFs, but combining only two of them at a time. The first two features are correlated, while the third feature has little or no correlation with the other two. The experiment is to investigate the effect of correlation in combining LFs. We structured the DDoS 2019 and Worf experiments to be similar so we can assess how our framework performs on two different NTP DDoS datasets: a synthetic dataset (DDoS 2019), and an "in-the-wild" dataset (Worf).

Data preparation. From the raw Worf data, we took only the following fields: "First Seen", "Last Seen", "Duration", "Source Port", "Bits Per Second", "Packets Per Second", "Bytes Per Package", "Packets", and "Bytes". We removed NaNs and duplicates, and sorted the data based on the "First Seen" field.

Selected Feature	Description	Corr. with Packets	Corr. with Source Port	Corr. with Bytes
Packets	Number of packets	1.0	0.060	0.887
Source Port	Source port	0.060	1.0	-0.067
Bytes	Number of bytes	0.887	-0.067	1.0

Table 7: Selected Worf features, description, and correlation.

Then we matched the “First Seen" field of our data to the “First Seen" field recorded on the alert report to create the ground truth labels. From this point, the procedure roughly followed the same path as the DDoS 2019 experiment.

Feature selection. We initially planned to use the same features as the ones we used in the DDoS 2019 experiments, but since we found many missing values on the “Last Seen" field (which was recorded using the same timestamp as the “First Seen" field in the case of missing values), we decided to exclude features that are calculated using “Last Seen", such as “Duration", “Bits Per Second", and “Packets Per Second". Following the similar procedures that we used in the DDoS 2019 experiments, we needed three features. To select important features, we computed the correlation coefficients, and, since we also had the ground truth labels, we computed the feature importance score, which listed “Packets", “Source Port", and “Bits Per Second" as the top three most important features. From these three features, we selected “Packets" and “Source Port" but not “Bits Per Second" since its values are calculated using “Last Seen", which are often unreliable due to the large amount of missing values. For the third feature, we instead selected one that is highly correlated with “Packets," since “Source Port" appeared to have low correlation with other features on this dataset. The last feature that we selected was “Bytes." Table 7 shows the selected features and their correlations.

Label generation. Although we wanted to structure our Worf experiments to be as similar as possible to the DDoS 2019 experiments, due to the different statistical characteristics of the two datasets, we could not use the same features or apply the same threshold values on the Worf experiments. Also, in real world collaboration scenarios, different research groups should *not* include specific numerical threshold values in their labeling functions, since this could be considered a privacy leak. Thus, we wrote labeling functions based on the statistical characteristics of the three features. Table 8 shows the statistical features that we considered in writing labeling functions. Besides these values, we also considered port 123 since it is known as the designated port for NTP. We also measured the labeling functions' F1 score, which in the result we will refer to as LF F1.

End-classifier training and evaluation. We used LSTM as our end-classifier in both experiments 1 and 2. We trained the LSTM models using the generated training labels. We fine-tuned our LSTM by trying out various values: learning rate (0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01, 0.03, 0.05), number of epochs (20, 30, 50), number of LSTM cells (16, 32, 64, 128, 256), and batch size (64, 128, 256). We used L2 regularization (between 0.2 and 0.6) and dropout (between 0.2 and 0.6) to guard against overfitting. Note that different labeling function combinations have different hyperparameter settings.

Worf Results. The results of the two experiments are shown in Tables 9 and 10.

4.4.1 ***Experiment 1: Combination of labeling functions that are based on one feature.*** In this experiment, our goal is to examine the effects of combining labeling functions that are based on one feature but using different threshold values. Out of the three selected features listed on Table 7, we used “Bytes" since the results showed an interesting case, which we will discuss in the following.

The statistical values for the mean and the 75th percentile on “Bytes" can be found in Table 8. The mean is 537,300 and the 75th percentile is 380. Looking at the results in Table 9, the classifier F1 scores for labels that classify data points with the number of bytes less than 380 (its 75th percentile) is 0.667, the classifier F1 score for labels that classify data points with the number of bytes less than 537,300 (its mean) is the same (0.667), and the classifier F1 score of the combination is also 0.667, which neither increase or decrease the individual scores.

The result appears to disagree with the result of experiment 1 in the DDoS NTP dataset where the resulting classifier F1 score of one-feature labeling function combination is 0.897, which improves upon the higher individual score (0.849). However, upon further investigation, we see that in the case of DDoS NTP experiment 1, the combination of the thresholds (less than 6600 and greater than 9000) actually *expands* the coverage, as one threshold covers values less than 6600, while another covers values greater than 9000, albeit there are values that are not covered.

