# FARE: Enabling Fine-grained Attack Categorization under Low-quality Labeled Data

Junjie Liang[1§], Wenbo Guo[1§], Tongbo Luo[2], Vasant Honavar[1], Gang Wang[3], Xinyu Xing[1]
[1]The Pennsylvania State University [2]JD Security Research Center, [3]University of Illinois at Urbana-Champaign
{jul672, wzg13, vhonavar, xxing}@ist.psu.edu, irobert0126@gmail.com, gangw@illinois.edu

***Abstract*—Supervised machine learning classifiers have been widely used for attack detection, but their training requires abundant high-quality labels. Unfortunately, high-quality labels are difficult to obtain in practice due to the high cost of data labeling and the constant evolution of attackers. Without such labels, it is challenging to train and deploy targeted countermeasures.**

**In this paper, we propose FARE, a clustering method to enable fine-grained attack categorization under low-quality labels. We focus on two common issues in data labels: 1) missing labels for certain attack classes or families; and 2) only having coarse-grained labels available for different attack types. The core idea of FARE is to take full advantage of the limited labels while using the underlying data distribution to consolidate the low-quality labels. We design an *ensemble model* to fuse the results of multiple unsupervised learning algorithms with the given labels to mitigate the negative impact of missing classes and coarse-grained labels. We then train an input transformation network to map the input data into a low-dimensional latent space for fine-grained clustering. Using two security datasets (Android malware and network intrusion traces), we show that FARE significantly outperforms the state-of-the-art (semi-)supervised learning methods in clustering quality/correctness. Further, we perform an initial deployment of FARE by working with a large e-commerce service to detect fraudulent accounts. With real-world A/B tests and manual investigation, we demonstrate the effectiveness of FARE to catch previously-unseen frauds.**

## I. Introduction

Machine learning is widely used to build security applications. Many security tasks such as malware detection and abuse/fraud identification can be formulated as a supervised classification problem [31], [45], [64], [10], [36], [19], [17]. By collecting and labeling benign and malicious samples, defenders can train supervised classifiers to distinguish attacks from benign data (or distinguish different attack types).

A key challenge faced by these supervised classifiers is that their training requires *abundant high-quality labels*. Many supervised models, especially deep-learning models, are data-hungry, requiring a large quantity of labeled data to achieve a decent training outcome. In addition, the labels need to have good coverage of *all the attack types of interest*. A classifier cannot reliably detect a certain type of attack unless the defender knows the attack exists and has collected labeled data for training.

Unfortunately, in practice, obtaining abundant high-quality labels is difficult. This is particularly true for security applications, due to the high cost of data labeling and the evolving nature of attacks. Data labeling is expensive because it requires manual efforts. Unlike labeling images or text, investigating new attack samples (*e.g.*, new malware families) requires substantial expertise, and often takes a longer time. As such, only a small portion of data samples can be labeled manually. Even for the labeled samples, the quality of the labels is often far from satisfying. There are two common issues faced by different security applications:

The first common issue is the *missing classes* in the labeled data. Take malware detection for example. The malware ecosystem is constantly evolving with new malware families appearing frequently over time [56]. As a result, the labeled dataset might miss certain malware families. Using a dataset with missing classes, the trained classifiers would have a hard time detecting related malware.

The second common issue is *coarse-grained labels*. Due to the lack of time or expertise of the analysts, the provided labels often lack specificity or contain errors. For example, for malware attribution, malware of different families may be incorrectly labeled as the same family; For online abuse classification, scrapers and trolls may be assigned to a generic "abusive" label. In practice, coarse-grained labels pose a key challenge to deploying timely and targeted countermeasures. For example, different malware has different kill chains, and scrapers and trolls should be given different penalties.

**Proposed Solution.** In this paper, we aim to enable fine-grained attack categorization using low-quality labels. The goal is to discover the clustering structures in the data to assist human analysts to derive high-quality labels. We propose FARE, a semi-supervised method to address the issues of both *missing classes* and *coarse-grained labels* in poorly-labeled datasets. At the high-level, FARE's input is a dataset where only a small portion of the data is labeled, and the labels are of a low-quality. After running FARE, it outputs the clustering assignment for all the data samples. The data samples are expected to be either correctly clustered under the known labels or form new groups to represent the new labels. By correctly recovering the clustering structures in the input dataset, FARE provides the much-needed support for human analysts to generate high-quality labels.

The core idea of FARE is to take full advantage of the limited labels while using the underlying data distribution to

§Equal contribution.

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.23xxx
www.ndss-symposium.org

consolidate the low-quality labels. More specifically, we design an *ensemble model* to fuse the results of multiple unsupervised learning algorithms with the given labels. This helps to mitigate the negative impact of missing classes and coarse-grained labels, and reduce the randomness of the learning outcome. Based on the fused labels, we design an *input transformation network* by extending the basic idea of metric learning [26]. The network maps the input data into a low-dimensional latent space, which makes it easier to identify fine-grained clusters.

**Experimental Evaluation.** We evaluate FARE with two popular security applications: malware categorization and network intrusion detection. We use existing datasets of 270,000 malware samples and 490,000 network events to perform controlled experiments. More specifically, by omitting different classes or merging data labels, we simulate different scenarios where only limited low-quality labels are available. We compare FARE with the state-of-the-art semi-supervised learning algorithms as well as unsupervised algorithms. Our results show that FARE significantly outperforms existing methods when there are missing classes or coarse-grained labels in the data, and maintains a comparable performance when the data labels are correct. We find that most existing methods have the implicit assumption that the classes in the labels are complete, and thus perform poorly when this assumption is violated. In addition, we show that FARE is less sensitive to the variations of datasets, and the ratio of available labels. This confirms the benefits of fusing unsupervised learning results with given labels to increase system stability. Finally, we show that the computational overhead of FARE is comparable to commonly-used clustering algorithms.

**Testing on a Real-world Service.** We work with an industrial partner to test FARE in their production environment to detect fraudulent accounts in a large e-commerce service. As the initial testing, we apply FARE to a sample of 200,000 active user accounts. The dataset only has 0.5% of confirmed fraudulent account labels, and 0.1% of confirmed trusted account labels. Through an A/B test, we show that FARE helps to discover previously-unseen fraudulent accounts. By initiating two-factor re-authentication requests to the detected accounts, we find 0% of them can successfully re-authenticate themselves, confirming a low false-positive rate. Further manual investigation reveals new attack types such as accounts exploiting mistagged prices for bulk product purchasing.

In summary, this paper makes three key contributions.

- We propose FARE to address the problem of low-quality data labels, a common challenge faced by learning-based security applications. We introduce a series of new designs to enable fine-grained attack categorization when the labeled data has missing classes or coarse-grained labels.
- Through experiments, we demonstrate existing semi-supervised and unsupervised methods are not capable of handling such low-quality labels. We show that FARE significantly outperforms existing methods in recovering the true clustering structure in the data.
- We tested FARE in a real-world online service system. We demonstrate the usefulness of FARE to analyze and categorize fraudulent accounts.

To facilitate future research, we release the code of FARE, and the malware and intrusion datasets used in this paper[1].

## II. Background and Problem Scope

We start by describing the background of three key security applications and the problems caused by missing-classes or coarse-grained labels. Then, we discuss our problem scope and assumptions.

### A. Security Applications

**Malware Identification and Classification.** Researchers have used machine learning methods to identify malware from benign software (*i.e.*, identification) and classifying malware into specific families (*i.e*, attribution) [1], [81], [56], [82]. Most existing works focus on the supervised learning setting (in which a fully and correctly labeled malware dataset is available) and have demonstrated promising performance of machine learning models. However, the problem becomes more challenging in semi-supervised learning or unsupervised learning settings when labels are incomplete. Labels are incomplete for two main reasons. First, malware evolution: one malware family could evolve into hundreds or even thousands of malware variants in a short period of time [73]. Second, labeling malware usually requires manual efforts from domain experts, which is a time-consuming process.

**Network Intrusion Detection.** Existing network intrusion detection systems can be categorized into rule-based system and anomaly-based system [65], [47], [51], [15], [38]. Rule-based systems detect a known attack by matching the attack with the existing patterns stored in the knowledge base. These systems are usually accurate on well-studied intrusions but can fail to detect previously-unseen attacks. Anomaly-based systems rely on unsupervised machine learning to detect out-of-distribution samples. In practice, security platforms often combine both systems for a better outcome. However, it is still plausible for attackers to adapt their behaviors to evade such detection systems. Identifying and characterizing such evasion attacks requires manual investigations from domain experts, which again is a time-consuming process.

**Fraudulent Account Detection.** Online service providers face serious threats from fraudulent accounts that are created for malicious activities (*e.g.,* spam, scam, illegal content scraping, and opinion manipulation) [14], [13], [68], [75], [71], [23], [33]. A recent report shows that fraudulent credit card accounts affected more than 250,000 U.S. consumers [27]. Similarly, detecting fraudulent accounts has been a cat-mouse game. The defenders are struggling with labeling new types of fraudulent accounts as they change their behaviors to evade detection.

### B. Problem Scope and Assumptions

A common challenge faced by these security applications is data labeling. While the data labeling problem also exists in other application domains (*e.g.*, image analysis and natural language processing), we argue that two characteristics of security applications make the problem more concerning. First, unlike labeling images, labeling security data requires domain expertise to perform in-depth manual analysis (and thus more

[1] https://url-to-be-released-for-camera-ready

time consuming). Second, attacker behavior shift is a norm in the security domain, which puts higher pressure on labeling data in a timely fashion. In practice, security analysts can only label a small subset of samples among a large volume of data. Below, we discuss two critical issues:

**Missing Classes.** The first issue is that the labeling is often incomplete, which means not all the incoming data samples have a label. Even for the labeled samples, it is difficult to guarantee that the labels perfectly cover all the attack categories. Take malware for example, it is unrealistic to assume that the security analysts are aware of all the malware families in the wild. As such, a common practice is to conservatively leave the previously unseen families as *unlabeled data*. With unlabeled data and missing classes, the trained classifier will have a bad performance when deployed in practice.

**Coarse-grained Labels.** Another common situation is that analysts mistakenly group data from several classes into one class, due to the lack of knowledge or time for in-depth analysis. For example, given several malware families under one parent family, an analyst who is only aware of the parent family could label all child families as the parent class. Worse, it is also possible for inexperienced analysts to assign two different malware families under the same family. Similarly, in online services, different attacks (scrapers, spammers, trolls) may be assigned to the same generic "abuse" label.

Fine-grained labels are the key to deploying effective countermeasures [21]. For example, different malware families usually have different kill chains (from malware delivery to exploitation, command & control, and data exfiltration/encryption). Knowing the fine-grained malware label allows defenders to use targeted countermeasures to disrupt the kill chain before the damage is made. Similarly, in large online services, different abusive accounts require different types of penalties. For example, network throttling and CAPTCHA can be effective against automated scrapers, but are ineffective against trolls controlled by real users.

**Problem Definition.** Given a dataset with $n$ true classes, we define the two problems as the following.

- *Missing classes*: labels are completely missing for $n_c$ classes. For the remaining $n-n_c$ classes, only a small portion of their samples have labels available.
- *Coarse-grained labels*: $n_g$ original classes are labeled as one union class. For these $n-n_g+1$ classes, only a small portion of their samples are labeled.

Our goal is to recover the true clustering structure of the input data by leveraging the limited low-quality labels. After processing the input dataset, we aim to 1) determine there are $n$ clusters in the dataset; and 2) correctly assign all the data samples (including the unlabeled samples) to the $n$ clusters.

Note that, our output is the clusters of data samples, but these clusters do not yet have "labels" (*i.e.*, what type of attack each cluster represents). In practice, human analysts will then inspect the output clusters to assign labels (*i.e.*, determining the attack type). This can be done by referencing the known labels or manually analyzing *a small number* of samples per cluster (see Section §VI and Appendix-F for more details). By recovering the correct clustering structures, we empower the human analysts to discover the previously missing classes and fine-grained sub-classes.

In this paper, we focus on recovering the true clustering structure. The human labeling part is out of the scope of this paper (*i.e.*, potential user studies are future works).

**Assumptions.** We assume the given labels of the known classes are correct ($n-n_c$ classes in the missing class setting, and $n-n_g+1$ classes in the coarse-grained label setting). In other words, we assume a small number of samples for the well-known classes are labeled correctly in the input dataset.

### C. Possible Solutions and Limitations

Before introducing our system design, we first briefly discuss the possible directions and their limitations.

The most straightforward direction is to ignore the low-quality labels and directly train a *supervised classifier* on the available labeled training data [1], [32], [44]. However, with limited and low-quality labels, a supervised classifier faces challenges to learn the accurate decision boundary of the true classes. More importantly, supervised classifiers cannot handle new classes that are not part of the labeled data. To detect new classes, an augmentation method is to use the *prediction probability* (or confidence) of the classifier [34]. Intuitively, a low prediction probability could indicate the input sample is from a new class. However, this approach has major limitations. First, confidence score is known to be unreliable on out-of-distribution (OOD) samples. A classifier could easily misclassify OOD samples with high confidence [30], [34]. Second, confidence score cannot group on new samples, which is inconvenient for the subsequent labeling process.

An opposite direction is to ignore any given labels and apply *unsupervised clustering* [28], [20] or *clustering ensemble* methods [67] to the training data. This direction could avoid misleading information introduced by the low-quality labels. However, as is shown later in Section §IV, without the guidance of any labels, unsupervised methods are less effective compared to those that can leverage the given labels.

A more promising direction is to apply a *semi-supervised learning* methods [59], [6]. Semi-supervised learning could leverage both the given labels and unlabeled data to learn a more accurate clustering structure of the true classes. Unfortunately, existing semi-supervised learning methods rely on a strong assumption. That is, there are (at least) a few labeled samples available in *all the classes* in the training set. In other words, they do not assume missing classes or coarse-grained labels in the training set. As is shown later in Section §IV, their performances are significantly jeopardized when there is a violation of this assumption.

