Feature Distribution Shift Mitigation with Contrastive Pretraining for Intrusion Detection

Weixing Wang[*†], Haojin Yang[*], Christoph Meinel[*],
Hasan Yagiz Özkan[†], Cristian Bermudez Serna[†], Carmen Mas-Machuca[‡†]
[*]Hasso Plattner Institut (HPI), Germany
[†]Chair of Communication Networks (LKN), Technical University of Munich (TUM), Germany
[‡]Chair of Communication Networks, University of the Bundeswehr Munich, Germany
{weixing.wang, haojin.yang, christoph.meinel}@hpi.de, {yagiz.oezkan, cristian.bermudez-serna}@tum.de, cmas@unibw.de

***Abstract*—In recent years, there has been a growing interest in using Machine Learning (ML), especially Deep Learning (DL) to solve Network Intrusion Detection (NID) problems. However, the feature distribution shift problem remains a difficulty, because the change in features' distributions over time negatively impacts the model's performance. As one promising solution, model pretraining has emerged as a novel training paradigm, which brings robustness against feature distribution shift and has proven to be successful in Computer Vision (CV) and Natural Language Processing (NLP). To verify whether this paradigm is beneficial for NID problem, we propose SwapCon, a ML model in the context of NID, which compresses shift-invariant feature information during the pretraining stage and refines during the finetuning stage. We exemplify the evidence of feature distribution shift using the Kyoto2006+ dataset. We demonstrate how pretraining a model with the proper size can increase robustness against feature distribution shifts by over 8%. Moreover, we show how an adequate numerical embedding strategy also enhances the performance of pretrained models. Further experiments show that the proposed SwapCon model also outperforms eXtreme Gradient Boosting (XGBoost) and K-Nearest Neighbor (KNN) based models by a large margin.**

***Index Terms*—Network Intrusion Detection (NID), Machine Learning (ML), Feature Distribution Shift**

I. Introduction

A NID system monitors and analyzes network traffic to identify and respond to unauthorized or malicious activities. These systems play a vital role in protecting modern networks from threats such as malware, ransomware, and other cyber-attacks [1]. According to the 2022 Cyber-threat Defense Report [2], the number of organizations that experienced at least one successful cyber-attack increased from 61.9% to 81.3% since 2014, and the mean annual IT security budget increased by 4.6% in the year 2022. Failing to secure a network properly can have severe financial and reputational consequences. For example, the 2017 Equifax data breach, which exposed the personal data of 147 million people, resulted in a loss of $439 million for the company.

ML algorithms can quickly and accurately identify patterns in large amounts of data. This ability makes ML models well suited for detecting anomalies, which may indicate intrusions in a network. However, applying ML for NID comes with the challenge of the feature distribution shift. For example, consider an ML model that is trained to detect network intrusions using data collected during 2005 and is later deployed in the same network in 2006. The model may not perform as accurately as it did during training due to the change in network configurations and user behaviors, which leads to a shift in the distribution of the features in the dataset that the ML model has not seen during training.

Model pertaining is a promising method to mitigate the aforementioned problem. In pretraining, an ML model is trained on a large dataset in an unsupervised manner to learn general patterns, which can be later finetuned for specific tasks. For instance, Hendrycks et al. [3] demonstrated how pretraining can increase model robustness against feature distribution shift by approximately 10% in image classification tasks.

The main contributions of this work are: *i)* we introduce SwapCon, a pretrained ML model robust against the feature distribution shift for NID. *ii)* We leverage contrastive pretraining for finding pattern representations in the data and a swapping augmentation strategy for creating positive and negative samples. Moreover, we use different embedding methods for numerical and categorical features to increase the feature space. *iii)* We test our model on the open-source Kyoto2006+ dataset [4], which contains 10 years of network traffic information. And *iv)* we study the effect of pretraining using the SwapCon model, while comparing our best variant with a KNN and an XGBoost based models. The results show that model pretraining increases robustness against the feature distribution shift in the selected dataset, while the other models suffer from performance drops.

II. Preliminary

A. Feature Distribution Shift

Feature Distribution Shift refers to a change in the distribution features between the training and testing sets. It can be caused by various factors, such as changes in user behavior, technological advancements, and shifts in popular applications or services.