This was not the case for the Worf dataset. The fact that the F1 scores are the same for different thresholds suggests that one of the thresholds, which in this case is 537,300, does not add benefits to 380. A clearer picture can be examined by looking at the tree in Figure 13.

Statistical Feature	Source Port	Packets	Bytes
count	300000.000000	3.000000e+05	3.000000e+05
mean	20915.958463	6.26448e+02	5.373005e+05
std	24088.129213	1.695706e+04	1.909392e+07
min	0.000000	1.000000e+00	2.500000e+01
25%	123.000000	1.000000e+00	7.600000e+01
50%	3074.000000	1.000000e+00	3.500000e+02
75%	47103.250000	1.000000e+02	3.800000e+02
max	65535.000000	8.495104e+06	9.140732e+09

Table 8: Statistical characteristics of the selected features from the Worf dataset

LF	LF F1	Classifier F1
LF_Bytes_75percentile_Benign	0.523	0.667
LF_Bytes_Mean_Benign	0.667	0.667
Combined	0.667	0.667

Table 9: The F1 scores of the one-featured labeling function (Bytes) combination of the Worf data.

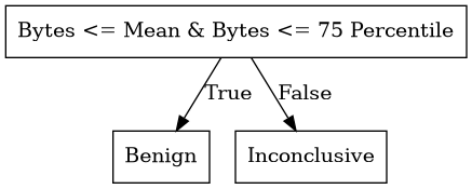


Figure 13: Decision tree generated from Table 9; an LF combination of different threshold values on "Bytes". The F1 score of the tree is 0.712, while the F1 score of the labeling function (LF F1 in Table 9) is 0.667.

Looking at the LF F1 scores on Table 9, we see that LF_Bytes_Half_75percentile_Benign generates a 0.523 LF F1, LF_Bytes_Mean_Benign generates a 0.667 LF F1, and the combination of the two labeling functions generates a 0.667 LF F1. Comparing these F1 scores with those of the classifer's, we see only a slight difference between LF F1 and classifier F1 of LF_Bytes_Half_75percentile_Benign, while a perfect match for LF_Bytes_Mean_Benign and the combined labeling functions (0.667). In this case, contrary to the cases we saw in DDoS NTP, the LF F1 generates the same F1 score as the classifier F1 for LF_Bytes_Half_75percentile_Benign. This could indicate that the labeling functions generalize quite well.

Dataset-specific Interpretability Tree. The tree in Figure 13 visualizes the threshold values and the conclusion that FLAMENCO uses to label the data. Both labeling functions aim to detect the benign case by categorizing the number of bytes less than a certain threshold to be benign. Both labeling functions aim to detect the benign case by categorizing the number of bytes less than a certain threshold to be benign. At the root, we see a condition where data with the number of bytes less than or equal to its mean *and* its 75th percentile are considered benign and values beyond this range are considered inconclusive. Since we are considering an "AND" condition, the combination of the two thresholds applies only to that of the lesser value, which in this case is 380 (the 75th percentile). Data with the number of bytes greater than 380 are considered inconclusive. Moreover, When we consider the range of the number of bytes in this dataset, which is from 25 to around 9 billion (Table 8), 380 covers a very small slice of values, since the rest of the values are considered inconclusive. Finding other threshold values that can capture the ranges in this inconclusive region requires further investigation. We realize that choosing a value between 380 and 9 billion resembles the attempt to find a needle in a haystack. We need a more methodical approach to find the values, which we leave for future work.

In addition to the discussion above, we also calculated the F1 score of the interpretability tree in Figure 13 and compared it to the F1 score of the manual labeling function generator shown in Table 9. Our approach in calculating the F1 score of the tree is the same as the one we used in the DDoS NTP experiment. We calculated the Tree F1 score by taking each data point from the test set and traversing the tree until the data point reaches a terminal node, which determines the label for that data point. We compared this label with the ground-truth label from the test set, then counted the number of true positives, true negatives, false positives, and false negatives. Using these numbers, we calculated precision and recall, and finally the F1 score. The combined labeling functions (LF F1) is 0.667. Our calculated F1 score of the tree is 0.712. We acknowledge that this is another example where the interpretability tree generates a more optimistic F1 score.

