# Towards Privacy-Preserving Classification-as-a-Service for DGA Detection

Arthur Drichel*, Mehdi Akbari Gurabi†, Tim Amelung‡, and Ulrike Meyer*
* RWTH Aachen University, Germany: {drichel, meyer}@itsec.rwth-aachen.de
† Fraunhofer FIT, Germany: mehdi.akbari.gurabi@fit.fraunhofer.de
‡ RWTH Aachen University, Germany: tim.amelung@rwth-aachen.de

***Abstract*—Domain generation algorithm (DGA) classifiers can be used to detect and block the establishment of a connection between bots and their command-and-control server. Classification-as-a-service (CaaS) can separate the classification of domain names from the need for real-world training data, which are difficult to obtain but mandatory for well performing classifiers. However, domain names as well as trained models may contain privacy-critical information which should not be leaked to either the model provider or the data provider. Several generic frameworks for privacy-preserving machine learning (ML) have been proposed in the past that can preserve data and model privacy. Thus, it seems high time to combine state-of-the-art DGA classifiers and privacy-preservation frameworks to enable privacy-preserving CaaS, preserving both, data and model privacy for the DGA detection use case.**

**In this work, we examine the real-world applicability of four generic frameworks for privacy-preserving ML using different state-of-the-art DGA detection models. Our results show that out-of-the-box DGA detection models are computationally infeasible for privacy-preserving inference in a real-world setting. We propose model simplifications that achieve a reduction in inference latency of up to 95%, and up to 97% in communication complexity while causing an accuracy penalty of less than 0.17%. Despite this significant improvement, real-time classification is still not feasible in a traditional two-party setting. Thus, more efficient secure multi-party computation (SMPC) or homomorphic encryption (HE) schemes are required to enable real-world feasibility of privacy-preserving CaaS for DGA detection.**

***Index Terms*—domain generation algorithm (DGA) detection, classification-as-a-service, privacy-enhancing technologies**

## I. Introduction

Domain generation algorithms (DGAs) are incorporated in many malware families and used to impede the blocking of the connection establishment between bots and their command and control (C2) server. For this purpose, they pseudo-randomly generate a huge number of domain names which are then queried. Nearly all of the queries result in non-existing domain (NXD) responses. The botnet master knows the generation scheme and can thus register a subset of the generated domains in advance. As soon as a bot queries one of the registered domains, it receives the valid IP address of the C2 server and is now ready to receive commands for malicious actions.

Machine learning (ML) can be used to train classifiers that are capable of separating benign NXDs, e.g., generated by typos or by misconfigured software, from malicious NXDs generated by DGAs. However, there are two hurdles to overcome. First, ML is a computationally expensive task and thus requires considerable computing power. Second, it takes a significant amount of real-world training data, often difficult to obtain, to produce classifiers that perform well.

Classification-as-a-service (CaaS) circumvents these issues by providing classification services on powerful cloud servers to a multitude of clients, enabling even resource constrained devices to benefit from the advantages of ML without having to access training data.

Many interesting ML applications classify sensitive data, for example medical information for cancer diagnosis [1]. Privacy-preserving measures can be used to reduce potential leakage of sensitive information. However, privacy preservation is particularly challenging in ML as there is an inherent need for huge amounts of data. Additionally, CaaS providers intend to protect their model from being stolen, as it is their intellectual property [2]. Consequently, privacy-preserving measures that protect the privacy of data and models are imperative.

NXD information as used by DGA detection models is generally sensitive, particularly in an enterprise environment. For instance, typos can indicate websites visited by employees and customers. Unintentional access to these could even increase the success rate of possible phishing attacks. Therefore, privacy measures are essential as revealing domain name information in plaintext to a third-party service provider for DGA detection is not desirable.

Several frameworks for privacy-preserving ML have been proposed in the past. However, their application comes with non-negligible costs compared to plaintext classifiers. First, the inference latency increases notably due to a substantial increase in computational complexity. Second, the communication complexity increases significantly, especially with secure multi-party computation (SMPC) based approaches. These additional costs pose a serious problem for DGA detection because real-time inference is required in most real-world settings. For the real-time classification of all NXDs that occur in a large real-world network, an inference latency of $405\mu s$ to $430\mu s$ is required according to [3], [4].

In this work, we implement different state-of-the-art DGA detection models in four generic privacy-preserving ML frameworks and investigate their real-world applicability. We comparatively evaluate the privacy-preserving implementa-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 833418. Simulations were performed with computing resources granted by RWTH Aachen University under project rwth0438.

tions and investigate how different model simplifications affect the models' accuracy in favor of lower inference latency and communication complexity in a trade-off study. We propose three model simplifications: reduction of the embedding dimension, removal of ReLU activation functions, and exchange of max with average pooling layers. Consequently, with this work we answer the question of whether privacy-preserving DGA detection can be realized with current tools in a real-world scenario.