B. Contrastive Learning

Contrastive Learning (CL) aims to find a proper latent space for the original feature space. In this space, similar samples

This work has received funding from the German Research Fundation (DFG) under grant numbers MA 6529/4-1 and KE 1863/10-1.

have a closer distance than dissimilar samples. To this end, a contrastive objective function is needed. In this work, the Noise Contrastive Estimation (NCE) loss function is applied. It converts the problem of modeling complex probability distributions of two feature spaces into a simpler binary classification task, where the model discriminates between genuine data samples and noisy samples, making training more computationally efficient and scalable.

C. Model Pretraining

Pretrained models have achieved massive success in the field of CV as well as NLP and have become the new paradigm for many ML tasks [5]. Model pretraining, as depicted in Fig. 1, adds an initial stage in the model training pipeline, where the model is pretrained on related datasets. Those datasets should be large and diverse, covering a wide range of variations, patterns, and examples relevant to the target task. For example, the famous BERT model is pretrained using many text sources such as English Wikipedia. The goal in the pretraining stage is to capture as much knowledge as possible from the data and store it in the model. After pretraining, the model is finetuned on the task-specific dataset. The knowledge gained during pretraining enables the model to efficiently learn the target task. For example, a pretrained language model can be finetuned for either sentiment analysis, language translation, or sentence classification.

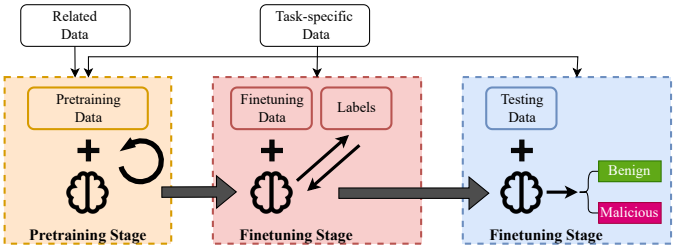

Fig. 1. ML model training pipeline with pretraining.

In tabular data, each piece of sample is a row containing a series of features. For this data representation, it is also possible to perform CL. Somepalli et al. [6] proposed a pretraining framework that first applies augmentation operations on rows and trains a transformer-based encoder using a contrastive objective function. Positive pairs are generated by mixing up several rows in a batch and the NCE loss function is used to minimize the distance between positive samples while distancing negative samples. In this work, contrastive pretraining is applied with a different data augmentation method.

D. ML Models for the NID Problem

Decision Trees (DTs) are one of the basic supervised ML algorithms used for classification tasks. Despite their simplicity, DT models show superior performance in many tasks, including NID. XGBoost is a DT model designed for gradient-boosting. The key idea about gradient boosting is to assemble multiple single DTs to boost the model performance. In [7], Dhaliwal et al. applied the XGBoost model on the NSL-KDD dataset and reached a 98.7% accuracy score on the test dataset. To the best of our knowledge, [8] is the first and so far, the only work that studies the feature distribution shift using a NID dataset, where the researchers proposed both ML and DL models.

III. Problem

Using the same visualization method of Drăgoi et al. [8], we show the evidence of feature distribution shift over time in the Kyoto2006+ dataset. In Fig. 2, the distribution of the network traffic feature *Dst_host_srv_count* is visualized on a yearly basis. As we can see, the feature distributions are similar within the first five years. Then, there is a sudden change starting from the year 2011.

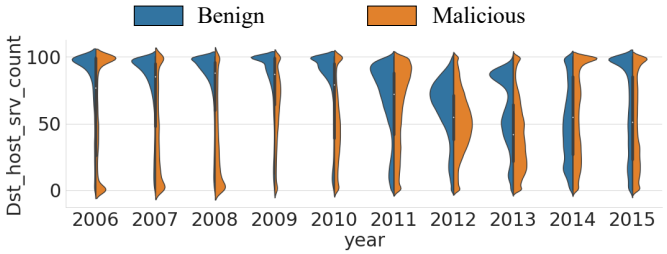

Fig. 2. Visualization of the shapes of a feature distribution shift over time in the Kyoto2006+ dataset [4]. In each year, the horizontal expansion of the feature plot shows its probability density. The Y-axis means the percentage value that ranges from 0 to 100 which relates to the feature values.