4.4.2 ***Experiment 2: Combination of labeling functions that are based two features.*** In this second experiment on the Worf dataset, our goal is similar to the first

First LF	Second LF	LF F1	Trained Classier F1	Correlation
LF_Packets_75percentile_Benign	-	0.0738	0.609	-
LF_Packets_Mean_Benign	-	0.505	0.667	-
LF_Bytes_Half_75percentile_Benign	-	0.516	0.667	-
LF_Bytes_Mean_Benign	-	0.667	0.667	-
LF_Source_Port_123_Attack	-	0	0.577	-
LF_Source_Port_75percentile_Benign	-	0.313	0.727	-
LF_Bytes_Half_75percentile_Benign	LF_Packets_75percentile_Benign	0.516	0.667	0.887
LF_Source_Port_123_Attack	LF_Bytes_Mean_Benign	0.667	0.667	-0.067
LF_Packets_Mean_Benign	LF_Source_Port_75percentile_Benign	0.593	0.667	0.060

Table 10: The F1 scores of the two-featured labeling function combinations of the Worf dataset. We colored the labeling function names to make it easier to track the combination of the labeling functions. LF_Packets_75percentile_Benign is dark yellow, LF_Packets_Mean_Benign is light yellow, LF_Bytes_Half_75percentile_Benign is grass green, LF_Bytes_Mean_Benign is mint green, LF_Source_Port_123_Attack is dark pink, and LF_Source_Port_75percentile_Benign is light pink.

experiment, that is, to investigate the effects of labeling function combinations on label quality. However, since we are also interested in examining the effect of correlation on the label quality, we combine two labeling functions that are based on two different features *at a time*. To do this, we selected three features with different levels of correlation. These features are the same as the ones we use in experiment 1. The selected features and their correlation are shown in Table 7. On the table, we see that “Packets" and “Bytes" are highly correlated (0.887), “Packet" and “Source Port" are weakly correlated (0.060), and “Source Port" and “Bytes" are also weakly correlated (-0.067). We then write several labeling functions based on these three features.

The individual and combined results can be seen on Table 10. We colored the table cells according to the features and labeling functions for easier reading. There are two labeling functions on each feature, making up a total of six labeling functions. The labeling functions that are based on “Packets" are the ones with shades of yellow, the labeling functions on “Bytes" are those with shades of green, and the labeling functions on “Source Port" are those with shades of pink.

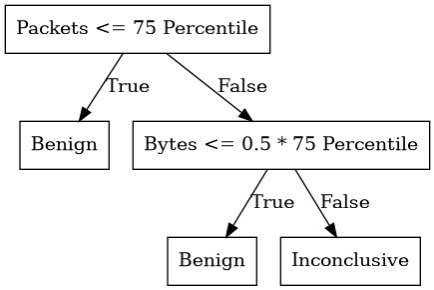
On Table 10, the first six rows show the individual F1 scores of six labeling functions, while the last three rows show the F1 scores of the combined labeling functions. Almost all of the individual labeling functions have an F1 score around 0.6 and 0.7. Note that in this experiment we define positive class as benign traffic, while negative class as malicious traffic. This is because we are looking from a network operator’s viewpoint where positive means benign traffic and negative means malicious traffic that needs to be blocked.

In contrast, the labeling function that classifies traffic with source port greater than or equal to its 75th percentile (47,103 as shown in Table 8) as benign generates a trained classifier F1 score of 0.727. This labeling function covers quite a high range of values, from 47,103 to 65,535, and thus coverage could be an important factor for the labeling function performance.

Besides coverage, we are also interested in investigating the impact of feature correlation on labeling function performance. On Table 10, the F1 score of the combined labeling functions are *all* 0.667, regardless of the correlation between the features from which they are based. Comparing this result and the result from the second DDoS NTP experiment, we lean towards a conclusion that feature correlation may have little impact on the F1 score of the combined labeling functions.

Dataset-specific Interpretability Tree. The trees that are generated from the three labeling function combinations are shown in Figures 14, 15, and 16.