## II. Related Work

In this section, we first present related work on state-of-the-art DGA detection classifiers. Subsequently, we discuss research efforts regarding privacy-preserving ML frameworks.

### A. DGA Detection Classifiers

Several approaches to DGA detection that leverage different amounts of data have been proposed in the past. These approaches can be divided into context-aware [5]–[10] and context-less approaches [3], [11]–[14]. The former use additional contextual information to improve the classification of domain names as either benign or malicious. The latter base their classification solely on information extracted from a single domain name and are therefore less resource-intensive and privacy-invasive compared to context-aware approaches.

In this work, we focus on context-less approaches, as they process less potentially privacy-critical data for classification. In addition, several studies have shown that context-less approaches are able to achieve state-of-the-art detection performance without relying on additional data [3], [11]–[13].

The proposed context-less ML classifiers can further be grouped according to feature-based (e.g., support vector machines (SVMs) or random forests (RFs) [11]) and featureless (deep learning) classifiers such as recurrent (RNNs) [12], convolutional (CNNs) [13], [14], and residual neural networks (ResNets) [3]. In previous studies, both types of classifiers have been contrasted and deep learning classifiers have been shown to outperform classical feature-based approaches to DGA detection [3], [12], [15], [16].

### B. Privacy-Preserving Machine Learning Frameworks

Research in the area of privacy-preserving ML has been rich in the past few years and several frameworks have been developed using different privacy-enhancing technologies.

The proposed ML frameworks typically only support a limited number of classifier types. Homomorphic encryption (HE) based approaches for privacy-preserving evaluation of decision trees and RFs are proposed in [17]–[20]. An approach to the private evaluation of SVMs is presented in [21]. Since deep learning classifiers outperform classical feature-based approaches in our use case, we are generally interested in privacy-preserving ML frameworks for neural networks. These frameworks can be categorized in HE-based [22]–[27], SMPC-based [28]–[34], functional encryption-based [35], and hybrid [36], [37] approaches.

In general, SMPC allows multiple parties to jointly evaluate a function without revealing their private inputs to said function. SMPC approaches can generally be categorized into approaches based on (arithmetic or boolean) secret sharing (e.g., [38]–[40]), garbled circuits (GC) [41], and approaches based on homomorphic encryption (e.g., [42], [43]). Some approaches to SMPC, such as [41], are restricted to a two-party setting, while others, such as [44], support multi-party protocols.

In the past years, a variety of different frameworks supporting the development and implementation of SMPC protocols have been proposed, which support one or more of the above approaches. ABY [45] is a framework for mixed-protocol secure two-party computation based on arithmetic sharing, boolean sharing, and Yao's GCs. Frameworks such as SPDZ [46] (restricted to linear functions) and function secret sharing (FSS) [47] support protocols based on secret sharing between parties. Both can be utilized for two or more parties. SecureNN [48] supports the implementation of secure three-party computation protocols for various neural network building blocks with support for both linear and non-linear functions.

HE schemes enable calculations to be evaluated on encrypted data without the need to decrypt them. While SMPC can be implemented through an HE scheme, in this paper we distinguish HE-based approaches from other SMPC-based approaches, as their application to privacy-preserving ML has significantly different effects on performance metrics. HE schemes are categorized by the operations they support. Additive and multiplicative HE schemes support addition and multiplication over encrypted data, respectively. Fully HE (FHE) schemes can support an unlimited number of addition and multiplication operations over encrypted data. Leveled HE schemes, on the other hand, only support a limited number of operations that are constrained to a fixed multiplicative depth [49].

There are two classic security settings for SMPC: semi-honest adversary model and malicious adversary model. In the semi-honest model, the adversary follows the protocol for computation still tries to learn additional sensitive information from the protocol transcript. In contrast, in the malicious model, the adversary deviates arbitrarily from the protocol. Most of the proposed privacy-preserving ML frameworks provide semi-honest security.

## III. Evaluation Setup

In this section, we present our evaluation setup including the considered CaaS setting, used data sources, selected privacy-preserving ML frameworks, and selected state-of-the-art classifiers.

### A. Classification-as-a-Service Setting

In this work, we generally consider a traditional CaaS setting with two parties: a *model owner* who holds the model and a *data owner* who holds the domain names.