Although the feature distribution shift is often present, it is likewise overseen. Song et al. demonstrated that the feature distributions of network traffic in the Kyoto2006+ dataset vary even from month to month [4]. Ignoring the feature distribution shift during model training can lead to serious consequences in the testing or deployment stages. Koh et al. constructed a benchmark containing datasets with different types of feature distribution shifts and demonstrated how standard training strategies yield substantially worse results in those datasets [9].

One way of mitigating the distribution shift problem is to continuously update and retrain ML models using the most recent and representative data. This allows the models to adapt to the changing distributions and maintain their accuracy in classifying network traffic. However, the cost of training models and labeling new data is often too large to be practical.

IV. Design

A. Dataset

The Kyoto2006+ dataset [4], which was built on 10 years of real network traffic data from 2006 to 2015, is used in this work. It consists 24 features and two classes which are denoted as benign and malicious. In our study, 14 out of 24 features are used to train ML models since the other 10 features are not generic. We refer the reader to [4] for more details.

We follow the proposed chronological approach in [8] to split all 10 years of data into three main splits that can

highlight the feature distribution shift over time. Fig. 3 shows the three splits on a time axis as well as the train/test split.

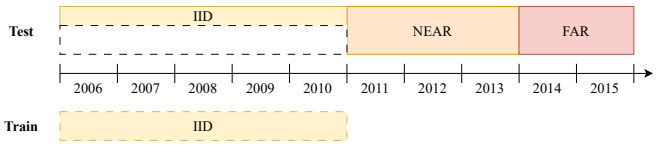

Fig. 3. Splits of *IID*, *NEAR*, and *FAR* based on 10 years of data. Note that the training set is sampled from the *IID* split.

IID, *NEAR* and *FAR* represent the different degrees of feature distribution shift. We denote the first split as *IID* (Independent and Identically Distributed) because the data distribution is the same as the training set. We assume that more recent data with respect to the *IID* split should have less feature distribution shift than more distant data. Models are pretrained and finetuned within the training set and tested in the three splits separately. The training set is divided into pretraining and fine-tuning sets with a ratio of 9:1, respectively. To remove the bias in the testing stage, 5000 benign and 5000 malicious samples are sampled from the original testing set, making in total of 10000 for each testing set.

B. SwapCon Overview

SwapCon consists of a stack of blocks where each block is a combination of a linear layer followed by a Batch Normalization (BN) layer. BN normalizes the layer activations in a neural network within each batch during training. By reducing the internal covariate shift, which is the change in the distribution of activations across different layers during training, BN facilitates faster learning and helps the network reach a better optimum. In SwapCon, two kinds of activation functions are applied after linear layers. The *Tanh* is used in the first layer and the *ReLu* activation function is applied in the rest of the layers.

C. Contrastive Pretraining

In pretraining, CL is leveraged to inject invariant latent features in the model weights in a self-supervised way. Fig. 4 depicts the high-level pipeline of the SwapCon pretraining procedure. The input is a mini-batch of samples which are represented as rows in the figure. In the first step, the feature embedding operation embeds the numerical or categorical feature into a higher dimension. After the data augmentation operation, both the original and augmented batches are processed by the same neural network. The NCE loss function in Eq. 1 is applied to the outputs.

CL relies on positive and negative sample pairs, which requires finding at least one negative and one positive variant of a single sample. While it is easy to regard other samples in the same mini-batch as negative ones, data augmentation techniques are applied to generate positive ones. In SwapCon, sample-based data augmentation is applied by randomizing the order of features in a sample. The augmented sample preserves all original information and thus is regarded as a positive instance. By comparing the contrastive pairs, the model learns the order-invariant characteristic of tabular data. This method yields a simple and effective self-supervised task.

$$\mathcal{L}_{NCE}^{(i,j)} = -\log \frac{\exp\left(sim\left(\mathbf{h}_i, \mathbf{h}_j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp\left(sim\left(\mathbf{h}_i, \mathbf{h}_k\right)/\tau\right)} \quad (1)$$