Finally, a related direction is *(generalized) zero-shot learning methods (GZSL/ZSL)*, which can be used to identify previously unseen classes in the testing data [60]. These methods treat the training data as the first task and try to transfer the learned model from the first task to the second task (*i.e.,* testing data). GZSL/ZSL methods assume the testing data contains unseen classes (that do not exist in the first task). However, to enable successful knowledge transfer, GZSL/ZSL methods require well-labeled training samples in the first task. Also,

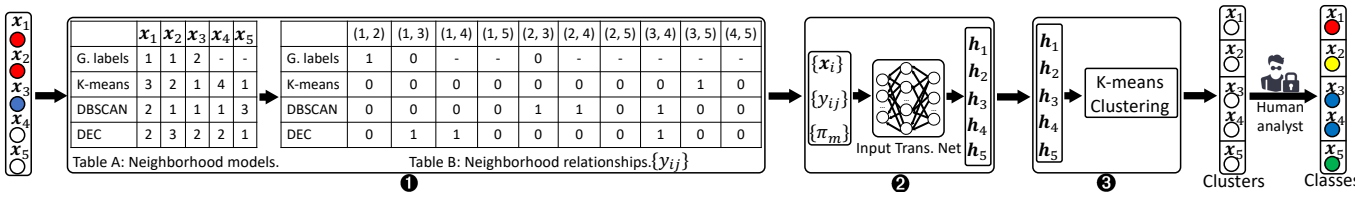



Fig. 1. The overview of FARE. The colors on the dots indicate the given labels (top) and the final clustering results (bottom). Dots of the same color have the same label; the transparent dot represents an unlabeled sample. The numbers in Table A refer to the cluster indexes. The 0s and 1s in Table B represent the neighborhood relationship. "-" stands for "not available". "G. Labels" is short for "Given Labels".

they need well-labeled side information (*e.g.,* in image classification tasks, side information are shared and nameable visual properties of objects). These requirements make GZSL/ZSL methods not suitable to solve our problem: (1) we assume the training labels are limited; (2) most security applications do not have well-defined notions for "side information". We have provided a brief supporting experiment in Appendix-B.

## III. Methodology of FARE

We design a system FARE to address the labeling problems mentioned above. FARE is short for "**F**ain-grained **A**ttack Categorization through **R**epresentation **E**nsemble". In the following, we first explain the intuitions behind the system design, followed by the technical details of each component. Finally, we describe an unsupervised version of FARE.

### A. Overview of System Design

In Figure 1, we provide an example to explain the workflow. The input of FARE is a set of data samples (*i.e.*, feature vectors). Their labels are incomplete or contain errors. In this example, we have 5 data samples. We use different colors to distinguish the *given labels* of these samples ( $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ have the same "red" label; $\boldsymbol{x}_3$ has the "blue" label; and $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$ are not labeled yet).

The given labels of the inputs contain errors. More specifically, the *true labels* of the 5 samples are shown on the rightmost side of Figure 1. There should be 4 ground-truth classes. Their true grouping is: $\{\boldsymbol{x}_1\}$, $\{\boldsymbol{x}_2\}$, $\{\boldsymbol{x}_3, \boldsymbol{x}_4\}$, $\{\boldsymbol{x}_5\}$.

Under this setting, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ represent the coarse-grained label problem; $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$ represent the missing class problem. After FARE processes the input data, our goal is to recover the true grouping of the 5 samples. After that, human analysts could inspect the clusters and assign the labels accordingly. In this example, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are correctly split into two fine-grained classes ("red" and "yellow"). $\boldsymbol{x}_3$ and $\boldsymbol{x}_4$ are grouped under the "blue" class. $\boldsymbol{x}_5$ then forms a new class "green".

To achieve this goal above, we design FARE to process the datasets with three key steps. In step ❶, we mitigate the uncertainties of the labels using an *ensemble method*. The idea is to fuse the results of multiple unsupervised algorithms with the given labels, to use the underlying data distribution to consolidate the labels. In step ❷, we transform the input space into a more compressed hidden space to represent the data distribution. The low-dimensional space allows us to perform accurate data clustering. In step ❸, we perform a K-means clustering on the compressed data to identify the underline cluster structures of the input data. Finally, we defer to human analysts to assign labels to the output clusters. The clusters produced by FARE could help human analysts identify missing classes and the fine-grained classes.

**Augmenting Labels with Unsupervised Learning Results.** In step ❶, we consider the given labels untrustworthy due to the missing classes and the coarse-grained labels. To consolidate the given labels, the only available information source is the data samples themselves. As such, we propose to use *multiple unsupervised learning algorithms* to extract the underlying data distributions (manifolds) to mitigate the uncertainty of the original labels.

More specifically, we fuse the labels from $M$ different sources. Among them, one source is the given labels, and the other $M-1$ sources are different unsupervised clustering algorithms. The reason to use *multiple* clustering algorithms is to reduce biases. Existing clustering algorithms intrinsically make assumptions about the data distribution, and they work well only when such an assumption is satisfied (*e.g.,* K-means assumes data are represented in Euclidean space). However, in practice, we cannot validate these assumptions without trial-and-error. For this reason, we apply multiple clustering methods and fuse their results. In this way, the system is less sensitive to the assumption made by certain clustering method (validated in Section §IV). In addition, clustering algorithms can be sensitive to hyper-parameters. To minimize such influence, we apply each clustering algorithm multiple times with different hyper-parameters and each setting has its own row. After clustering, the results from each model[2] are shown in Table A in Figure 1.

To fuse the labels from the $M$ sources, we need to find a uniform way to represent the clustering results. We solve this problem by constructing a *neighborhood relationship table*, which is Table B in Figure 1. In this table, each column represents a pair of input samples; each row represents either a clustering algorithm (row-2 to row-4) or the original given labels (row-1). This table describes the pair-wise relationship between all pairs of input samples.

Given a clustering algorithm, if a pair of input samples are grouped into the same cluster, we set their relationship value as 1 (0 otherwise). Similarly, for the original given labels, if the two samples share the same label, we set their relationship value as 1. If they have different labels, we set the value to 0. If at least one sample in the pair is unlabeled, we set their relationship as "not available" ("-"). The neighborhood relationship table makes it possible to fuse the results across algorithms because we don't need to align the specific clusters to the specific labels. Instead, all the algorithms share the same

[2]For simplicity, in the example of Figure 1, we only apply each clustering algorithm once with one set of hyper-parameters.

format that captures the pair-wised relationships of the input data samples. For convenience, we refer to each row in this table as a *neighborhood model*. In Figure 1, we have $M = 4$ neighborhood models.

To merge the results of $M$ neighborhood models, we introduce a hyper-parameter $\{\pi_m\}_{m=1}^M$ to represent the weight of each model. A higher weight means the model is more important. In Section §III-D, we describe how to calculate $\pi_m$ via a validation set.

**Input Transformation.** In the next step ❷, we transform the input samples and labels into low-dimensional vectors as an accurate representation of the input data space. This low-dimensional space allows us to cluster the input data into fine-grained clusters. Given the neighborhood relationship table $\{y_{ij}^m\}_{m=1}^M$, and the model weights $\{\pi_m\}_{m=1}^M$, we train an input transformation network to transforms an input $\boldsymbol{x}_i$ to a hidden representation $\boldsymbol{h}_i$. The network is trained to achieve two goals. First, we want to transform the inputs into more separable representations while preserving the pair-wise neighborhood relationships. In other words, the hidden representation still reflects the original data distribution, but should make these samples easier to cluster. Second, the transformation function will project the high dimensional input to a lower-dimensional Euclidean space. As mentioned in Section §VII, traditional clustering methods suffer from the curse of dimensionality. A low-dimensional space enables more efficient clustering.

Note that this input transformation network is different from traditional unsupervised auto-encoder [46] that are used to compress the original inputs. The key difference are two-folds. First, auto-encoder is unsupervised, while FARE's transformation network utilizes both unlabeled and labeled samples. Second, auto-encoder is trained for input reconstruction. FARE is trained to *contrast* different samples to learn a more separable space, which benefits later clustering.

**Final Clustering.** In the final step ❸, we simply apply the K-means algorithm on the hidden representations to generate more fine-grained clusters. We choose K-means with the following considerations. First, K-means works particularly well if Euclidean distance. The input transformation in the previous step has mapped inputs to a Euclidean space. Second, other candidate algorithms such as DBSCAN are not suitable because their assumptions do not match well with the hidden representations or they do not rely on the notion of distance (*e.g.,* density-based algorithms) for clustering. The main task in this step is to determine the final number of clusters $K$. In the following, we will discuss our method in detail.

### *B. Technical Details*

In this section, we present the technical details of each component in `FARE`. We start by defining key notations. Given an input dataset $\mathcal{X} = \{\mathcal{X}_\mathcal{Y}, \mathcal{X}_\mathcal{U}\}$, where $\mathcal{X}_\mathcal{Y}$ corresponds to the labeled samples set and $\mathcal{X}_\mathcal{U}$ denotes the set of unlabeled samples. Within the dataset, each sample $\boldsymbol{x} \in \mathcal{R}^{p\times 1}$ is a $p$ dimensional vector. If the sample has a label, the label is represented by an integer value, indicating the corresponding sample's class. We use $\{\cdot\}$ as an abbreviation for $\{\cdot\}_{m=1}^M$.

**Ensemble of Neighborhood Models.** In ❶ of Figure 1, we compute the ensemble of labels from multiple unsupervised algorithms and the given labels. We define a set of $M$ neighborhood models (denoted as $\mathcal{M}$), and each model is used to decide a set of pair-wise neighborhood relationships of samples in $\mathcal{X}$. As shown in Figure 1 Table A, one of the neighborhood models is the given labels and the other $M-1$ neighborhood models are the clustering algorithms.

For each neighborhood model in $\mathcal{M}$, we then decide the pair-wise neighborhood relationships for samples in $\mathcal{X}$. Given a pair of samples ($\boldsymbol{x}_i$, $\boldsymbol{x}_j$), the neighborhood relationship captured by the $m^{th}$ model is denoted as $y_{ij}^m$. As mentioned in Section §III-A, $y_{ij}^m = 1$ if the two samples are clustered into the same cluster (0 otherwise) by the $m^{th}$model . For the "given labels", the same rule applies (but if either input is in the unlabeled set $\mathcal{X}_\mathcal{U}$, $y_{ij}^m$ is unavailable).

To aggregate the neighborhood relationships from all the models in $\mathcal{M}$, we set weights $\pi_m$ on each neighborhood model. To calculate this $\pi_m$, we first define a priori $p_m$ for each model. This $p_m$ is also a hyper-parameter (configuration details are in Section §III-D). After deciding the value of $\{p_m\}$, we then calculate $\{\pi_m\}$ by normalizing $\{p_m\}$ using a softmax function: $\pi_m = \frac{e^{p_m}}{\sum_{m\in\mathcal{M}} e^{p_m}}$.

**Input Transformation Network.** The input transformation network aims to transform the input samples into a low-dimensional hidden space to identify the underlying clusters. Based on $\{y_{ij}^m\}$ and $\{\pi_m\}$, we want to learn a network $f$ to map any input sample $\boldsymbol{x}$ from $\mathcal{X}$ to $\boldsymbol{h}$ in a hidden space. As discussed in Section §III-A, the hidden space should (1) maintain an accurate representation of the neighborhood relationships of the input samples, and (2) make it easier to perform clustering.

To achieve these goals, we first apply a metric learning loss to train the transformation network. Metric learning [79], [69] transforms the input samples into hidden representations while keeping the sample distance (*i.e.,* the relative distance between pair-wise of samples) consistent with that in the input space. Mathematically, given a pair of samples $\boldsymbol{x}_i, \boldsymbol{x}_j$, and neighborhood relationship $y_{ij}$, a typical pair-wise metric learning loss has the following form [26]:

$$\tilde{\mathcal{L}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = y_{ij} d_{ij}^2 + (1 - y_{ij})(\alpha - d_{ij})_+^2 , \tag{1}$$

where $(\cdot)_+$ is short for $\max(0, \cdot)$ and $d_{ij}$ is the distance of the hidden representations of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. This loss function ensures that the distance of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is minimized in the latent space if they are neighbors (*i.e.,* belonging to the same cluster). Oppositely, we maximize their distance up to a radius defined by $\alpha > 0$, such that dissimilar pairs contribute to the loss function only when their distance is within this radius.

With this metric learning loss, we learn the hidden representation of the input samples such that inputs from the same class have a smaller distance than those from different classes. This makes the hidden representations from different classes more separable. Another benefit is that metric learning converts the samples into representations in the Euclidean space, where the Euclidean distance can be used as the distance function. To be specific, we define the distance function of $\boldsymbol{x}_i, \boldsymbol{x}_j$ in the hidden space as follows:

$$d_{ij} = d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{h}_i - \boldsymbol{h}_j\|_2 . \tag{2}$$

Here, $\boldsymbol{h_i} = f(\boldsymbol{x}_i)$ is the hidden representation of $\boldsymbol{x}_i$. It should be noted that we only need to define distance function in the hidden space since the neighborhood relationships of the original input samples have already been captured by $y_{ij}$.

We can integrate the multiple sets of neighborhood relationships into the loss function in Equation (1). To be specific, given a sample pair $\boldsymbol{x}_i, \boldsymbol{x}_j$, their neighborhood relations $\{y_{ij}^m\}$, and the model weights $\{\pi_m\}$, the loss function of this sample pair is defined as follows:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sum_{m \in \mathcal{M}} \pi_m \delta_{ij}^m \tilde{\mathcal{L}}(\boldsymbol{x}_i, \boldsymbol{x}_j | m) \\ &= \sum_{m \in \mathcal{M}} \pi_m \delta_{ij}^m \left[ y_{ij}^m d_{ij}^2 + (1 - y_{ij}^m)(\alpha - d_{ij})_+^2 \right] .\end{aligned} \tag{3}$$

This loss function also handles the special cases when $y_{ij}^m$ is "unavailable" for (incomplete) given labels. We introduce an indicator $\delta_{ij}^m$: if $y_{ij}^m$ is unavailable, we set $\delta_{ij}^m = 0$ (and 1 otherwise).