In Figure 14, the tree portrays the combination of labeling functions on “Packets" and “Bytes." Since “Packets" is listed as the most important feature according to the feature importance calculation, we placed “Packets" at the root. Here, the labeling function considers data with the number of packets less than or equal to its 75th percentile (100) to be benign and the values beyond this range to be inconclusive. We propose that since the first labeling function refrains from labeling the data with the number of packets greater than 100, we could place the second labeling function as a further attempt to arrive at a decision. In this case, the second labeling function considers the data with a number of bytes less than or equal to half of its 75th percentile to be benign, and leave the data beyond this value to be inconclusive. We acknowledge that we can keep adding labeling functions to combine until there is no inconclusive result. However, in this experiment, we wanted to keep it simple by combining only two labeling functions at a time. Considering the tree and the F1 scores in Table 10, we see that the individual



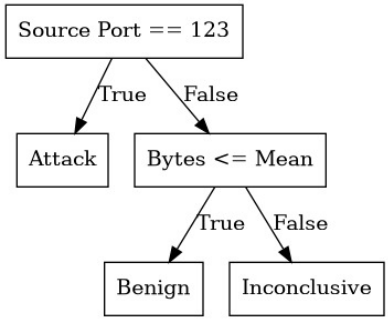

```python
def LF_Packets_75percentile_Benign(x):

    val = x.number2.get_attrib_tokens()
    if float(val[0]) <= df_stats_features.iloc[idx['75%']]['Packets']:
        return 1
    else:
        return 0
```

```python
def LF_Bytes_Half_75percentile_Benign(x):

    val = x.number3.get_attrib_tokens()
    if float(val[0]) <= (0.5 * df_stats_features.iloc[dict_stat_features_index['75%']]['Bytes']):
        return 1
    else:
        return 0
```

Figure 14: The decision tree generated from labeling functions that are based on “Bytes" and “Packets" from the Worf dataset. The F1 score of this tree is 0.708, while the F1 score of the labeling function (LF F1) is 0.512 (Table 10).



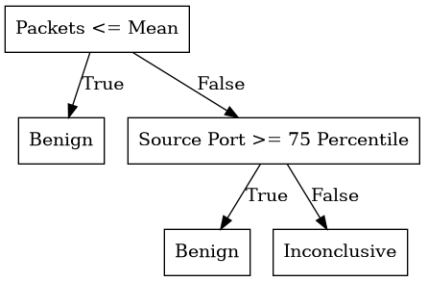

```python
def LF_Source_Port_123_Attack(x):

    val = x.number1.get_attrib_tokens()
    if float(val[0]) == 123:
        return -1
    else:
        return 0
```

```python
def LF_Bytes_Mean_Benign(x):

    val = x.number3.get_attrib_tokens()
    if float(val[0]) <= df_stats_features.iloc[dict_stat_features_index['mean']]['Bytes']:
        return 1
    else:
        return 0
```

Figure 15: The decision tree generated from labeling functions that are based on “Source Port" and “Bytes" from the Worf dataset. The F1 score of the tree is 0.592, while the LF F1 of the combined labeling functions is 0.667 (Table 10).




```python
def LF_Packets_Mean_Benign(x):

    val = x.number2.get_attrib_tokens()
    if float(val[0]) <= df_stats_features.iloc[dict_stat_features_index['mean']]['Packets']:
        return 1
    else:
        return 0
```

```python
def LF_Source_Port_75percentile_Benign(x):
    val = x.number1.get_attrib_tokens()
    if float(val[0]) >= df_stats_features.iloc[dict_stat_features_index['75%']]['Source Port']:
        return 1
    else:
        return 0
```

Figure 16: The decision tree generated from labeling functions that are based on “Source Port" and “Packet Size" from the Worf dataset. The F1 score of the tree is 0.688, while the LF F1 of the combined labeling functions is 0.593 (Table 10).

First LF	Second LF	LF F1	Cls. F1	Tree F1	Figure Number
LF_Bytes_75percentile_Benign	LF_Bytes_Mean_Benign	0.667	0.667	0.712	13
LF_Bytes_Half_75percentile_Benign	LF_Packets_75percentile_Benign	0.516	0.667	0.707	14
LF_Source_Port_123_Attack	LF_Bytes_Mean_Benign	0.667	0.667	0.592	15
LF_Packets_Mean_Benign	LF_Source_Port_75percentile_Benign	0.593	0.667	0.688	16

Table 11: LF F1, trained classifier F1, and tree F1 of the labeling functions and the trees in Figure 13, 14, 15, and 16, all evaluated on the test set.