In pretraining, the NCE loss function of Eq. 1 is considered, where $\mathbb{1}_{[k\neq i]}$ is the identity function and τ is a hyperparameter that controls the importance of negative samples. For simplicity, we use one as the default τ value. $\mathbf{h}_i$ and $\mathbf{h}_j$ represent the original sample and its augmented version, respectively. The similarity function *sim* computes the cosine similarity between the two representations. Given an anchor sample h_i, the model tries to distinguish the positive from a set of negative samples by minimizing the $\mathcal{L}_{NCE}^{(i,j)}$ function.

D. Finetuning Strategy

After the pretraining stage, the model is finetuned with supervised learning. Fig. 5 depicts this procedure. Here, additional classification layers are stacked after the pretrained model for the binary classification task.

There are two ways for finetuning. The first one is called full finetuning, where the weights of both the pretrained model and the classification layer are updated during training. The second is called partial finetuning, where only the weights of the last classification layers are updated and the weights in the pretrained model are fixed. While the first method leads to faster convergence and possibly higher performance, it breaks the data invariance information learned during the pretraining stage. Both methods are applied and discussed in Section V.

E. Numerical Feature Embedding

Guo et al. demonstrated that numerical feature embedding often brings better performance for ML models [10]. Hence, three numerical feature embedding strategies are implemented and studied. The first two methods are variants of the binning method, which is used to categorize a range of varied, continuous values into a smaller number of distinct bins.

1) Exponential Binning (EB): This is used when the data is concentrated at one end of the value range. As such, bins have finer intervals where the numerical values are also denser. As depicted in Fig. 6, the solid line represented the range of a numerical feature in the training set and a bin is marked within $[b_t, b_{t+1}]$. Every numerical feature that has the value covered by a bin will be assigned to the corresponding category. This is also the baseline embedding method used in our experiments.

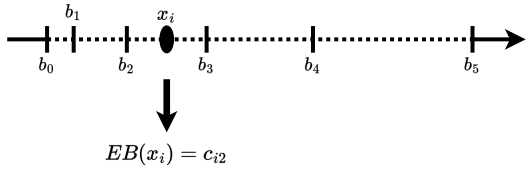

Fig. 6. Illustration of the EB strategy. Here x_i is the i-th feature in the sample.

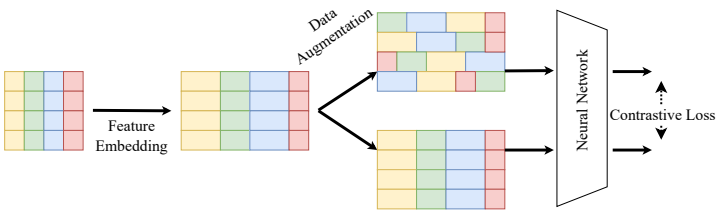

Fig. 4. SwapCon pretraining pipeline. Colors represent different features.

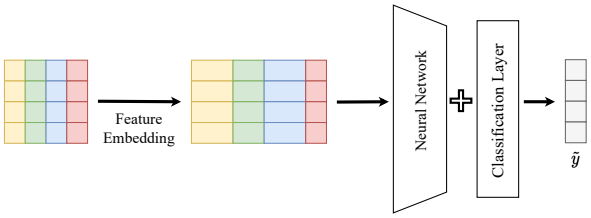

Fig. 5. The finetuning pipeline of SwapCon.

2) Piecewise Linear Embedding (PLE): Proposed by Gorishniy et al. [11] is inspired by the one-hot embedding method with a more precise representation of the numerical value. In PLE, each feature value range is split into a disjoint set of bins with the same width. A numerical feature is represented with a vector of length T, where T is the number of bins. We set T to be 128 in all experiments. Fig. 7 depicts the PLE encoding strategy.

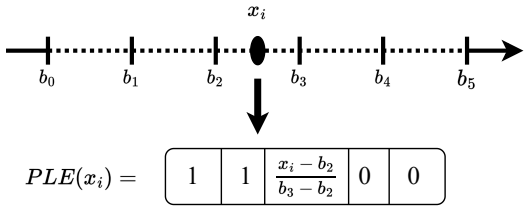

Fig. 7. Illustration of the PLE strategy.