The loss function in Equation (3) has the form of total probability [52], where $\pi_m$ can be taken as the priori of each neighborhood model. The final loss can be calculated by integrating the individual loss obtained from each set of neighborhood relationships obtained from each neighborhood model. In other words, this loss function only minimizes (or maximizes) the distance between a pair of samples in the hidden space when most of the neighborhood models agree that they are neighbors (or non-neighbors). The loss over the entire dataset (*i.e.,* $\mathcal{X}$) is computed by averaging the loss on each sample pair plus a regularization term on model parameters:

$$\mathcal{L} = \frac{1}{|\mathcal{X}|^2} \sum_{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{x}_j) + \lambda \left\| \theta \right\|_2 , \tag{4}$$

where $|\mathcal{X}|$ is the number of samples in $\mathcal{X}$, $\theta$ represents the parameters of $f$, and $\lambda$ controls the regularization strength.

Our network is a Multilayer Perceptron (MLP) with multiple hidden layers and one output layer. We set the output layer to have a much lower dimensionality than the original input (*i.e.,* $\boldsymbol{h} \in \mathcal{R}^{q \times 1}$, where $q << p$). We train this network by minimizing the loss function in Equation (4) with the training algorithm introduced in Section §III-D. Appendix-A lists the exact network architectures used in this paper.

**Final Clustering with K-means.** Given the latent representations $\boldsymbol{h}$, we apply K-means to categorize these samples into $K$ different clusters. Technically, K-means has two steps: 1) initializing the cluster centers and allocate each data point to its nearest center; 2) updating each cluster center using the mean feature vector of the data points in the corresponding cluster. The update procedure iterates between these steps until the change of cluster centers between two consecutive iterations is below a certain threshold. More formally, in the first step, the cluster that a given point $\boldsymbol{x}_i$ belongs to is solved by:

$$c_i = \arg\min_k \left\| \boldsymbol{h}_i - \boldsymbol{s}_k \right\|_2^2 , \tag{5}$$

where $\boldsymbol{s}_k$ is the feature vector for cluster center $k$.

On the second step, let $\mathcal{S}_k$ denotes the set of data points in cluster $k$, then the cluster center $s_k$ is updated using:

$$\boldsymbol{s}_k = \frac{1}{|\mathcal{S}_k|} \sum_{\boldsymbol{x}_i \in \mathcal{S}_k} \boldsymbol{h}_i . \tag{6}$$

**Number of Clusters.** The performance of K-means depends on the choice of $K$. In this paper, we select $K$ under the guidance of silhouette coefficients, a popular unsupervised clustering evaluation metrics that is used to determine the degree of separation between clusters [61]. Formally, let $a_i$ be the mean distance between sample $\boldsymbol{x}_i \in \mathcal{S}_{c_i}$ and all other data points in the same cluster $c_i$ and $b_i$ be the minimum value of mean distance between $\boldsymbol{x}_i$ and all other data points in cluster k ($k \neq c_i$) across all $k = 1, ..., K$. Then the silhouette coefficient of sample $\boldsymbol{x}_i$ is defined as:

$$Sil_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}, \text{if } |\mathcal{S}_{c_i}| > 1 . \tag{7}$$

For cases where $|\mathcal{S}_{c_i}| = 1$, we simply set $Sil_i = 0$. Given the $Sil_i$ of each sample, the silhouette coefficient of a clustering results with $K$ clusters (*i.e.,* $Sil(K)$) is defined as the maximum value of $\tilde{Sil}_k$ across all clusters, where $\tilde{Sil}_k$ is the mean silhouette coefficients over the $Sil_i$ of all samples within cluster $k$. The final choice of $K$ is then determined by the value with the largest $Sil(K)$.

### C. Unsupervised Extension of FARE

While FARE is designed to take low-quality labels as inputs, it can be extended to an unsupervised version (without taking any labels). Recall that FARE obtains $M$ sets of neighborhood relationships (one from given labels and $M - 1$ from clustering algorithms). When the "given labels" are completely unavailable, FARE can work with the $M - 1$ clustering algorithms to obtain the neighborhood relationships. In this way, we can use FARE as an unsupervised method.

In comparison with the existing clustering methods, the advantage of FARE is it fuses the neighborhood relationships from multiple models. We expect FARE to be less sensitive to the variations in input data distribution and hyper-parameters, and thus produce more reliable results. We will validate this intuition in Section §IV.

### D. Training Strategy and Hyper-parameters

We apply Adam optimizer to minimize the loss function in Equation (4) and set its learning rate as 0.001. Technical details of this optimization technique can be found in [41].

**Neighborhood Models.** We select three different clustering methods: K-means [28], DBSCAN [20], and DEC [78]. The rationale behind these choices is as following: K-means, DBSCAN, DEC, and DAGMM are four existing clustering methods that have been applied to different security applications. DAGMM is designed for anomaly detection but not for multi-classes clustering tasks. As such, we use the other three methods for FARE. These methods are reasonably diversified to meet our needs. As mentioned before, we apply the selected methods with different hyper-parameters and form $M - 1$ neighborhood models in total. The choice of $M$ and clustering parameters is further discussed in Appendix-A.

**Hyper-parameters.** We define the following hyper-parameters in FARE: the distance radius $\alpha$, the neighborhood model weight $\{p_m\}$, and the regularization coefficients $\lambda$. In this paper, we fixed $\alpha = q$, where $q$ is the output dimension of $f$, and set $\lambda$ to a small value $0.01$. The most important hyper-parameter is the neighborhood model weights $\{p_m\}$. Empirically, we find that it is useful to use different weights for the supervised neighborhood model (*i.e.,* the "given labels") and the unsupervised models. However, among the $M-1$ unsupervised models, we can simply use the same weight to reduce the complexity of parameter tuning while still getting comparable results. For simplicity, in this paper, we set $p_m = 1$ for all the $M-1$ clustering models, and only tune a single $p_1$ to adjust the weight for the "given labels". To determine $p_1$, we use a validation set during training. That is, we set the $p_1$ as the optimal value from $[1, 10]$ that yields the highest adjusted mutual information (AMI) on the *labeled validation samples*. AMI is our evaluation metric, explained in Section §IV-A. More details about the hyper-parameters are in Appendix-A.

**Computational Overhead.** Compared with existing clustering algorithms, FARE has introduced a few additional steps. However, the computational overhead of FARE is comparable to existing clustering algorithms. We will provide the empirical evaluation in Section §IV-C, and discuss the asymptotic complexity in Section §VI.

## IV. Evaluation

We evaluate the effectiveness of FARE on two security applications: malware categorization and network intrusion detection. We focus on four key aspects: 1) validating our design choices, 2) comparing FARE with the state-of-the-art semi-supervised and unsupervised algorithms, 3) evaluating the computational overhead of FARE, and 4) evaluating the sensitivity of FARE to label quality. Later in Section §V, we will describe our experience deploying and testing FARE in a real-world online service to detect fraudulent accounts.

### A. Experimental Setup

**Malware Categorization.** We choose a malware dataset with 270,000 samples[3]. The dataset contains 6 different classes, including one benign class of 150,000 samples and five malware classes of 120,000 samples. For malware classes, the number of samples per class ranges from 15,000 to 37,500. We construct the training set by randomly selecting $70\%$ of samples and used the rest of the samples as the testing set. $20\%$ of labeled samples randomly selected from the training set are held out for validation. Note that we split data randomly instead of splitting temporally [56] because we are performing data clustering to identify fine-grained malware families instead of performing prediction tasks. In this dataset, each sample is represented as a vector of $100$ features, indicating the sandbox behavior of the corresponding software.

**Network Intrusion Detection.** We select the widely used KDDCUP dataset [37]. Each sample is a vector of $120$ features, representing the corresponding network traffic behaviors (See [70] for the detailed feature description). While this dataset is not new, it provides an opportunity to evaluate FARE on highly imbalanced data. In this paper, we selected a subset with $9$ classes, one of which has 97,278 normal network traffic and the rest $8$ classes represent $8$ types of intrusions.[4] Our selected dataset has $493,346$ samples. Note that the selected classes cover $99.8\%$ of the samples in the dataset. We remove the remaining $0.2\%$, because they will be treated as noise by most learning algorithms. We randomly split the dataset into training and testing set with a ratio of 70:30, and randomly pick $20\%$ of the labeled training samples as the validation set.

**Evaluation Metric.** The output of FARE is a set of clusters. To assess the clustering quality, we use a commonly used metric called Adjusted Mutual Information (AMI) [74], which measures the correlation between a cluster assignment and the ground truth labels. In addition, we also consider the traditional *accuracy* metric to provide a different perspective.

Note that the *accuracy* metric has some known limitations to evaluate clustering algorithms. First, different clustering algorithms may produce different numbers of clusters. To compute the accuracy, we need to align clusters to labels. In this paper, given a cluster, we assign the cluster's label as the most prevalent ground-truth label within this cluster. Second, the accuracy metric is sensitive to data distribution across classes [74]. For example, if one class is significantly bigger than all other classes (*i.e.*, the benign class in network intrusion detection), then producing one big cluster may trivially get high accuracy.

For this reason, we use AMI as the primary metric. We only report accuracy for selective experiments as reference (*e.g.*, Table I). To compute AMI, the first step is to draw the contingency table where each element represents the number of overlapped samples in each cluster and the ground truth class. Then we can compute the mutual information [42] between the clustering results and ground truth labels based on the contingency table. Finally, the AMI is obtained by normalizing the mutual information. AMI takes values from $[-1, 1]$, and A higher value indicates a better performance. The key advantage of AMI (compared to the accuracy metric) is AMI normalizes the results of different cluster sizes and is not easily biased by large clusters. The detailed explanation on how to calculate AMI and its advantage over accuracy is in Appendix-C.

**Baseline Methods.** We mainly compare FARE with two popular semi-supervised methods MixMatch [6] and Ladder [59] that have been used for security applications. The two systems are proposed recently (in 2015 and 2019 respectively), and have been highly cited. We also include a supervised deep neural network (DNN) as the baseline. However, our preliminary evaluation quickly reveals that these algorithms, when applied end-to-end, perform poorly under missing classes or coarse-grained labels. We have presented the detailed results in Appendix-B. For example, most of their AMIs would fall under $0.6$ when the training set misses the labels for 2+ classes or has 2+ classes sharing a union label. The reason is that none of these algorithms assume there are missing classes or coarse-grained labels in the training data. As a result, they all set the number of final classes as the number of given labels (or seen classes), and only classify samples to known classes.

In order to *fairly* compare FARE with existing baseline

[3]The dataset [2] is collected and shared by a security company.

[4]We preserved the top-8 intrusion classes ranked by the number of samples: neptune (107,201), smurf (280,790), backscatter (2,203), satan (1,589), ip sweeping (1,247), port sweeping (1,040), warezclient (1,020), teardrop (979).

algorithms, we need to adapt existing algorithms to work under missing classes and coarse-grained labels. More specifically, we slightly amend existing algorithms with the same last step of FARE: the final clustering component and mechanism to determine the number of clusters (step ❸ in Figure 1). For each experiment setup, we first run the existing baseline algorithms on the training data to train their networks. For all three baseline algorithms, the last hidden layer of their networks can output a latent vector of the original input. Instead of using the latent vectors for classification, we run the same the K-means clustering on these latent vectors (step ❸ in FARE) to identify the fine-grained clusters. In this way, these baseline algorithms can perform better under missing classes and coarse-grained labels. We denote the amended version of MixMatch, Ladder, and DNN as MixMatch+, Ladder+, and DNN+, respectively, and use them as our baselines for evaluation.

In addition to semi-supervised baselines, we also compare the unsupervised version of FARE with the base clustering algorithms (DBSCAN, Kmeans, and DEC) and existing ensemble clustering algorithms (CSPA and HGPA [67]).

Note that we did not include GZSL/ZSL methods as main baselines considering the different assumptions and problem setups (see Section §II-C). We only presented a brief experiment in Appendix-B to run our methods against two GZSL methods (*i.e.,* OSDN [4] and DEM [80]). The results confirm that GZSL methods do not work well in our setup.

**Training, Validation, and Testing.** At a high level, we use the training set to identify the final clusters and their centroids. We use the validation set for parameter turning during the training process. The testing data is used to test the quality of the final clusters: for each testing sample, we compute its latent vector and assign it to the nearest cluster based on its distance to the cluster centroids. If not otherwise stated, AMIs/Accuracy are calculated based on testing data for performance evaluation.

More specifically, for FARE, we use the training set to construct the neighborhood models to form the ensemble, learn the parameters for the input transformation network, and identify the centroids for the final clusters. We use the validation set to select the weight for the " given labels" (*i.e.,* $p_1$) and determine the number of final clusters $K$. For baseline algorithms (*i.e.,* MixMatch+, Ladder+, and DNN+), we follow a similar process (but do not need to train the ensemble component).

### B. Experimental Design

We design the four experiments to evaluate the effectiveness of FARE from different aspects. First, we validate the ensemble method used in FARE by comparing its performance with the individual base clustering algorithms and existing ensemble clustering methods. Second, we quantify the advantage of FARE over three baselines in the "missing classes" setting. Third, we run FARE and baselines in the "coarse-grained labels" setting. Finally, we evaluate the robustness of FARE to other factors such as the ratio of available labels and the number of neighborhood models.