Labeling function combination	Prediction Difference	LF	Cls	Tree
Worf LF_Bytes_75Percentile_Benign	LF	0.0	0.0	0.2855
	Cls	0.0	0.0	0.2855
Worf LF_Bytes_Mean_Benign	Tree	0.2855	0.2855	0.0
Worf LF_Source_Port_123_Attack	LF	0.0	0.0	0.4533
	Cls	0.0	0.0	0.4533
Worf LF_Bytes_Greater_Mean_Benign	Tree	0.4533	0.4533	0.0
Worf LF_Packet_Mean_Benign	LF	0.0	0.2475	0.316
	Cls	0.2475	0.0	0.1164
Worf LF_Source_75Percentile_Benign	Tree	0.316	0.1164	0.0

Table 12: Prediction difference between LF, trained classifier, and tree on the Worf dataset. The first column shows the labeling function combinations, the second column lists the three classifiers for each labeling function combination, the third, fourth, and the fifth columns show the prediction differences that correspond with the classifiers in column 2.

labeling functions have F1 scores of 0.665 and 0.667, and the combined labeling functions generate an F1 score of 0.667, which does not improve upon the individual scores. One possible reason for this is that although the labeling functions are on different features and covers two different ranges, their coverage might be redundant. This result suggests that redundant labeling function coverage might not improve the individual labeling function performance.

The combination of the labeling functions on “Source Port" and “Bytes", however, tells a different story. The F1 score of LF_Source_Port_123_Attack is 0.577, and when combined with LF_Bytes_Mean_Benign, which has a score of 0.667, the F1 score of the combination is 0.667, which is the same as that of LF_Bytes_Mean_Benign. Looking at the tree on Figure 15, we see that the data with source port other than 123 are considered inconclusive by the first labeling function. As in the case of the previous combination, the second labeling function is another attempt to classify this inconclusive region. Here, the data with source port other than 123 are checked if their number of bytes is less than or equal to its mean. If the condition is true, these data are considered benign, and if false, the data are still considered inconclusive. We see that the F1 score of the combined labeling functions is the same as that of LF_Bytes_Mean_Benign. One possible reason for this result is that the second labeling function does not add value to the first labeling function, which could be because the second labeling function is redundant to the first one. On the other hand, the first labeling function brings value to the second labeling function.

Next, we are looking at the combination of LF_Packets_Mean_Benign and LF_Source_Port_75percentile_Benign. The F1 score of the first labeling function is 0.667, and the score of the second labeling function is 0.727. The combination, however, yields 0.667, which is the same as the first labeling function, but lower than the second. The tree in Figure 16 shows that data with the number of packets less than or equal to their mean is considered benign, and beyond that, the labeling function considers them inconclusive. In this inconclusive region, the second labeling function checks their source port and considers the data with a source port greater than or equal to their 75th percentile to be benign. Beyond these two conditions, the data are considered inconclusive. In this result, the higher performing labeling function (LF_Source_Port_75percentile_Benign) adds no value to the lower performing labeling function (LF_Packets_Mean_Benign) as the latter’s F1 score is the same. The combination of these two labeling functions seem to bring no benefit to both labeling functions. Even worse, the labeling function combination actually performs worse than LF_Source_Port_75percentile_Benign.

Moreover, the fact that the combined labeling function yields the same F1 score as LF_Packets_Mean_Benign while decreasing the score of LF_Source_Port_75percentile_Benign seems to suggest that LF_Source_Port_75percentile_Benign brings redundant information to LF_Packets_Mean_Benign, and maybe even conflicts.

For this part of the Worf experiment, we see that LF F1 in Table 10 (grass green and dark yellow) is 0.516. The F1 score of the tree in Figure 14 is 0.707, which, again, is more optimistic than LF F1.

Surprisingly, however, the F1 score of the tree in Figure 15 is 0.592, while LF F1 of the combination of LF_Source_Port_123_Attack and LF_Bytes_Mean_Benign is 0.667 (refer to Table 10). This result disagrees with the ones we have before, where the tree consistently generates a more optimistic F1 score.

Finally, the F1 score of the tree in Figure 16 is 0.688, while the LF F1 of the combination of LF_Packets_Mean_Benign

and LF_Source_Port_75percentile_Benign is 0.593, as shown in Table 10. Here we see another example where the tree generates a higher F1 score compared to the manual labeling function generator's, which is actually consistent with almost all cases we investigated in this paper.

yConsidering the F1 results of the three classifiers, and especially when we are comparing only LF and Cls, we see that the trained classifier does not improve LF F1 that much, since some of its F1 scores are only slightly higher than that of LF.