PLE offers multiple benefits. As a preprocessing strategy, PLE provides a more fine-grained numerical value representation. Normally, a part of the information is lost when a scalar number is categorized into bins, but PLE establishes a direct bijection between the single dimension and a higher dimension, which enlarges the feature space for the following learning process. Moreover, PLE preserves the inherent magnitude relationship of the numbers. In the EB method, numbers are converted into categories, thus losing their magnitude and can not be compared. But with PLE, two numbers can still be compared after the embedding.

3) Learnable Embedding (LE): It is a model-aware embedding schema, where an encoder for each numerical feature is jointly trained with the model in the pretraining stage. Each numerical feature of a sample is fed into a linear layer that does not share weights with others. After pretraining, the encoders are fixed. In LE, the same values of different features should have different meanings and thus different representations. Fig. 8 shows how LE works.

V. Evaluation

A. Pretraining with SwapCon

In the first experiment, SwapCon is pretrained with a network with three layers, and another three classification layers are added during the finetuning stage. The model is pretrained with two epochs and a mini-batch with 512 samples. Note that all experiments are averaged over three executions.

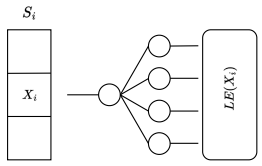

Fig. 8. Illustration of the LE strategy. The size of the embedding is controlled by the linear layer.

TABLE I
Performance of model with 6 layers.

AUC(↑)	With Pretrain		Without Pretrain	
	Fix	No-Fix	Fix	No-Fix
IID	98.83	**99.28**	97.58	99.12
NEAR	94.25	**98.81**	94.01	97.23
FAR	67.92	42.71	**68.81**	42.08

B. Finetuning with SwapCon

In the finetune stage, the early-stopping strategy is adopted, which means the training procedure will be stopped when the performance on the validation set starts to deteriorate. With this strategy, all training finishes within two epochs. For simplicity, we use the Area Under the Curve (AUC) metric on a balanced testing set.

As shown in Tbl. I, four variants of models are presented. Note that *Fix* means partial finetuning and *No-Fix* means full finetuning. Models without pretraining would have random initial parameter weights for the first three layers, while models with pretraining inherit weights from that stage. Here, partial finetuning means fixing the weights from the first three layers and only training the classification layers during finetuning. The best results in each split are denoted in bold. By comparing the table rows, we see that the performances of all variants decrease as the time distance becomes larger. Comparing against the results in the *IID* split, the performance drops from 0.5% to 4% in the *NEAR* split and up to 50% in the *FAR* split. This demonstrates that feature distribution shifts can substantially undermine model performance. A larger gap in feature distributions means a larger drop in performance.

TABLE II
MODEL PERFORMANCE WHEN FIXING FROM 0 TO ALL LAYERS.

AUC(↑)	No-Fix	Fix 1. Layer	Fix 1∼2 Layers	Fix 1∼3 Layers	Fix 1∼4 Layers	Fix 1∼5 Layers	Fix 1∼6 Layers
IID	**99.12**	98.31	98.33	97.93	96.77	91.31	49.76
NEAR	97.32	**97.68**	97.63	95.27	92.67	86.77	51.59
FAR	42.08	50.28	52.49	64.89	**65.84**	65.07	53.63

However, the next section shows that a more sophisticated pretraining strategy can mitigate the problem to some extent.

1) Effect of Pretraining: The difference between pretrained and non-pretrained models lies in the first layers. Those pretrained layers are expected to deliver better data representations which could facilitate finetuning. By comparing column one with column three and column two with column four in Tbl. I, we could see that the model performance increases for the *IID* and *NEAR* splits when using pretrained weights. It should not be surprising for the *IID* split because the pretraining is also done using the data from this split. We could interpret the gain in performance as a result of training on more data. However, thanks to the knowledge learned through the unsupervised pretraining, the model performance increases by around 1% in the *NEAR* split. Nevertheless, pretraining seems to have side effects in the *FAR* split. A pretrained model does not necessarily perform better than the model trained from scratch. This may be because the knowledge learned in the pretraining stage becomes less representative when the feature distribution gap becomes large.