**Experiment I: Comparing with Unsupervised Methods.** In this experiment, we want to examine if the ensemble of multiple clustering algorithms indeed introduces benefits. Recall that FARE takes the ensemble of $M$ neighborhood models. By default, we set $M = 151$ where one neighborhood model is the "given labels", and the other 150 neighborhood models are contributed by three clustering algorithms (K-means, DBSCAN, and DEC). More specifically, by varying the hyper-parameters for each clustering algorithm, each algorithm contributes to 50 neighborhood models (150 in total). For the unsupervised version of FARE (Section §III-C), we set $M = 150$ by excluding the model from the "given labels".

In this experiment, we first compare the unsupervised version of FARE with the individual clustering algorithms and the ensemble clustering methods CSPA and HGPA. Given an evaluation dataset, we first remove the labels, and apply the unsupervised version of FARE and the individual clustering algorithms (*i.e.,* K-means, DBSCAN, and DEC) to the dataset. We then apply CSPA and HGPA on top of the 150 neighborhood models. Note that both HGPA and CSPA encountered out-of-memory issues when being applied to the full datasets (due to their $O(n^2)$ memory consumption). As such, we run both methods on $10\%$ of randomly sampled data points. To ensure a fair comparison, we produce one set of result for `Unsup. FARE` on the same $10\%$ of the training dataset.

Next, to explore the benefits of using (partial) labels, we then use $1\%$ of the original label to train FARE, and compare it with the unsupervised FARE. For each setting, we repeat the experiment $50$ times. Note that DBSCAN and ensemble clustering cannot be used to classify testing data. As such, for this experiment, we report their *training* AMI and Accuracy. Finally, to evaluate the computational cost of FARE, we also record the overall training time of each method.

**Experiment II: Missing Classes.** This experiment focuses on the end-to-end performance of FARE under the missing classes setting. More specifically, we construct training datasets that mimic the scenarios where a certain number of classes are missing in the given labels. For each dataset, we randomly selected $n_c$ classes and marked all the training samples in these classes as "unlabeled samples". For the rest of the classes, we only keep $1\%$ of labeled samples for each class and mark the remaining samples as "unlabeled".

In this experiment, we vary $n_c$ to examine its influence upon the system performance. To make sure our results are not biased by the choice of missing classes, we randomly select the classes to mark as "missing". For a given $n_c$, we randomly select $n_c$ classes as missing classes for 10 times. This generates 10 training sets for each $n_c$. In total, we have $|n_c| \times 10$ training sets. On each training set, we run FARE, and our baselines (*i.e.,* DNN+, MixMatch+, and Ladder+) and compute the testing AMI. In addition, we also want to examine the capability of each algorithm in recovering the actual number of classes. We use $K$ to denote the final number of clusters. Finally, we compute the mean and standard deviation of testing AMI and $K$ under each $n_c$ setting (across 10 training datasets). In addition to AMI, accuracy metric is also calculated (for selective settings) in Appendix-C.

**Experiment III: Coarse-grained Labels.** This experiment is an end-to-end evaluation of FARE under the coarse-grained label setting. Suppose a dataset originally contains $n$ class, we randomly selected $n_g$ classes and merged their training samples into a union label. For the rest of the classes, we also

TABLE I. PERFORMANCE COMPARISON BASED ON MEANS AND STANDARD DEVIATIONS OF AMIS, ACCURACY, AND RUNTIME. "Unsup. FARE" REPRESENTS THE UNSUPERVISED VERSION OF FARE. SINCE NEITHER CSPA NOR HGPA SCALE TO THE FULL DATASET, WE REPORT THEIR RESULTS AND THAT OF Unsup. FARE ON 10% OF TRAINING SET.

| Dataset | | MALWARE | | | Network Intrusion | | |
|---|---|---|---|---|---|---|---|
| Metric | | AMI | Accuracy | Runtime (s) | AMI | Accuracy | Runtime (s) |
| Full Training set | FARE | $0.87 \pm 0.01$ | $0.97 \pm 0$ | 434.05 | $0.98 \pm 0$ | $1 \pm 0$ | 8,943.08 |
| | Unsup. FARE | $0.74 \pm 0$ | $0.81 \pm 0.01$ | 432.12 | $0.78 \pm 0$ | $0.99 \pm 0$ | 8,942.52 |
| | Kmeans | $0.47 \pm 0.12$ | $0.51 \pm 0.04$ | 26.99 | $0.39 \pm 0.18$ | $0.64 \pm 0.12$ | 16.30 |
| | DBSCAN | $0.69 \pm 0.03$ | $0.77 \pm 0.02$ | 174.63 | $0.38 \pm 0.1$ | $0.66 \pm 0.04$ | 8,918.36 |
| | DEC | $0.37 \pm 0.09$ | $0.47 \pm 0.07$ | 342.42 | $0.64 \pm 0.12$ | $0.85 \pm 0.04$ | 725.58 |
| 10% Training set | Unsup. FARE | $0.72 \pm 0.01$ | $0.80 \pm 0.01$ | 77.33 | $0.76 \pm 0$ | $0.98 \pm 0$ | 1,801.74 |
| | CSPA | $0.5 \pm 0.04$ | $0.61 \pm 0.06$ | 176.29 | $0.36 \pm 0.11$ | $0.64 \pm 0.08$ | 2,013.77 |
| | HGPA | $0.57 \pm 0.03$ | $0.69 \pm 0.05$ | 90.1 | $0.4 \pm 0.09$ | $0.79 \pm 0.06$ | 1,804.82 |

only keep $1\%$ of the labeled training samples in each class and mark the remaining samples as "unlabeled". With this setup, the training set only has in total $n - n_g + 1$ classes, and each class has $1\%$ data labeled.

We vary $n_g$ to examine its influence on the system performance. For each $n_g$, we also randomly sample different classes to merge into the union class for 10 times and construct 10 training sets. In total, we have $|n_g| \times 10$ training sets. We then run FARE and baselines on each training set and calculate the mean and standard deviation for the testing AMI and $K$. Similar to before, the accuracy metric is also reported (for selective settings) in Appendix-C.

Note that here we only consider the setting where the chosen classes are merged into *one* union label. In Appendix-E, we have included additional experimental results for *multiple* union labels. The overall conclusion is consistent, and thus the results for multiple union labels are omitted here for brevity.

**Experiment IV: Algorithm Sensitivity.** Finally, we examine the sensitivity of FARE and Unsup.FARE to the number of unsupervised neighborhood models $M' = M - 1$. For FARE, we fix $n_c = \lfloor n/2 \rfloor$ or $n_g = \lfloor n/2 \rfloor$ and labeled data ratio as $1\%$. We randomly select $M'$ =10, 20, 50, and 100 from the pool of 150 neighborhood models (used in Experiment I–III) to generate the ensemble. We run FARE and Unsup.FARE 10 times for each $M'$ and record the mean AMIs.

We also tested the algorithm sensitivity to other factors such as *the ratio of available labels*, *the output dimension of the transformation network* $q$, and *the number of true classes in a dataset*. By default, the ratio of available labels is $1\%$ and $q = 32$. We experimentally tested different ratio and $q$, and found the algorithm performance was not sensitive to neither factors. In addition, our security datasets only have up to 6 and 9 classes respectively. So we further tested FARE on an image dataset with more classes (i.e., 43), and confirmed that FARE still performed well. Due to space limit, we present the detailed results in Appendix-G and Appendix-H.

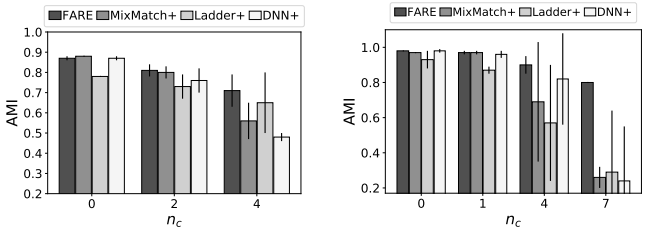
### C. Experiment Results

**FARE vs. Base Clustering Algorithms.** Table I shows the AMI and accuracy of FARE (both supervised and unsupervised versions) and other individual clustering algorithms. First, we observe that the mean AMI and accuracy of the unsupervised FARE (*i.e.*, Unsup. FARE) are consistently higher than all other clustering methods on both datasets. The performance of individual clustering methods varies on different datasets. This validates our hypothesis that existing clustering methods have different assumptions on the data distribution and work well only when the data distribution matches the assumptions. With the ensemble of multiple unsupervised models, Unsup. FARE performs consistently better. Note that Unsup. FARE has lower standard deviations, indicating its results are more stable across different training rounds. In addition, Unsup. FARE significantly outperforms K-means. This means, if we apply K-means without the input transformation network, the performance will suffer. Note that the DEC algorithm uses an auto-encoder to transform the inputs. The higher AMI of unsupervised FARE over DEC confirms the advantage of our data transformation function over the state-of-art auto-encoder.

Supervised FARE is performing better than unsupervised FARE. For example, on the network intrusion dataset, the AMI is boosted from 0.68 to 0.98. Recall that in this experiment, supervised FARE only takes $1\%$ of the labels. This confirms the benefits of combining supervised learning (even with limited labels) and unsupervised results.

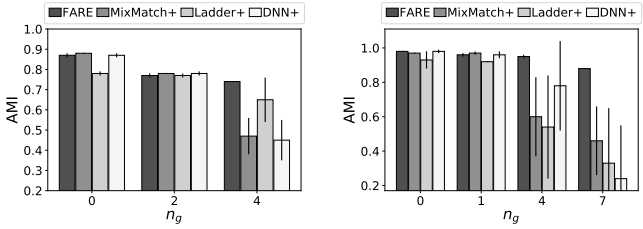
**FARE vs. Existing Clustering Ensemble Methods.** As shown in Table I, FARE and Unsup. FARE both outperform existing clustering ensemble methods CSPA and HGPA in terms of AMI and Accuracy. There are two possible reasons. First, FARE aggregates clustering ensembles with the given labels, and the labels bring in performance gains. Second, CSPA and HGPA struggle under high-dimensional input space. In comparison, FARE's input transformation network projects the inputs into a lower-dimensional space with well-defined distance, which makes clustering more effective.

**Computational Overhead.** Table I also shows the training time for each algorithm. We observe that both FARE and Unsup.FARE adds only a small fraction of the runtime on top of the existing clustering algorithms. Since the different neighborhood models are independent, we can parallelize their clustering process. As a result, the performance bottleneck is introduced by the slowest clustering algorithm in the ensemble. In our case, the slowest algorithm on the malware dataset is DEC, and the slowest algorithm on the intrusion dataset is DBSCAN. As shown in Table I, the added overhead by FARE and Unsup.FARE is considerably small, while the gains on AMIs are significant, which is a worthy trade-off. Since all of the base clustering algorithms are widely used in both academia and industry as benchmark clustering methods, we argue that the computational cost introduced by our design does not jeopardize its usage in practice. Later in Section §VI, we have further discussions on the computational cost. Table I also shows Unsup.FARE are faster than CSPA and HGPA on $10\%$ of the training dataset. This is because our method avoids the expensive cluster alignment step in CSPA and HGPA. Instead, we use pair-wise relationships to fuse the




(a) Malware categorization. (b) Intrusion detection.

Fig. 2. The performances of FARE and the baselines in the **missing classes** settings. We show the mean AMI and the standard deviation. $n_c$ is the number of missing classes.




(a) Malware categorization. (b) Intrusion detection.

Fig. 3. The performances of FARE and baselines in the **coarse-grained label** setting. We show mean AMI and standard deviation. $n_g$ is the number of classes in the union class.

base clustering results. In addition, we support batch-learning, which further reduces the memory and computational cost.

**Performance under the Missing Class Setting.** As shown in Figure 2, when there is no missing class in the training data (*i.e.,* $n_c = 0$), the performance of all the systems are fairly comparable on both datasets. Recall that the training dataset only has $1\%$ of labeled samples. With limited labels, the supervised learning method do not outperform the semi-supervised methods. Then as the number of missing classes $n_c$ increases, the baseline methods start to exhibit inconsistent and degraded performances. For example, when $n_c = 4$ in Figure 2(a), and $n_c = 7$ in Figure 2(b), it means 4 out of 6 classes are missing in the malware datasets, and 7 out of 9 classes are missing in the network intrusion dataset. The average reduction of baseline performances is around $50.5\%$. The worst AMI is even lower than $0.25$. There are two possible reasons. First, existing algorithms are highly dependent on the assumption that the labeled classes are complete (*i.e.,* at least a few labeled samples are expected to be available in each class). When this assumption is no longer held in the training data, their performances suffer. Another explanation is that existing methods cannot correctly recover the data manifolds of the unlabeled classes in the training set [24]. Without such information, they cannot make correct decisions on testing samples from these unlabeled classes. As $n_c$ increases, we notice that the standard deviations of AMIs also increase for baselines. This indicates that the choices of the missing classes also have an impact on the baseline methods' performance.

In comparison, FARE demonstrates a much higher AMI across all the settings. As shown in Figure 2, the number of missing classes only have a small impact on FARE. For example, in Figure 2(b), the AMI of FARE only decreases from $0.97$ to $0.83$ when $n_c$ increase from $0$ to $7$. On average, FARE has a reduction of $15.4\%$ of AMI across the two datasets. The results confirm the benefits of using the ensemble of unsupervised learning results when the given labels have missing classes (*i.e.*, low-quality labels). The ensemble component of FARE helps to extract useful information (*i.e.,* data manifolds) of the missing classes. The standard deviations of FARE are also consistently lower than those of the baseline methods. This again confirms the benefit of the ensemble in reducing FARE's sensitivity to the choice of missing classes.