F1 and prediction comparison between manual labeling function generators, trained classifiers, and dataset-specific interpretability tree. Similar to what we did in the DDoS 2019 experiments, we compared the F1 scores of the three classifiers to assess the similarity and the difference between them. The F1 comparison is shown in Table 11. In this table, we see that LF F1s are sometimes the same as Cls F1s (rows 1 and 3), while Tree F1s are always different than those of LF and Cls. Considering this table, we only investigated the labeling function combinations of LF_Bytes_75percentile_Benign and LF_Bytes_Mean_Benign, LF_Source_Port_123_Attack and LF_Bytes_Mean_Benign, and LF_Packets_Mean_Benign and LF_Source_Port_75percentile_Benign since the F1 scores of the classifiers for these combinations are close to each other. Table 12 shows the prediction difference between the three classifier pairs.Note that we define prediction difference as the fraction of predictions that are different between the two methods in each pair.

On Table 11, we see the same F1 score for LF and Cls of the combination of LF_Bytes_75percentile_Benign and LF_Bytes_Mean_Benign (row 1), and the combination of LF_Source_Port_123_Attack and LF_Bytes_Mean_Benign, and LF_Packets_Mean_Benign (row 3). The Tree F1, however, are slightly different. The corresponding prediction difference for LF_Bytes_75percentile_Benign and LF_Bytes_Mean_Benign combination is shown in Table 12 row 1.

Reading LF along the row and matching LF along the column gave us 0.0 prediction difference, which is to be expected since both are the LF classifier. Continuing to read the table this way, we see that LF on row 1 and the Cls column have 0.0 prediction difference, which is consistent with the F1 scores of both classifiers in Table 11. Reading LF along the row and Tree along the column gave us 0.2855 prediction difference. This result is also consistent with the F1 scores of LF and Tree in Table 11 (row 1), where there is a slight difference between both F1 scores.

Looking at the the combination of LF_Source_Port_123_Attack and LF_Bytes_Mean_Benign in Table 11 (row 3), the F1 scores of LF and Cls are the same (0.667), while that of Tree is lower (0.592). As we did in the previous labeling function combination, we investigated the discrepancy of the classifiers' F1 by looking into their predictions. In Table 12, we see that LF and Cls made the same prediction, since their prediction difference is 0.0. We also see that the prediction difference between LF and Tree is 0.4533, and the prediction difference between Cls and Tree is also 0.4533. This result is expected since LF and Cls have the same F1 score and thus their prediction difference with Tree would also be also the same.

Next, the third labeling function combination is LF_Packets_Mean_Benign and LF_Source_Port_75percentile_Benign, whose F1 scores are all different for the classifiers (Table 11 row 4).

Looking deeper into their predictions in Table 12, we see that LF and Cls have a 0.2475 prediction difference, while LF and Tree have a 0.316 prediction difference. Next, we see that the prediction difference between Cls and LF is 0.2475, and the prediction difference between Cls and Tree is 0.1164. From these results, we can see a symmetric relationship between LF and Cls, since the prediction difference between LF and Cls is the same as that of Cls and LF, which is 0.2475. Following this, we see the prediction between Tree and LF and also between Tree and Cls. The prediction difference between Tree and LF is 0.316, while that of Tree and Cls is 0.1164. Here we see another symmetric relationship since the prediction difference between LF and Tree is also 0.316, and the predictions difference between Cls and Tree is 0.1164. All these results show a symmetric relationship.

Summary. The goal of these two experiments is to show how our framework can support privacy-preserving collaboration, analyze factors that can affect label quality, and provide a level of interpretability on the results. We used the Worf dataset, which is an "in-the-wild" DDoS NTP dataset. We supposed that different research groups contribute their labeling functions to be combined, and we examined the effects of feature correlation on label quality. In the first experiment, we combined labeling functions that are based on one feature, and showed that the combination of labeling functions that have the same coverage has no effect on label quality. In the second experiment, we combined labeling functions that are based on two features that have different levels of correlation. We found that feature correlation has little impact, if any, on label quality, while labeling function coverage plays a more important role in deciding the label quality. Regarding the interpretability tree's F1 scores, we see that in in many cases, Tree produced F1 scores that are higher than the other two classifiers. This is strange since usually a less precise approximation (like found in Tree) will perform worse than a more precise method. We leave the investigation of this issue for future work. Finally, in our attempt to conduct consistency checks, we also looked into

the prediction differences between the labeling function combinations, which are shown in Table 12. In all cases, we see that the prediction difference between classifier pairs are symmetric. Moreover, considering the F1 results on Table 4, we see that while the tree gives a more optimistic F1 scores than the other two methods, since the tree is only an approximation, we are looking only at the LF F1 and Cls F1. We see that in a case that LF is not zero, its F1 is as good as the trained classifier.