2) Effect of Fixing Model Weights: By looking at Tbl. I again, we could also find whether fixing the model weights also has different effects on the three splits. This time we compare column one with column two and column three with column four. For the *IID* and *NEAR* splits, there is a slight increase when the first layers are not fixed, and for the *FAR* split there is a huge decrease of over 25%. When fixing the first layers, the trainable parameters become less, which is equivalent to having a simpler model. And since the model is trained on the *IID* split, the performance drop in this split is understandable. The *NEAR* split is very close to the *IID* split and the same effect applies. However, since the distributions of the features in the *FAR* split are very different from those in the other two sets, a more complex model that learns too "well" on the *IID* split suffers from the feature distribution shift the most. In contrast, a simpler model cannot be optimized as precisely as a complex model in the *IID* split, but it gains some robustness against dissimilar data in the *FAR* split.

C. Determining the Best Model Size

From the previous results, we found that although larger models can perform better in the *IID* and *NEAR* splits, they generalize worse in the *FAR* splits. So we trained models with different numbers of fixed layers to verify the hypothesis between the model size and their generalization ability. As shown in Tbl. II, as more layers are fixed, the model performance decreases in the *IID* and *NEAR* split. In the *FAR* split, the model performs better in the beginning but then worsens in the end. We further notice that when using only three layers, the model performs well enough against feature distribution shift in the *FAR* split and preserves the most performance in the other two splits.

Combining the above analysis, we train a model with three layers. The first two layers are pretrained and the last layer is the classification layer which is finetuned. Tbl. III includes all variants. In the first row, SwapCon(3+3) means that the models have six layers, in which the first three layers are pretrained. Likewise, SwapCon(2+1) means that out of a total three layers, the first two of them are used for pretraining. The best combination is marked with *.

By comparing column seven with column eight and column five with column six in Tbl. III, we find that fixing the first two layers without pertaining will decrease the model performance since only one layer can be trained. At the same time, when the first two layers are pretrained, fixing them would bring a performance boost, which addresses the benefits of pretraining.

D. Better Numerical Embedding

As demonstrated by Guo et al. in [10], the importance of numerical feature embedding is often overlooked. When looking at the data, we notice that different features may have the same numerical values, but the meanings behind the numbers differ. Thus various features should have different embedding vectors in a latent feature space. To bring scalar features into higher dimensions, we adopt the three previously introduced numerical feature embedding strategies and evaluate their impacts on the pretrained models. Notice that the EB method is the baseline that is used in the previous experiments.

Tbl. IV shows the results on selected models. Note that for simplicity we only use the best model from the previous section. As we can see, the PLE method offers a 0.5%∼0.7% percent performance increase for all three splits. The LE method delivers the best results in *IID* and *NEAR* splits; however, it decreases the model performance in the *FAR* split. The reason may be the mismatch between feature representations and the actual samples. When pretraining with the LE method, we essentially train a series of 1-layer neurons for numerical feature representation only considering the data in the *IID* split. When we test the model in the *IID* and *NEAR* splits, the fixed LE components can provide the most suitable embeddings so that the final results are better than the other methods. But this way of embedding numerical features becomes an unwanted bias in the *FAR* split and leads to the

TABLE III
ALL VARIANTS OF THE MODEL TRAINED WITH SWAPCON.

AUC(↑)	SwapCon (3+3)				SwapCon (2+1)*			
	With Pretrain		Without Pretrain		With Pretrain*		Without Pretrain	
	Fix	No-Fix	Fix	No-Fix	Fix*	No-Fix	Fix	No-Fix
IID	98.83	**99.28**	97.58	99.12	98.64	98.99	94.82	97.53
NEAR	94.25	**98.81**	94.01	97.23	98.32	98.12	89.63	95.14
FAR	67.92	42.71	68.81	42.08	**72.31**	70.99	64.02	64.78

TABLE IV
IMPACT OF DIFFERENT NUMERICAL EMBEDDING TECHNIQUES.