Table II (the left half) shows the final number of classes identified by FARE and other baselines. The ground-truth number of classes is 6 and 9 for malware and intrusion datasets, respectively. As we increase the number of missing classes $n_c$, we can see that the baseline algorithms are more likely to underestimate the true number of classes on both datasets. In contrast, FARE successfully recovers the true number of classes in the malware dataset regardless of the severity of missing classes. For the network intrusion dataset, while not being able to identify the true number of classes, FARE has a lower estimation error. It should be noted that even when $n_c = 0$, none of the baseline methods can correctly recover the true number classes for the intrusion dataset. We suspect this is due to the class imbalance issue. In the intrusion dataset, the top three classes take $98.5\%$ of samples, which makes it difficult for the clustering methods to identify the minor classes. Interestingly, as the number of missing classes increases, FARE is approaching the true number of classes (possibly because the given labels become less influential). Together with the results in Figure 2, we conclude that without missing classes, FARE is comparable to the supervised and semi-supervised baselines. When there are missing classes in the given labels, FARE demonstrates significant advantages.

**Performance under the Coarse-grained Label Setting.** Figure 3 shows the performance of each method under the coarse-grained label settings. We observe that the performance of baseline methods reduces dramatically as $n_g$ increases (*i.e.,* lower AMI and higher standard deviation). This means existing methods lack the capability in dealing with the coarse-grained labels. On the contrary, coarse-grained labels only have a minor impact on FARE. The average AMI reduction is only $8.2\%$ across all the settings. The standard deviations are low for FARE, indicating that FARE is robust to the choices of classes in the union label.

Table II (the right half) presents the number of final classes identified by each method under the coarse-grained label settings. For the malware dataset, we have the same observation as before: FARE correctly recovers the true number of classes regardless of the degree of coarse-grained labels while other baseline algorithms cannot. For the intrusion dataset, however, the observation is quite different. On the one hand, neither FARE nor the baseline methods can estimate the true number of classes correctly. On the other hand, unlike the baseline methods that occasionally overestimate the true number of classes, FARE consistently underestimates it. For example, when $n_g = 7$, the number classes estimated by FARE is only $4$. We speculate that this is caused by the compound effect introduced by coarse-grained labels and the extreme class imbalance. Recall that in the intrusion dataset, the top 3 classes count for 98.5% of the samples. The coarse-grained label

TABLE II. THE NUMBER OF CLUSTERS $K$ DISCOVERED BY FARE AND THE BASELINE ALGORITHMS UNDER DIFFERENT SETTINGS. $N$ REPRESENTS THE GROUND-TRUTH NUMBER OF CLASSES IN EACH DATASET.

| Methods | Num. of missing classes ($n_c$) | | | | | | | Num. of mistaken grouped classes ($n_g$) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Malware ($N=6$) | | | Intrusion ($N=9$) | | | | Malware ($N=6$) | | Intrusion ($N=9$) | | |
| | 0 | 2 | 4 | 0 | 1 | 4 | 7 | 2 | 4 | 1 | 4 | 7 |
| FARE | $\mathbf{6 \pm 0}$ | $\mathbf{6 \pm 0}$ | $\mathbf{6 \pm 0}$ | $\mathbf{6 \pm 0}$ | $\mathbf{6 \pm 0}$ | $\mathbf{8 \pm 1.25}$ | $\mathbf{10 \pm 1.89}$ | $\mathbf{6 \pm 0}$ | $\mathbf{6 \pm 0}$ | $\mathbf{5 \pm 0}$ | $5 \pm 0.47$ | $4 \pm 0$ |
| MixMatch+ | $\mathbf{6 \pm 0}$ | $4 \pm 0$ | $4 \pm 0$ | $5 \pm 0.47$ | $4 \pm 0$ | $6 \pm 1.69$ | $5 \pm 1.41$ | $5 \pm 0$ | $4 \pm 0$ | $5 \pm 0.47$ | $5 \pm 0.47$ | $\mathbf{7 \pm 1.25}$ |
| Ladder+ | $4 \pm 0$ | $4 \pm 0$ | $5 \pm 0$ | $5 \pm 0$ | $6 \pm 2.44$ | $6 \pm 0$ | $7 \pm 2.36$ | $4 \pm 0$ | $5 \pm 0.47$ | $5 \pm 0.47$ | $7 \pm 2.05$ | $16 \pm 3.77$ |
| DNN+ | $\mathbf{6 \pm 0}$ | $5 \pm 0.82$ | $4 \pm 0$ | $5 \pm 0.92$ | $\mathbf{6 \pm 0}$ | $6 \pm 0$ | $4 \pm 0$ | $5 \pm 0$ | $5 \pm 5.44$ | $5 \pm 0.47$ | $\mathbf{6 \pm 1.7}$ | $15 \pm 2.49$ |

TABLE III. MEAN AMIS AND STANDARD DEVIATIONS OBTAINED BY VARYING $M'$ IN FARE AND Unsup.FARE.

| # Neighb. Models $M'$ | | 10 | 20 | 50 | 100 | All (150) | 10 | 20 | 50 | 100 | All (150) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Dataset | Malware | | | | | Intrusion | | | | |
| FARE | $n_c = \lfloor n/2 \rfloor$, 1% labels | $0.68 \pm 0.09$ | $0.73 \pm 0.01$ | $0.76 \pm 0.01$ | $0.75 \pm 0.01$ | $0.75 \pm 0$ | $0.89 \pm 0.10$ | $0.88 \pm 0.11$ | $0.88 \pm 0.08$ | $0.89 \pm 0.05$ | $0.89 \pm 0.05$ |
| | $n_g = \lfloor n/2 \rfloor$, 1% labels | $0.74 \pm 0.04$ | $0.75 \pm 0$ | $0.74 \pm 0$ | $0.74 \pm 0$ | $0.74 \pm 0$ | $0.90 \pm 0.05$ | $0.90 \pm 0.05$ | $0.90 \pm 0.05$ | $0.90 \pm 0.05$ | $0.90 \pm 0.05$ |
| Unsup. FARE | Dataset | Malware | | | | | Intrusion | | | | |
| | labels are not used | $0.60 \pm 0.08$ | $0.71 \pm 0.01$ | $0.74 \pm 0.01$ | $0.74 \pm 0.01$ | $0.74 \pm 0$ | $0.83 \pm 0.05$ | $0.79 \pm 0.02$ | $0.79 \pm 0.03$ | $0.78 \pm 0.00$ | $0.78 \pm 0.00$ |

can lead to misguided data manifold, and highly imbalanced classes undermine the effectiveness of the clustering ensemble. When both issues are presented, FARE is less effective. In summary, for the general coarse-grained label setting, FARE significantly outperforms the baseline methods in terms of clustering quality and estimating the number of classes. When the classes are extremely imbalanced, FARE and other baselines are less effective in estimating the true number of classes.

**Sensitivity to the Number of Neighborhood Models.** Table III shows the results of FARE and Unsup.FARE with different number of neighborhood models $M$ under the missing class/coarse-grained label settings. We vary the $M' = M - 1$ neighborhood models from clustering algorithms. We can observe that the performance of FARE is robust with respect to the number of neighborhood models. As we add more neighborhood models to the ensemble, the mean AMI is increasing and the standard deviation is decreasing, but only by a small margin. This means with 20 or even 10 neighborhood models, the performance of FARE is already good. Similar to FARE, Unsup.FARE is also not sensitive to $M'$.

## V. REAL-WORLD TEST: FRAUD DETECTION

Following the controlled experiments, we next describe our experience of the initial deployment and testing of FARE in collaboration with a real-world online service JD.com. Company JD.com is a large e-commerce service with hundreds of millions of active users. We work together to apply FARE to identify the fine-grained classes of fraudulent accounts, especially the previously-unknown types of fraud. As the initial testing effort, we apply FARE on an internal dataset of 200,000 active users. Below, we describe our testing methodology, and key observations and discoveries.

**Dataset from Company JD.com.** The dataset contains 200,000 active users randomly sampled from the e-commerce site database. Each user is represented as a 264-dimensional feature vector. The feature vector is encoded using their internal feature engineering method. As the specific details of the feature engineering process are not revealed to us (which is confidential information), we only provide a high-level description here. The features are extracted from three different types of information: 1) product information (*e.g.,* product brand and product category), 2) shipping information (*e.g.,* shipping address and carrier information), and 3) purchasing information (*e.g.,* price, amount, discounts, and time).

The dataset has a very small portion of labels, including $0.5\%$ of confirmed fraudulent users, and $0.1\%$ of trustworthy users. The remaining 99.4% of users are unlabelled. First, $0.5\%$ of the accounts are labeled as "fraudulent". This label is based on JD.com's customer service department — they have received complaints on these $0.5\%$ accounts who were conducting fraudulent activities in the last two months (with further confirmations from the security team). About $0.1\%$ of remaining users accounts are labeled as "trusted" since they are associated with company JD.com's enterprise partners or VIP customers. This dataset represents the common challenges we described before: only a small portion of labels are available and the labels are likely to be coarse-grained and biased.

TABLE IV. GROUP-A REPRESENTS THE FRAUDULENT ACCOUNTS IDENTIFIED BY FARE; GROUP-B REPRESENTS THE CONFIRMED LEGITIMATE USERS. WE RECORD THE LOGIN ATTEMPT RATE (LAR) AND THE AUTHENTICATION PASS RATE (APR) FOR BOTH GROUPS.

| Group | 1-day (LAR, APR) | 1-week (LAR, APR) | 1-month (LAR, APR) |
|---|---|---|---|
| A: FARE-detected | $(\mathbf{20.9\%}, \mathbf{0.0\%})$ | $(\mathbf{25.3\%}, \mathbf{0.0\%})$ | $(\mathbf{39.3\%}, \mathbf{0.0\%})$ |
| B: Confirmed-legit. | $(22.1\%, 100\%)$ | $(27.9\%, 100\%)$ | $(30.9\%, 100\%)$ |

**A/B Test Experiments.** Using this dataset, our goal is to pinpoint the unlabeled users who also conducted fraudulent activities in the past two months but have not yet been complained by online retailers through the customer service.

To validate whether FARE can truly identify those accounts, we design an A/B test experiment for two groups of users. Group-A is the fraudulent accounts that FARE identified from the unlabeled user sets, and Group-B is the labeled trustworthy users. For both groups of users, we revoke their sign-in cookies, and force them to re-enter their passwords, and use their registered phone numbers to perform two-factor authentication through SMS code. Then, we keep monitoring the login activities of both groups of users for one month after the forced re-login. During the monitoring period, we record the login attempt rate (LAR) as well as the authentication pass rate (APR). Here, the LAR indicates the percentage of the users who have correctly entered their passwords when performing log-in. The APR specifies, among users who attempted the log-in, the percentage of sign-in sessions with the correct two-factor authentication code.

The rationale behind the A/B experiment is that attackers behind the fraud campaigns usually purchase a large corpus of fake accounts from third-party vendors to conduct malicious activities. When the third-party vendors create these fake accounts for sale, they needed to register the accounts by using the phone numbers under their control. When selling these accounts, the third-party vendors would provide the account names and passwords so that the buyers can log in

to these accounts. However, if JD.com forces a two-factor authentication after the account delivery, the buyers would not be able to receive the SMS code tied to each of the accounts, and thus cannot use these accounts to continue their campaigns to snatch coupons, promote illegitimate products, or write fake reviews. It should be noticed that while re-authentication is a powerful tool, it cannot be excessively used. When blindly triggering re-authentication to all users, it could jeopardize normal users' experience and significantly increase the burden of the customer service department. This is because legitimate users may sometimes change their phone numbers and forgot to update their online profiles. Even if such normal users only take a small portion, considering the hundreds of millions of active users in JD.com, the absolute number is still very large. They can easily overwhelm the customer service if the re-authentication is triggered at the same time.

**A/B Experiment Results.** While our controlled experiments in Section IV-C have shown FARE's good performance, we still want to stay conservative in this initial real-world testing. Specifically, we want to suppress the potential false positives of FARE since false positives disrupt the customer service's daily operations. As a result, JD.com permitted us to initialize SMS re-authentication for an entire cluster of users only if this FARE-identified cluster contains at least $5\%$ of the already-confirmed fraudulent accounts. While this approach may significantly under-report the fraudulent accounts identified by FARE, we believe it is the right trade-off for the initial testing. For the other clusters (*e.g.,* those that contain fraudulent labels but do not meet the $5\%$ threshold), they are still valuable for further analysis, but are excluded from the A/B test.

Under this guideline, FARE revoked 2,000 unlabeled sign-in sessions and initialized the corresponding re-authentication through SMS. In Table IV, we show the LAR and APR of each group across three different time windows – one day, one week, and one month. We can observe that for the confirmed trusted users (Group-B), the return sign-in rate across a month is $30\%$ with a $100\%$ of success rate for passing the SMS re-authentication. On the contrary, for the FARE-detected fraudulent users (Group-A), the return rate is about $10\%$ higher, but the success rate of SMS re-authentication is $0\%$. This implies that the users FARE detected are highly likely to be the fake accounts associated with fraudulent activities.

**Manual Analysis and Observations.** In addition to our A/B experiment, we also devote efforts to manual examinations. We focus on accounts that failed the SMS re-authentication and analyzed their history logs. While we are not allowed to provide the precise numbers and statistics of the discovered fraudulent activities, we want to provide qualitative results regarding our key findings.

*First*, for many clusters of the newly identified fraudulent accounts, accounts in each cluster usually have the same login time and come from the same or similar sets of IP addresses. This implies the user accounts in the same cluster are likely conducted by a single entity using automated programs. *Second*, we find that certain groups of fraudulent users would heavily apply coupons on their purchases. For almost all of their purchased items, they applied an abnormal amount of coupons to significantly reduce the purchasing price. More importantly, users in the same group even share the same physical shipping addresses. These clusters are likely to represent the organized efforts to (automatically) collect coupons, purchase products in bulk, and then resell them with higher prices. *Third*, we also discover fraudulent clusters that regularly buy products from certain retailers and leave positive reviews. More importantly, after leaving the positive reviews, these accounts then file product returns and get a refund. We suspect these accounts are colluding with the retailers for promoting their products. *Fourth*, by analyzing the historical activities of these fraudulent accounts, we were surprised to discover that many products have mistagged prices. For example, some products owned by the e-commerce site were mistagged with a low price for weeks without being noticed by the product team. The fraudulent accounts have been exploiting these mistagged prices to subside their purchases. These mistagged prices are previously unknown to JD.com. JD.com has started to actions to perform systematic detection of mistagged prices.