5 SUMMARY AND FUTURE WORK

In this work, we propose a framework to create training labels for network data with the ability to (1) handle the diversity of networking data, (2) create labels programmatically to scale domain expertise, (3) support privacy-preserving collaborations in a community effort to improve label quality, and (4) provide a level of interpretability on the results. We show that our framework can handle latency measurement data (traceroute and ping) that have multimodal distribution and are used for noise detection, and synthetic and "in-the-wild" flow data that are used for DDoS NTP detection. Our framework allows users to create labels programmatically to make it easier to scale domain expertise to more datasets. We show how our framework can support privacy-preserving collaboration by demonstrating how labeling functions are used and what combination of labeling functions can affect label quality. Lastly, we designed our framework to map the labeling functions to a tree to provide a level of interpretability through visualization of the decision path that the framework use to classify the data.

We realize that there are network data types beyond latency measurements and flow data, and to improve our framework's versatility, we plan to develop components that can handle data from a different network layer (*e.g.,* layer 2).

Moreover, we found that our design of labeling function mapping to the tree is quite restrictive, as it only allows users to write labeling functions that strictly follow certain patterns, such as limiting one classification rule for each labeling function and using "inconclusive" decision in each labeling function. We seek to design a more flexible method to allow more variations of labeling functions. With a more flexible mapping between the labeling functions and the trees, we expect that the trees would generate closer F1 scores to the F1 scores of the manual labeling function generators (LF F1). In addition to this restrictive nature, we also realize that forming heuristics for user-defined labeling function can be difficult, and thus we seek to find a more methodical way to help users in writing labeling functions.

Concretely, our future works include the following:

- Develop components to handle data from a different network layer.
- Develop a more flexible mapping between labeling functions and the trees.
- Modify tree structures to render a closer F1 score to the LF F1 score.
- Develop a component that can provide a more methodical way for users to form heuristics.

REFERENCES

[1] Explainable machine learning challenge.

[2] CAIDA: The IPv4 Routed /24 Topology Dataset, Nov. 2019.

[3] Canadian Institute of Cybersecurity, May 2020.

[4] The Front Range Gigapop: Connecting colorado and wyoming with hundreds of gigabits, September 2020.

[5] Ripe Atlas, May 2020.

[6] Ateniese, G., Mancini, L. V., Spognardi, A., Villani, A., Vitali, D., and Felici, G. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks 10*, 3 (2015), 137–150.

[7] Bach, S. H., Rodriguez, D., Liu, Y., Luo, C., Shao, H., Xia, C., Sen, S., Ratner, A., Hancock, B., Alborzi, H., et al. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *International Conference on Management of Data* (2019), ACM, pp. 362–375.

[8] Bunescu, R., and Mooney, R. Learning to extract relations from the web using minimal supervision. In *Association of Computational Linguistics* (2007), pp. 576–583.

[9] Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing 307* (2018), 72–77.

[10] Cloud, G. Explainable ai: Tools and frameworks to understand and interpret your machine learning models, 2021.

[11] Craven, M., and Kumlien, J. Constructing biological knowledge bases by extracting information from text sources. In *International Conference on Intelligent Systems for Molecular Biology*, AAAI.

[12] Czyz, J., Kallitsis, M., Gharaibeh, M., Papadopoulos, C., Bailey, M., and Karir, M. Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks. In *Internet Measurement Conference* (2014), pp. 435–448.

[13] Das, N., Chaba, S., Wu, R., Gandhi, S., Chau, D. H., and Chu, X. Goggles: Automatic image labeling with affinity coding. In *International Conference on Management of Data* (2020), ACM, pp. 1717–1732.

[14] Flores, A. W., Bechtel, K., and Lowenkamp, C. T. False positives, false negatives, and false analyses: A rejoinder to machine bias: There's software used across the country to predict future criminals. and it's biased against blacks. *Federal Probation 80* (2016), 38.

[15] Fredrikson, M., Jha, S., and Ristenpart, T. Model inversion attacks that exploit confidence information and basic countermeasures. In *Conference on Computer and Communications Security* (2015), ACM, pp. 1322–1333.