AUC(↑)	SwapCon (2+1) & With Pretrain & Fix		
	EB	PLE	LE
IID	98.64	99.35	**99.65**
NEAR	98.32	98.85	**98.94**
FAR	72.31	**72.93**	70.99

worst performance. This experiment shows that embeddings should be contextualized and updated for new data.

E. Comparison

This section presents the performance of two traditional ML models, XGBoost and KNN. We choose XGBoost because it is often used in NID studies [7], and it can perform well on the general tabular dataset. The KNN is also included because it also learns by comparing sample distance in the dataset, which is relevant to the concept of CL. Tbl. V shows the results on three dataset splits, including our best SwapCon model. We regard the best SwapCon model as the combination described in Tbl. IV with the PLE numerical embedding strategy.

TABLE V
COMPARISONS BETWEEN SWAPCON AND ML MODELS.

AUC(↑)	SwapCon best	XGBoost	KNN
IID	**99.35**	97.29	93.71
NEAR	**98.85**	83.87	82.84
FAR	**72.93**	47.31	32.36

The results show that the two ML models are both not robust to feature distribution shifts. However, the XGBoost model performs better than KNN on all three splits.

VI. CONCLUSION

In this paper, we addressed the problem of feature distribution shift in Network Intrusion Detection (NID) and proposed SwapCon, a solution using model pretraining. Our approach involves compressing time-invariant feature information into the model during the pretraining stage and refining the model for classification in the finetuning stage. We also explore the importance of numerical feature embedding in improving the accuracy of Machine Learning (ML) models. We evaluated our approach on the Kyoto2006+ dataset and compared it with XGBoost and KNN based models. Our results show that pretraining with SwapCon increases robustness against feature distribution shift and outperforms the other ML models in terms of accuracy. Pretraining brings the most benefit in the *FAR* split where feature distribution shift affects the dataset the most. Our experiments also suggest that numerical feature embedding is an important factor in improving the performance of pretrained models. We also show that the model size impacts the pertaining gain. An in-depth study of the relationship between model size and pretraining gain remains for future work.

REFERENCES

[1] B. Mukherjee, L. Heberlein, and K. Levitt, "Network Intrusion Detection," *IEEE Network*, vol. 8, no. 3, pp. 26–41, 1994.

[2] Cyber-Edge Group. "2022 Cyberthreat Defense Report". Accessed: September 13, 2023. [Online]. Available: https://cyber-edge.com/resources/2022-cyberthreat-defense-report/

[3] D. Hendrycks, K. Lee, and M. Mazeika, "Using Pre-training Can Improve Model Robustness and Uncertainty," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2712–2721.

[4] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical Analysis of Honeypot Data and Building of Kyoto 2006+ Dataset for NIDS Evaluation," in *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security*, 2011, pp. 29–36.

[5] S. Long, F. Cao, S. C. Han, and H. Yang, "Vision-and-language Pretrained Models: A Survey," *arXiv preprint arXiv:2204.07356*, 2022.

[6] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Bruss, and T. Goldstein, "Saint: Improved Neural Networks for Tabular Data Via Row Attention and Contrastive Pre-training," *arXiv preprint arXiv:2106.01342*, 2021.

[7] S. S. Dhaliwal, A.-A. Nahid, and R. Abbas, "Effective Intrusion Detection System Using XGBoost," *Information*, vol. 9, no. 7, p. 149, 2018.

[8] M. Drăgoi, E. Burceanu, E. Haller, A. Manolache, and F. Brad, "AnoShift: A Distribution Shift Benchmark for Unsupervised Anomaly Detection," *arXiv preprint arXiv:2206.15476*, 2022.

[9] P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao *et al.*, "Wilds: A Benchmark of In-the-wild Distribution Shifts," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5637–5664.

[10] H. Guo, B. Chen, R. Tang, W. Zhang, Z. Li, and X. He, "An Embedding Learning Framework for Numerical Features in CTR Prediction," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2910–2918.

[11] Y. Gorishniy, I. Rubachev, and A. Babenko, "On Embeddings for Numerical Features in Tabular Deep Learning," *arXiv preprint arXiv:2203.05556*, 2022.