## VI. Discussion

**Post-clustering Processing.** The goal of FARE is to categorize the input dataset into fine-grained clusters, and help the analysts to derive high-quality labels. After FARE is applied, the post-processing is to either align the obtained clusters with the known classes in the "given labels" or assign them with new labels. Two strategies can be applied to accomplish these tasks. The first strategy is to manually analyze samples in a given cluster to assign meaningful labels. For example, the analysts can identify a small number of representative samples (*e.g.*, based on centrality) in a given cluster for in-depth analysis. We argue that, by producing high-quality clusters, FARE saves analysts' time who only need to investigate a smaller number of representative samples. To further save manual efforts, analysts might take the second strategy, which is to align clusters with the given labels. For each cluster that contains given labels, we can compute a matching score (*i.e.,* $\frac{\text{\# of matched samples}}{\text{\# of total samples in the cluster}}$) and find the label with the highest matching score. The analyst can set a cut-off threshold (*e.g.,* $0.9$): if the highest matching score is above this threshold, the cluster stays with the existing label. Appendix-F shows a running example. The second strategy, while efficient, should be applied carefully (*e.g.*, to well-known classes only) since the given labels are not entirely trustworthy.

**Computational Complexity.** In Section §IV, we show that FARE only introduces a small computational overhead on top of the clustering algorithms. Here, we compare the asymptotic complexity of FARE and with those of the (semi-)supervised baselines. Specifically, the computational complexity of FARE is $O(\max\{I_d B_d |\theta_d| K,\ pN^2, I_i M B_i^2 |\theta_i|\})$, where $N$ is the number of samples. $I_d$, $B_d$, and $|\theta_d|$ represents the number of training iteration, batch size, and model parameters of DEC. Similar, $I_i$, $B_i$, and $|\theta_i|$ represents the number of training iteration, batch size, and parameters of input transformation model (see Appendix-D for the derivation).

The computational cost of the (semi)-supervised baselines are: MixMatch – $O(IB^2 A\,|\theta|)$, Ladder and DNN – $O(IB\,|\theta|)$, where $A$ is the number of data augmentation rounds in MixMatch. When $pN^2 < \max\{I_d B_d |\theta_d| K, I_i M B_i^2 |\theta_i|\}$, similar to MixMatch, the complexity of FARE is also quadratic to the batch size $B$. A practical example is the malware dataset in Section §IV, where the average run time of MixMatch (*i.e.,* $400.33s$) is similar to FARE (*i.e.,* $434.05s$). For a very

large scale dataset with an ultra-high dimensionality, FARE may be slower than semi-supervised learning methods due to the high cost of DBSCAN. However, recent research has proposed accelerate DBSCAN through parallel computing [29] or GPU [25]. These strategies can also be applied to further accelerate FARE for very large-scale datasets. With the above analysis, we can conclude that the computational cost of FARE is acceptable. Our real-world deployment in Section §V also confirms that practicality of running FARE in production.

**Hyperparameters.** FARE has the following hyper-parameters: the number of neighborhood model $M$, the hyper-parameter inherited from contrastive learning (the output dimension of input transformation net $q$, the distance radius $\alpha$ and the regularization coefficients $\lambda$), the hyper-parameters inherited from base clustering algorithms, and the hyper-parameters introduced by our design ($K$ and $p_1$). As discussed in Section §III-D, we set $\lambda$ to a small value and select the $K$ and $p_1$ based on the AMIs computed on a validation set. For $M$, as shown in Section §IV, FARE can achieve a good performance with merely 10 unsupervised neighborhood models. Appendix-G shows that FARE is also robust to the subtly changes in the distance radius $\alpha$ and the output dimension $q$. For the hyper-parameters of clustering algorithms, existing works have provided suggested default setting [11].

**Online Setup.** While primarily designed for offline analysis, FARE can also be used in an online fashion. As is elaborated in Section §III-A, FARE processes a dataset with three steps. After the first two steps, FARE could learn a transformation function. Using this function to map each data sample into a low-dimensional space, FARE then employs K-means to assign data to the corresponding category. This clustering step could be done incrementally. Therefore, it introduces only lightweight computation and offers the possibility of performing online clustering. In our current design, the first two steps are more computationally intensive than the third step. Therefore, it is challenging to update the transformation function in an online fashion. However, this does not hinder the online usage of FARE.

After learning a transformation function, even without frequently updating it, FARE could still perform clustering accurately. Take our current deployment in the real-world online service J as an example. In order to capture the distribution/covariate shift [58], [8], we retrain and update our transformation function weekly. We observe that this setup does not jeopardize FARE's efficacy, which implies the feasibility of FARE's online usage. However, we admit that the retraining cycle could vary for different applications because of the variation in the data dynamics. This work will leave the in-depth online usage exploration as part of future work.

**Adversarial Attacks.** As an machine learning algorithm, FARE could be vulnerable to adversarial manipulations such as poisoning attacks and adversarial evasion attacks. Researchers have explored data poisoning attacks on unsupervised learning algorithms. To the best of our knowledge, existing attacks target a specific learning algorithm (*e.g.,* hierarchical clustering [9] or graph-based clustering [16]), which are not directly applicable to our algorithm yet. For adversarial evasion, attackers may leverage transferability and use adversarial examples generated from a supervised classifier to attack our method. However, transferability relies on the assumption that supervised classifiers on the same problem/data share similar decision boundaries. In our case, FARE is trained with both labeled and unlabeled data and uses the ensemble of multiple algorithms, which may produce different cluster boundaries (compared with those of the classifier). In addition, generating realizable adversarial malware example is a challenging task. It requires the adversarial example to be an executable binary that preserves the original malicious functions. We leave the evaluation of FARE's adversarial robustness to future research.

**Corrupted Labels.** In our current threat model, we assume the provided labels are either missing or coarse-grained. In Appendix-I, we further tested FARE under corrupted training labels (i.e., samples mislabeled to the wrong classes). The results show that FARE's performance slightly drops as more labels are corrupted. In practice, there are potential ways to mitigate the negative effect of corrupted labels. For example, we could measure the discrepancy between the given labels and the unsupervised clustering results based on the neighborhood relationship table. If the difference is unusually large, defenders should further inspect the labels or conservatively apply the unsupervised version of FARE. We defer the implementation and evaluation of this idea to future work.

**Limitations and Future Works.** Our work has a few limitations. First, we mainly choose clustering algorithms that are already widely used in the security domain. Some of the clustering algorithms indeed have drawbacks. For example, we show that DBSCAN, in certain settings, becomes the computational bottleneck for FARE. As future work, we want to explore alternative clustering algorithms that can further accelerate the system. Second, our system could still under-estimate the number of true classes when the data is extremely imbalanced. Future work may investigate other solutions such as down-sampling large clusters and then run FARE alternatively. Third, we apply the same weight for all the unsupervised models in the ensemble to simplify the parameter tuning. It is possible to further improve FARE's performance by designing a fine-tuning strategy for the weights of these unsupervised models. Fourth, we mainly evaluate the impact of missing classes and coarse-grained labels in *separate experiments*. Due to the space limit, we have added a brief experiment (in Appendix-E) where both labeling issues are present in the same training dataset. Finally, we tested and demonstrated FARE's effectiveness on three security applications (and one image classification task in Appendix-H). As part of future work, we will validate our system's generalizability to other (non-)security applications.

## VII. Related work

**Supervised Learning Methods.** Traditional supervised learning methods such as Support Vector Machines (SVMs) and random forests have long been used to classify malware [43], [64], [5], [22], [50], detect network intrusions [48], [39], [18], [63], and identify fraudulent accounts [7], [54], [72]. Recently, deep learning models have been used for similar purposes [1], [32], [44]. As is shown in Section §IV, the effectiveness of these methods decreases significantly under low-quality labels.

**Semi-supervised Learning Methods.** SSL can be trained with partially labeled data. They are usually composed of an unsupervised component and a supervised component. The unsupervised component projects an input sample $\boldsymbol{x}$ to a

hidden representation $h$ and the supervised component predicts its label $y$ from the hidden representation $h$ [24], [3]. Related semi-supervised systems include Ladder [59], MixMatch [6], and ODDS [31]. ODDS uses data augmentation techniques to train a bot detector with limited labels, but it primarily works in a binary classification setting (and thus does not meet our need for attack categorization). Ladder [59] is applied for network intrusion detection, and MixMatch [6] has been tested mainly on image datasets. As is shown in Section §IV, low-quality labels in the training data would significantly jeopardize the performance of semi-supervised methods.

**Unsupervised Learning Methods.** Clustering algorithms such as K-means [28] and DBSCAN [20] have been applied to identity and group malware samples [76], [12], [40], network intrusion events [49], [15], and fraudulent accounts [57], [62]. However, these methods are not good at handling high-dimensional inputs due to the "curse of dimensionality" [83]. To overcome this challenge, more advanced techniques such as DEC [78], [53] and DAGMM [84] use deep neural networks to learn a desired low-dimensional representation of original inputs before applying the clustering method. To improve the stability of clustering, clustering ensemble methods are proposed, which combine the clustering outputs from multiple (weak) base models using some consensus functions. For example, `CSPA` and `HGPA` [67], [77] are two popular clustering ensemble methods. CSPA utilizes the probability of two data points co-locating in the same cluster as the consensus measure; HGPA represents the outputs from the base clusters as a hyper-graph and converts the clustering task into a hyper-graph partitioning problem. Without the guidance of labels, unsupervised learning methods are usually outperformed by semi-supervised learning methods.

**Zero-shot Learning Methods.** ZSL and GZSL have been recently used in network intrusion detection tasks [60]. These methods transfer the knowledge learned from one task to a second task [24] and can be used to classify previously unseen classes in the testing set. This is done by learning a feature mapping function based on the well-labeled data of the first task (training set), and transform the inputs of the second task (testing set) through the mapping function. As is discussed in Section §II-C, due to the need of rich training labels and "side information" to construct the feature mapping, ZSL/GZSL methods are not suitable for our problem.

## VIII. Conclusion

This paper introduces `FARE`, a new method to derive accurate and robust clustering results for security applications under low-quality label data. By computing an ensemble of "given labels" and multiple supervised learning results, we use a transformation network to transform input samples into a low-dimensional space for fine-grained clustering. We evaluate `FARE` with both controlled experiments (for malware classification and network intrusion detection) and real-world deployment and testing (for fraudulent account detection). We demonstrate the benefits of `FARE` over existing semi-supervised methods and its usefulness in practice.

## Acknowledgment

We thank anonymous reviewers for their constructive comments and suggestions. This work was supported in part by NSF grants CNS-2030521 and CNS-1717028, and Amazon Research Award.

## References

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proc. of NDSS*, 2014.

[2] A. Authors, "Malware dataset," 2019, https://1drv.ms/u/s!AgWFTItaV0h8jLAcx9dRzqIlMvUWtg?e=O3Gj5q.

[3] S. Becker and G. E. Hinton, "Self-organizing neural network that discovers surfaces in random-dot stereograms," *Nature*, 1992.

[4] A. Bendale and T. E. Boult, "Towards open set deep networks," in *Proc. of CVPR*, 2016.

[5] K. Berlin, D. Slater, and J. Saxe, "Malicious behavior detection using windows audit logs," in *Proc. of AI & Security Workshop of CCS*, 2015.

[6] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. A. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," in *Proc. of NeurIPS*, 2019.

[7] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, "Data mining for credit card fraud: A comparative study," *Decision Support Systems*, 2011.

[8] S. Bickel, M. Brückner, and T. Scheffer, "Discriminative learning under covariate shift." *Journal of Machine Learning Research (JMLR)*, 2009.

[9] B. Biggio, S. R. Bulò, I. Pillai, M. Mura, E. Z. Mequanint, M. Pelillo, and F. Roli, "Poisoning complete-linkage hierarchical clustering," in *Proc. of SPR and SSPR*, 2014.

[10] Y. Boshmaf, D. Logothetis, G. Siganos, J. Leria, J. Lorenzo, M. Ripeanu, and K. Beznosov, "Integro: Leveraging victim prediction for robust fake account detection in osns," in *Proc. of NDSS*, 2015.

[11] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler *et al.*, "API design for machine learning software: experiences from the scikit-learn project," in *ECML-PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013.

[12] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proc. of ACM workshop on Security and privacy in smartphones and mobile devices*, 2011.

[13] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, "Aiding the detection of fake accounts in large scale social online services," in *Proc. of USENIX NSDI*, 2012.

[14] Q. Cao, X. Yang, J. Yu, and C. Palow, "Uncovering large groups of active malicious accounts in online social networks," in *Proc. of CCS*, 2014.

[15] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, 2012.

[16] Y. Chen, Y. Nadji, A. Kountouras, F. Monrose, R. Perdisci, M. Antonakakis, and N. Vasiloglou, "Practical attacks against graph-based clustering," in *Proc. of CCS*, 2017.

[17] Y. Chen, S. Wang, D. She, and S. Jana, "On training robust pdf malware classifiers," in *Proc. of USENIX Security*, 2020.

[18] R. Chitrakar and C. Huang, "Selection of candidate support vectors in incremental svm for network intrusion detection," *computers & security*, 2014.

[19] Y. Duan, X. Li, J. Wang, and H. Yin, "Deepbindiff: Learning program-wide code representations for binary diffing," in *Proc. of NDSS*, 2018.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. of KDD*, 1996.