[16] Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., and Ristenpart, T. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *Security Symposium* (2014), USENIX, pp. 17–32.

[17] Freeman, K. Algorithmic injustice: How the wisconsin supreme court failed to protect due process rights in state v. loomis. *North Carolina Journal of Law & Technology 18*, 5 (2016), 75.

[18] Hayes, J., and Ohrimenko, O. Contamination attacks and mitigation in multi-party machine learning.

[19] Hildebrand, C. The beat goes on, May 2021.

[20] Hummel, R., Hildebrand, C., Modi, H., Conrad, C., Dobbins, R., Bjarnson, S., Belanger, J., Sockrider, G., Alcoy, P., and Bienkowski, T. 2h2020 netscout threat intelligence report, Jun 2021.

[21] IBM. Explainable ai, 2021.

[22] Larson, J., Mattu, S., Kirchner, L., and Angwin, J. How we analyzed the compas recidivism algorithm, 2016.

[23] Lashkari, A. H., Draper-Gil, G., Mamun, M. S. I., and Ghorbani, A. A. Characterization of tor traffic using time based features. In *International Conference on Information System Security and Privacy* (2017).

[24] Lavinia, Y., Durairajan, R., Rejaie, R., and Willinger, W. Challenges in using ml for networking research: How to label if you must. In *Workshop on Network Meets AI & ML* (2020), ACM, pp. 21–27.

[25] McGough, M. How bad is sacramento's air, exactly? google results appear at odds with reality, some say, 2018.

[26] McKinney, W. Data structures for statistical computing in python. In *Python in Science Conference* (2010), S. van der Walt and J. Millman, Eds., pp. 56 – 61.

[27] Moore, A. W., and Zuev, D. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS* (2005), pp. 50–60.

[28] Muthukumar, A., and Durairajan, R. Denoising internet delay measurements using weak supervision. In *International Conference On Machine Learning And Applications* (2019), IEEE, pp. 479–484.

[29] O'neil, C. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Crown, 2016.

[30] Papernot, N., McDaniel, P., Sinha, A., and Wellman, M. P. Sok: Security and privacy in machine learning. In *European Symposium on Security and Privacy* (2018), IEEE, pp. 399–414.

[31] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[32] Pustozerova, A., and Mayer, R. Information leaks in federated learning. In *Network and Distributed System Security Symposium* (2020).

[33] Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. Snorkel: Rapid training data creation with weak supervision. In *VLDB Endowment* (2017), vol. 11, NIH Public Access, p. 269.

[34] Ratner, A. J., De Sa, C. M., Wu, S., Selsam, D., and Ré, C. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems 29* (2016), 3567–3575.

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

[36] Rossow, C. Amplification hell: Revisiting network protocols for ddos abuse. In *Network and Distributed System Security Symposium* (2014).

[37] Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence 1*, 5 (2019), 206–215.

[38] Rudin, C., and Radin, J. Why are we using black box models in ai when we don't need to? a lesson from an explainable ai competition. *Harvard Data Science Review 1*, 2 (2019).

[39] Rudin, C., Wang, C., and Coker, B. The age of secrecy and unfairness in recidivism prediction.

[40] Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *Symposium on Security and Privacy* (2017), IEEE, pp. 3–18.

[41] Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. Stealing machine learning models via prediction apis. In *Security Symposium* (2016), USENIX, pp. 601–618.

[42] Turek, M. Explainable artificial intelligence (xai), 2021.

[43] Varma, P., and Ré, C. Snuba: Automating weak supervision to label training data. In *VLDB Endowment* (2018), vol. 12, NIH Public Access, p. 223.

[44] Varshney, K. R., and Alemzadeh, H. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big data 5*, 3 (2017), 246–255.

[45] Wang, C., Han, B., Patel, B., Mohideen, F., and Rudin, C. In pursuit of interpretable, fair and accurate machine learning for criminal recidivism prediction.

[46] Wexler, R. When a computer program keeps you in jail: How computers are harming criminal justice. *New York Times 13* (2017).

[47] Yang, J., Fan, J., Wei, Z., Li, G., Liu, T., and Du, X. Cost-effective data annotation using game-based crowdsourcing. *VLDB Endowment 12*, 1 (2018), 57–70.

[48] Yuen, M.-C., King, I., and Leung, K.-S. A survey of crowdsourcing systems. In *SocialCom* (2011), IEEE, pp. 766–773.