[21] FireEye, "M-trends reports: Insights into today's breaches and cyber attacks," 2020, https://content.fireeye.com/m-trends/rpt-m-trends-2020.

[22] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, 2014.

[23] N. Z. Gong, M. Frank, and P. Mittal, "Sybilbelief: A semi-supervised learning approach for structure-based sybil detection," *IEEE Transactions on Information Forensics and Security*, 2014.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[25] M. Gowanlock, C. M. Rude, D. M. Blair *et al.*, "Clustering throughput optimization on the gpu," in *Proc. of IPDPS*, 2017.

[26] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proc. of CVPR*, 2006.

[27] B. Harkness, "Dealing with fraud and identity theft," 2019, https://www.creditcardinsider.com/learn/fraud-identity-theft/.

[28] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C*, 1979.

[29] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, "Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce," in *Proc. of ICPADS*, 2011.

[30] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *Proc. of ICLR*, 2017.

[31] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, "Throwing darts in the dark? detecting bots with limited data using neural data augmentation," in *Proc. of IEEE S&P*, 2020.

[32] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. of BIONETICS*, 2016.

[33] J. Jia, B. Wang, and N. Z. Gong, "Random walk based fake account detection in online social networks," in *Proc. of DSN*, 2017.

[34] H. Jiang, B. Kim, M. Guan, and M. Gupta, "To trust or not to trust a classifier," in *Proc. of NeurIPS*, 2018.

[35] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *Proc. of ICML*, 2018.

[36] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proc. of USENIX Security*, 2017.

[37] KDDCup, "Network intrusion data," 1999, https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data.

[38] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, 2019.

[39] D. S. Kim, H.-N. Nguyen, and J. S. Park, "Genetic algorithm to improve svm based network intrusion detection system," in *Proc. of AINA*, 2005.

[40] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in computer virology*, 2011.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[42] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, 2004.

[43] M. Kruczkowski and E. N. Szynkiewicz, "Support vector machine for malware analysis and classification," in *Proc. of WIIAT*, 2014.

[44] S. Kudugunta and E. Ferrara, "Deep neural networks for bot detection," *Information Sciences*, 2018.

[45] P. Laskov and N. Šrndić, "Static detection of malicious javascript-bearing pdf documents," in *Proc. of ACSAC*, 2011.

[46] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, 2015.

[47] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Proc. of IEEE S&P*, 1999.

[48] H. Li, X.-H. Guan, X. Zan, and C.-Z. HAN, "Network intrusion detection based on support vector machine," *Journal of Computer Research and Development*, 2003.

[49] Z. Li, Y. Li, and L. Xu, "Anomaly intrusion detection method based on k-means clustering algorithm with particle swarm optimization," in *Proc. of ICM*, 2011.

[50] N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, "Machine learning aided android malware classification," *Computers & Electrical Engineering*, 2017.

[51] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *Proc. of NDSS*, 2018.

[52] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[53] C. K. Ng, F. Jiang, L. Y. Zhang, and W. Zhou, "Static malware clustering using enhanced deep embedding method," *Concurrency and Computation: Practice and Experience*, 2019.

[54] X. Niu, L. Wang, and X. Yang, "A comparison study of credit card fraud detection: Supervised versus unsupervised," *arXiv preprint arXiv:1904.10604*, 2019.

[55] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration," *PyTorch*, 2017.

[56] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. of USENIX Security*, 2019.

[57] C. Phua, V. Lee, K. Smith, and R. Gayler, "A comprehensive survey of data mining-based fraud detection research," *arXiv preprint arXiv:1009.6119*, 2010.

[58] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. The MIT Press, 2009.

[59] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, "Semi-supervised learning with ladder networks," in *Proc. of NeurIPS*, 2015.

[60] J. Rivero, B. Ribeiro, N. Chen, and F. S. Leite, "A grassmannian approach to zero-shot learning for network intrusion detection," in *Proc. of ICONIP*, 2017.

[61] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, 1987.

[62] A. S. Sabau, "Survey of clustering based financial fraud detection research," *Informatica Economica*, 2012.

[63] T. Shon, Y. Kim, C. Lee, and J. Moon, "A machine learning framework for network anomaly detection using svm and ga," in *Proc. of IEEE SMC information assurance workshop*, 2005.

[64] C. Smutz and A. Stavrou, "Malicious pdf detection using metadata and structural features," in *Proc. of ACSAC*, 2012.

[65] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. of IEEE S&P*, 2010.

[66] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition," in *Proc. of IJCNN*, 2011.

[67] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *Journal of machine learning research (JMLR)*, 2002.

[68] G. Stringhini, P. Mourlanne, G. Jacob, M. Egele, C. Kruegel, and G. Vigna, "Evilcohort: Detecting communities of malicious accounts on online services," in *Proc. of USENIX Security*, 2015.

[69] J. L. Suárez, S. García, and F. Herrera, "A tutorial on distance metric learning: Mathematical foundations, algorithms and software," *arXiv preprint arXiv:1812.05944*, 2018.

[70] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proc. of CISDA*, 2009.

[71] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson, "Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse," in *Proc. of USENIX Security*, 2013.

[72] O. Varol, E. Ferrara, C. A. Davis, F. Menczer, and A. Flammini, "Online human-bot interactions: Detection, estimation, and characterization," in *Proc. of Weblogs and Social Media Workshop of AAAI*, 2017.

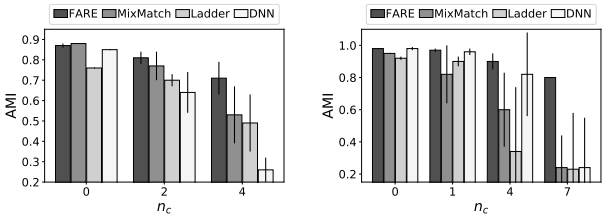
[73] P. Vibert, "The rapid evolution of the ransomware industry," 2019, https://www.cybersecurity-review.com/articles/the-rapid-evolution-of-the-ransomware-industry/.

[74] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, 2010.

[75] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao, "Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers," in *Proc. of USENIX Security*, 2014.

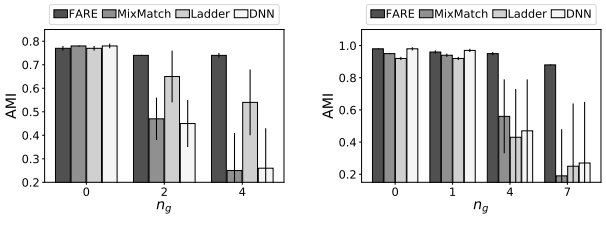
[76] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Proc. of Asia Joint Conference on Information Security*, 2012.

[77] X. Wu, T. Ma, J. Cao, Y. Tian, and A. Alabdulkarim, "A comparative study of clustering ensemble algorithms," *Computers & Electrical Engineering*, 2018.




(a) Malware categorization. (b) Intrusion detection.

Fig. 4. FARE vs. baselines in **missing classes**.




(a) Malware categorization. (b) Intrusion detection.

Fig. 5. FARE vs. baselines in **coarse-grained labels**.


[78] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proc. of ICML*, 2016.

[79] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng, "Distance metric learning with application to clustering with side-information," in *Proc. of NeurIPS*, 2003.

[80] L. Zhang, T. Xiang, and S. Gong, "Learning a deep embedding model for zero-shot learning," in *Proc. of CVPR*, 2017.

[81] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," in *Proc. of CCS*, 2014.

[82] Z. Zhu and T. Dumitraş, "Featuresmith: Automatically engineering features for malware detection by mining the security literature," in *Proc. of CCS*, 2016.

[83] A. Zimek, E. Schubert, and H.-P. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2012.

[84] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Proc. of ICLR*, 2018.


## APPENDIX-A. IMPLEMENTATION AND HYPER-PARAMETER.

We implemented FARE with the PyTorch [55] package. We implemented K-means and DBSCAN with the APIs in the scikit-learn package[11]. We adopted the hyper-parameters suggested by scikit-learn package for K-means and DBSCAN. The network architecture of DEC is the same as other DNNs, which are specificied as the following. (1) Network Structure: we used an MLP with the architecture of "input dimension-500-500-2000-32" with Tanh activation for each layer. We also adopted a weight decay with the strength of $0.01$ as the regularization. (2) Minibatch size: we created two trunks of minibatch samplers sampling from unlabeled and labeled data. Supervised DNN only uses the labeled sampler. MixMatch and Ladder utilize both samplers to ensure each batch contains both unlabeled and labeled data. The batch size of labeled and unlabeled sampler is 64 and 128 respectively. (3) Training epochs: the maximum training epoch is $1000$. We used an early-stopping mechanism: we stop when either the loss on the validation set increases in two consecutive epochs, or the loss decrease on the training set is below $0.01$ in two consecutive epochs. These DNNs were trained with the Adam optimizer, with the learning rate of $0.001$.

TABLE V. MEAN AMIS AND STANDARD DEVIATION OF FARE AND GENERALIZED ZERO-SHOT LEARNING METHODS (GZSL).

| Label Condition | $n_c = \lfloor n/2 \rfloor$, 1% labels | | $n_g = \lfloor n/2 \rfloor$, 1% labels | |
|---|---|---|---|---|
| Dataset | Malware | Intrusion | Malware | Intrusion |
| FARE | $0.75 \pm 0$ | $0.89 \pm 0.05$ | $0.74 \pm 0$ | $0.90 \pm 0.05$ |
| OSDN | $0.64 \pm 0.09$ | $0.40 \pm 0.35$ | $0.63 \pm 0.06$ | $0.63 \pm 0.39$ |
| DEM | $0.01 \pm 0.01$ | $0.00 \pm 0$ | $0.00 \pm 0$ | $0.00 \pm 0$ |

TABLE VI. DEMONSTRATION OF CONTINGENCY TABLE. COLUMN 2-4 REPRESENTS THE COUNTS OF OVERLAPPING SAMPLES BETWEEN EACH PREDICTED CLUSTER AND TRUE CLASS. COLUMN 5 AND ROW 4 SUMS UP THE SAMPLES IN PREDICTED CLUSTERS AND TRUE CLASSS.

| Predictions \ Labels | Class 1 | Class 2 | Class 3 | Sums |
|---|---|---|---|---|
| Cluster 1 | 960 | 18 | 2 | 980 |
| Cluster 2 | 20 | 0 | 0 | 20 |
| Sums | 980 | 18 | 2 | 1000 |

Regarding the hyper-parameters of FARE, first, we ran DEC, k-means, and DBSCAN 50 times with the hyper-parameters introduced above. Together with the "given labels", we constructed in total $M = 151$ neighborhood models. Then we set the distance radius $\alpha = 32$, and the regularization coefficients $\lambda = 0.01$. For the weight of the "given labels" $p_1$, we applied the selection mechanism introduced in Section §III-D and set it as $10$ in the malware dataset and $\{1, 7, 10\}$ in the network intrusion dataset ($1$ is used in $n_g/n_c = 7$, $7$ is used for $n_g/n_c = 4$ and $n_g = 1$, and the rest are $10$). Finally, for the input transformation network, we used the same set of hyper-parameters as that of the semi-supervised baselines.

## APPENDIX-B. BASELINES PERFORMANCE.

**FARE vs. Semi-supervised Baselines.** Figure 4 and 5 shows the comparison between FARE and the baselines' original implementations. The AMIs of the baselines decrease drastically as $n_c$ and $n_g$ increases. The worst AMI is even lower than $0.1$. The results indicate that, without further adaptation, none of the baselines can handle missing classes/coarse-grained labeled data. As thus, we amended all baselines in our evaluation.

**FARE vs. Generalized Zero-shot Learning.** We compared FARE with two representative GZSL methods OSDN [4] and DEM [80], under the low-quality label setups. For the missing class setting, we fixed $n_c = \lfloor n/2 \rfloor$ for each dataset and kept $1\%$ labeled training sample in each known class. We ran FARE, OSDN, and DEM on the constructed training set and compared their testing AMIs. For the coarse-grained label setting, we fixed $n_g = \lfloor n/2 \rfloor$ for each dataset and followed the same procedure. In both experiments, we ran each method $10$ times. Note that DEM requires side information (*i.e.*, shared semantic properties) about the connection between the training and testing sets. Since our datasets do not provide such side information, we set to provide *random information* to DEM. As shown in Table V, FARE significantly outperforms OSDN and DEM in both setups. This is because GZSL requires rich training samples from the known classes. However, in our setting, we only have $1\%$ of labels in known classes. We can also observe that without meaningful side information, DEM completely fails in our task. This result shows that GZSL methods are not suitable for our setup. As such, we do not consider them as the baselines in our experiments.

TABLE VII. ACCURACY COMPARISON ON SELECTED MISSING CLASSES AND COARSE-GRAINED LABEL SETTINGS.

| Dataset | MALWARE | | Network Intrusion | |
|---|---|---|---|---|
| Label Setting | $n_c = 2$ | $n_g = 2$ | $n_c = 4$ | $n_g = 4$ |
| FARE | $0.92 \pm 0.03$ | $0.88 \pm 0.02$ | $1 \pm 0$ | $0.99 \pm 0$ |
| MixMatch+ | $0.92 \pm 0.03$ | $0.89 \pm 0.01$ | $0.97 \pm 0.01$ | $0.96 \pm 0.01$ |
| Ladder+ | $0.86 \pm 0.05$ | $0.87 \pm 0.01$ | $0.97 \pm 0.02$ | $0.95 \pm 0.02$ |
| DNN+ | $0.88 \pm 0.06$ | $0.88 \pm 0.02$ | $0.98 \pm 0$ | $0.98 \pm 0$ |

TABLE VIII. FARE VS. BASELINES IN TWO UNION CLASSES SETTING.

| Dataset | MALWARE ($N = 6$) | | Network Intrusion ($N = 9$) | |
|---|---|---|---|---|
| Metric | AMI | $K$ | AMI | $K$ |
| FARE | $0.74 \pm 0$ | $6 \pm 0$ | $0.95 \pm 0.01$ | $7 \pm 0.82$ |
| MixMatch+ | $0.48 \pm 0.05$ | $4 \pm 0$ | $0.56 \pm 0.30$ | $5.33 \pm 0.47$ |
| Ladder+ | $0.64 \pm 0.08$ | $4 \pm 0$ | $0.55 \pm 0.28$ | $6.33 \pm 0.47$ |
| DNN+ | $0.45 \pm 0.06$ | $4 \pm 0$ | $0.59 \pm 0.42$ | $4 \pm 0$ |

## APPENDIX-C. AMI VS. ACCURACY.

We use a concrete example to explain the AMI metric and show its advantage over the accuracy metric. Given a highly imbalanced dataset of 1000 samples of three classes, class-1 has 980 samples, class-2 has 18 samples, and class-3 has 2 samples. Suppose a clustering method (denoted as Method A.) categorized the samples into 2 clusters with the contingency table shown in Table VI. Method A. wrongly categorized all the samples from class 2 and 3 into cluster-1, and mistakenly assigned 20 samples from class 1 into cluster-2. Given this contingency table, we can compute the AMI by using the function API in `scikit-learn` [11], i.e., $-0.003$. The mathematical details about AMI can be found in [74]. We can also compute the accuracy as $960/1000 = 0.96$. Despite making serious mistakes, Method A still has an extremely high accuracy. In contrast, AMI is not biased towards the large class and provides a more reasonable score (i.e., a negative score). This is because AMI not only considers the clustering correctness of samples in each class, but more importantly adjusts the score based on the cluster size.

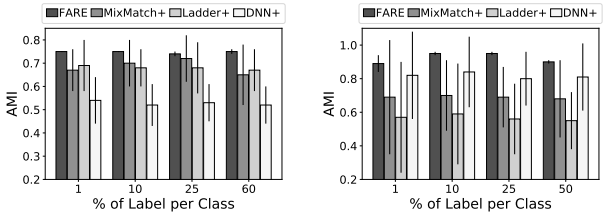
Next, we use real experiment results to discuss the difference between AMI and Accuracy. Table VII shows the *accuracy* of different algorithms under missing-classes and coarse-grained labels. We have two observations. First, FARE is still better than (or at least comparable with) the three baselines under the accuracy metrics. Second, the accuracy metric looks high for all the algorithms, especially for the network intrusion dataset. The reason is that the network intrusion dataset has a dominating benign class, and the highly imbalanced classes lead to misleading accuracy results. For this reason, we used AMI as the primary metric in the paper.

## APPENDIX-D. TIME COMPLEXITY OF FARE.

The computational cost of FARE comes from three aspects. The first part comes from the bases clustering methods. The base clustering algorithms are independent and thus can be run in parallel. As such, the computation bottleneck is the slowest algorithm among DEC ($O(I_d B_d|\theta_d|K)$ [78], k-means ($O(I_k KNp)$) [28], and DBSCAN ($O(pN^2)$) [20]. That is, $\max\{O(I_d B_d|\theta_d|K), O(I_k KNp), O(pN^2)\}$, where $I_d$ and $B_d$ represents the number of training iterations, and the batch size. $|\theta_d|$ is the number of parameters in the DEC model. $I_K$ and $K$ are the number of training iterations and the number of clusters k-means. $N$ is the total number of samples, and $p$ is the dimension of the input space. Given that $I_K K << N$ in most cases, this complexity can be represented as $O(\max\{I_d B_d|\theta_d|K,\ pN^2\})$. The second part comes from computing the neighborhood relationship and training the transformation network. In each iteration, FARE derives the neighborhood relationship for each base method with a cost of $O((B_i)^2)$, and updates the network parameters with the cost of $O(|\theta_i|)$. As such, the time complexity of this part is $O(I_i M(B_i)^2|\theta_i|)$, where $I_i$ and $B_i$ is the number of training iteration and batch size, $|\theta_i|$ is the number of parameters of the transformation network. The third part is introduced by the final k-means clustering. In total, FARE's computational complexity is $O(\max\{I_d B_d|\theta_d|K, pN^2, I_i M B_i^2|\theta_i|\})$.

TABLE IX. DECISION MATRIX FOR LABEL ALIGNMENT.

| Prediction | Matching Score | | Conf. | Assignment |
|---|---|---|---|---|
| | Class1 | Class2 | | (Conf.$\geq 0.9$) |
| Cluster1 | 0.9 | 0.1 | 0.9 | Class1 |
| Cluster2 | 0.4 | 0.6 | 0.6 | New Class |
| Cluster3 | 0.05 | 0.95 | 0.95 | Class2 |
| Cluster4 | 0.1 | 0.9 | 0.9 | Class2 |




(a) Malware categorization. (b) Intrusion detection.

Fig. 6. The performance of FARE and baselines under the missing class setting (*i.e.,* $n_c = \lfloor n/2 \rfloor$) with different ratios of available labels.

## APPENDIX-E. ADDITIONAL EXPERIMENTS.

**Multiple Union Classes.** We compared FARE with the amended baselines in a coarse-label setting with two union classes. Specifically, we randomly selected 6 classes from the malware and intrusion dataset and relabeled them into two union classes. Together with the remain original classes, the training set has in total $n - 4$ classes. We still only used 1% of the labels for each class. We ran each method 10 times with different random seeds. As shown in Table VIII, FARE achieves significantly higher AMIs and lower standard deviations than all the amended baselines on both datasets. In addition, FARE could identify the true number of classes on the malware dataset and has lowest estimation errors on the intrusion dataset. This result demonstrates that FARE could generalize its effectiveness in multiple union class settings.

**Setup w/ both Missing Classes and Coarse-grained Labels.** We tested FARE under a setting with both missing classes and coarse-grained labels. Specifically, we randomly selected 4 classes from both datasets, relabeled 2 of them as a union class, and eliminated the labels of the rest. In this way, we constructed a training set with $n - 3$ classes. Similarly, we preserved 1% labels in each class and ran FARE 10 times. The results AMI and $K$ are: $(0.76 \pm 0,\ 6 \pm 0)$ on the malware dataset, and $(0.92 \pm 0.06,\ 5.67 \pm 2.36)$ on the intrusion dataset.

## APPENDIX-F. CLASS-CLUSTER ALIGNMENT.

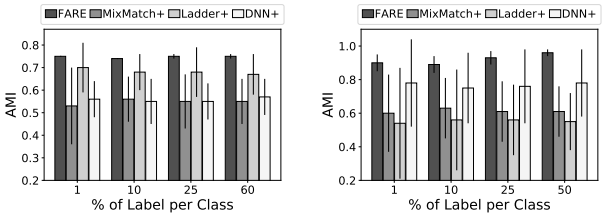
We use an example to explain the strategy of post-clustering analysis introduced in Section §VI. Suppose we

TABLE X. MEAN AMIS AND STANDARD DEVIATIONS OBTAINED BY VARYING THE $\alpha$ AND $q$ IN FARE.

| Label Condition | $n_c = \lfloor n/2 \rfloor$, 1% labels | | | | $n_g = \lfloor n/2 \rfloor$, 1% labels | | | |
|---|---|---|---|---|---|---|---|---|
| # $q$ and $\alpha$ | 8 | 16 | 32 (Our choice) | 64 | 8 | 16 | 32 (Our choice) | 64 |
| Malware | $0.76 \pm 0.05$ | $0.78 \pm 0.02$ | $0.75 \pm 0$ | $0.74 \pm 0.06$ | $0.74 \pm 0$ | $0.75 \pm 0$ | $0.74 \pm 0$ | $0.74 \pm 0$ |
| Intrusion | $0.89 \pm 0.05$ | $0.86 \pm 0.02$ | $0.89 \pm 0.05$ | $0.89 \pm 0.06$ | $0.87 \pm 0$ | $0.87 \pm 0$ | $0.90 \pm 0.05$ | $0.85 \pm 0.04$ |

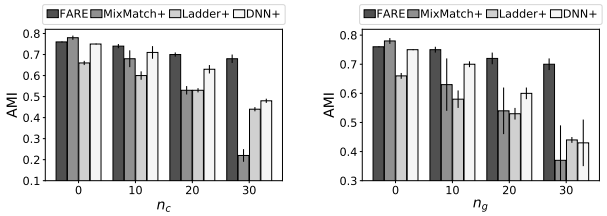
TABLE XI. THE PERFORMANCE OF FARE UNDER DIFFERENT PERCENTAGES OF CORRUPTED LABELS.

| Dataset | Noisty Label Percentage $x$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | | 10 | | 25 | | 50 | | 75 | |
| | AMI | Accuracy | AMI | Accuracy | AMI | Accuracy | AMI | Accuracy | AMI | Accuracy |
| Malware | $0.87 \pm 0.01$ | $0.97 \pm 0$ | $0.83 \pm 0.01$ | $0.96 \pm 0.01$ | $0.81 \pm 0.01$ | $0.96 \pm 0.01$ | $0.79 \pm 0.03$ | $0.82 \pm 0.05$ | $0.73 \pm 0.01$ | $0.74 \pm 0$ |
| Network Intrusion | $0.98 \pm 0$ | $1 \pm 0$ | $0.94 \pm 0.03$ | $0.99 \pm 0$ | $0.84 \pm 0.05$ | $0.99 \pm 0$ | $0.75 \pm 0.02$ | $0.98 \pm 0$ | $0.75 \pm 0.01$ | $0.98 \pm 0$ |




(a) Malware categorization. (b) Intrusion detection.

Fig. 7. The performance of FARE and baselines under the coarse-grained label setting (*i.e.,* $n_g = \lfloor n/2 \rfloor$) with different ratios of available labels.




(a) Missing class. (b) Coarse-grained label.

Fig. 8. Performance comparison of FARE and baselines under missing class and coarse-grained label settings on the GTSRB dataset.

have the clustering result of a dataset, we first compute the matching score of between every cluster-class pair, constructed a decision matrix in Table IX. Then, we obtain the confidence score for each cluster (*i.e.,* the maximum matching score among all known classes). Finally, we set the threshold as $0.9$ and assigned the clusters to the known classes, accordingly. As we can observe from the assignment (column 5 in Table IX), we set cluster 2 as an unknown (possibly new) class, because its confidence is below the threshold. In addition, both cluster 3 and 4 are assigned to class 2. This indicates that class 2 represents a coarse-grained label.

## APPENDIX-G. HYPER-PARAMETER SENSITIVITY

**Sensitivity to Ratio of Available Labels.** We evaluate the impact of the ratio of available labels: For the missing class settings, we fixed $n_c = \lfloor n/2 \rfloor$ for each dataset, and then varied the ratio of labeled samples in the training data as $1\%$, $10\%$, $25\%$, and $50\%$ to construct $4$ groups of training sets. We ran each method by sampling the missing classes $10$ times and reported testing AMIs. For the coarse-grained label settings, we fixed $n_g = \lfloor n/2 \rfloor$ for each dataset and followed the same procedure. In both experiments, we set the number of clusters $K$ to the corresponding ground-truth number.

Figure 6 and Figure 7 show the results. The performances of all the methods are quite consistent with respect to different ratios of available labels. This result indicates that FARE is not sensitive to the raito of labeled samples. Also, the results show that $1\%$ labeled data is enough for FARE to achieve a high AMI — the extra information provided by the clustering ensemble has helped to boost the performance.

**Sensitivity to the Latent Dimensionality.** We also tested the sensitivity of FARE to the output dimension of the transformation network $q$, and the distance radius $\alpha$. As mentioned in Section §III-D, $\alpha = q$, and thus we changed them together. We used the above setups (*i.e.,* $n_c = \lfloor n/2 \rfloor$ and $n_g = \lfloor n/2 \rfloor$) with $1\%$ labels and $151$ neighborhood models. We set the $\alpha$ (and $q$) as 8, 16, 32, and 64, and ran FARE $10$ times per setting. As shown in Table X, subtly varying $\alpha$ and $q$ in a certain range does not affect the performance of FARE.

## APPENDIX-H. FARE ON DATASET WITH MORE CLASSES.

In Section §IV, we demonstrated the effectiveness of FARE on two security datasets with 6 and 9 classes, respectively. Here, we further tested it on an image dataset with more classes. we used a traffic sign recognition dataset called GTSRB [66] which has 39,209 training samples and 12,630 testing samples. The dataset has 43 classes, each of which represents a type of traffic sign (*e.g.,* stop sign, speed limit). We followed the same missing-class and coarse-grained label setups as other experiments. For each labeled class, we randomly selected $10\%$ training samples as labeled data, and set $n_c$ and $n_g$ as 0, 10, 20, and 30. For each setup, we ran FARE and the baselines (DNN+, MixMatch+, and Ladders+) 10 times and reported the testing AMIs. In Figure 8, we observe similar results as those on the malware and intrusion dataset. FARE is comparable with the baselines on the original dataset, and outperforms the baselines when handling low-quality labels. This result confirms the advantage of FARE over other methods under a large number true classes.

## APPENDIX-I. FARE UNDER CORRUPTED LABELS.

We tested FARE under corrupted labels (which can be caused by either random labeling errors or poisoning attacks). Specifically, we considered a setup without missing classes and coarse-grained labels. By following the setup in [35], we randomly sampled $x\%$ of the labeled samples and relabeled each sample with a wrong class randomly selected from the known classes. Here, we set $x$ =10, 25, 50, 75. For each $x$, we ran FARE on the corrupted training data $10$ times and reported the testing AMIs. As shown in Table XI, the performance of FARE is still reasonably good, even when $x = 75$. This is again because the clustering ensemble mitigates the negative influence of incorrect labels. However, compared to the clean label setting (Table I), the performance drops as more labels are corrupted. The corrupted labels indeed impose a negative influence upon FARE. In Section §VI, we have discussed the potential solutions to alleviate the negative impact.