TABLE I
OVERVIEW OF PRIVACY-PRESERVING ML FRAMEWORKS

| ML Framework | Parties | Approach | Protocol | Dense | LSTM | Conv1D | Max Pool 1D | Avg Pool 1D | ReLU | Sigmoid | Embedding |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PySyft [31] | 3 | SMPC | SPDZ, SecureNN, FSS | ✓ | ✗ | only 2D | only 2D | ✓ | ✓ | ✗ | ✗ |
| TF-Encrypted [32] | 3 | SMPC | SPDZ, SecureNN, ABY3 | ✓ | ✗ | only 2D | only 2D | only 2D | ✓ | ✓ | ✗ |
| MP2ML [36] | 2 | HE & SMPC | CKKS Scheme, ABY | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| SecureQ8 [34] | 2 or 3 | SMPC | OTSemi2k, OTSemiPrime, Replicated2k, ReplicatedPrime, SPDZ2k, Lowgear, PsReplicated2k, PsReplicatedPrime [39] | ✓ | ✗ | only 2D | only 2D | only 2D | ✓ | ✓ | ✗ |
| CrypTen [33] | 3 | SMPC | SPDZ, GMW | ✓ | ✗ | only 2D | only 2D | only 2D | ✓ | ✗ | ✗ |

When using SMPC, model owner and data owner secret share their respective inputs between their computation servers, such that no single party is able to reconstruct data. In a two-party setting, both the model owner and the data owner each run a computation server. However, honest majority SMPC protocols, as used in some of the proposed privacy-preserving ML frameworks, require an independent third party to preserve privacy. In a three-party setting, an additional computation server is operated by an independent third party, since neither the model owner nor the data owner can operate this server without violating the other party's privacy. In this work, we evaluate both, two and three-party settings with respect to the computation servers.

When using HE, the data owner encrypts a domain name to be classified with an HE scheme and transmits the encrypted data to the model owner. The model owner runs inference on the encrypted data and then transmits the encrypted result back to the data owner. Finally, the data owner decrypts the result to obtain the final prediction.

Both the application of SMPC and HE-based approaches to CaaS are associated with non-negligible costs compared to plaintext classifiers. Communication complexity as well as inference latency increase significantly, which is a serious problem for DGA detection because real-time inference is required in most real-world environments.

Thus it is questionable whether an inference latency of $405\mu s$ to $430\mu s$, that is required for real-time classification [3], [4], can be achieved by privacy-preserving DGA detection classifiers with current tools.

## B. Data Sources

We obtain data for our experiments from two data sources.

*1) Malicious Data: DGArchive:* We collect malicious labeled samples from DGArchive [50] which is a database containing DGA-generated domains for a multitude of different DGA families. From this source, we obtain approximately 93 million unique domain names generated by 88 different DGAs.

*2) Benign Data: University Network:* We receive benign domain names from the central DNS resolver of the campus network of RWTH Aachen University. The campus network incorporates several academic and administrative networks, networks from student residences, eduroam [51], and the network of the university hospital. From this source, we obtain a one-month recording of September 2019 consisting of approximately 26 million unique NXDs.

*3) Dataset:* We filter our benign data against all samples of DGArchive and remove all matches to clean our data as much as possible. Subsequently, we randomly draw at most 10k samples for each DGA in DGArchive. We follow the guidelines of [52] and select all available samples for DGAs for which less than 10k samples are known in order to increase the detection performance of a trained classifier even for weakly represented DGAs. In addition, we randomly select the same amount of benign samples as we drew malicious samples. Thereby, we create a balanced dataset that consists of approximately one million samples, 500k generated by 88 different DGAs and 500k benign samples.

We use this dataset in all following experiments to train and evaluate privacy-preserving DGA detection classifiers. To this end, we split 80% of the samples for training a suitable classifier and 20% for subsequent testing. We always choose the best classifier within 50 training epochs for evaluation.

## C. Selected Privacy-Preserving ML Frameworks

In this section, we present the selected privacy-preserving ML frameworks that we use for our comparative evaluation. We restrict our selection to open-source frameworks that are implemented on top of TensorFlow [53] or PyTorch [54]. By this selection method, we minimize the variance of performance metrics and are able to structure and apply a uniform evaluation framework.

In total, we investigate five ML frameworks in more detail. We present the frameworks including the required number of computation parties, supported protocols, and supported neural network building blocks in Table I.

The frameworks support two-party or three-party settings, which is not a restriction caused by supported SMPC protocols but rather a design choice made by the developers. A traditional CaaS setting includes only two parties. However, we also cover three-party frameworks to enable a performance comparison of two-party and three-party approaches.

*1) PySyft:* We select PySyft [31] which is built on top of PyTorch. It uses SPDZ for linear functions and SecureNN or FSS for the evaluation of non-linear functions. We use FSS to evaluate non-linear functions in PySyft in order to increase the variety of protocols, as it is only supported by this framework.

*2) TFE:* We select TF-Encrypted (TFE) [32] which is implemented on top of TensorFlow. It uses Pond, a three-party implementation of the SPDZ protocol which only supports linear functions. For non-linear functions, it makes use of ABY3 (a three-party variant of ABY) or SecureNN. For our evaluation, we use SecureNN to evaluate non-linear functions in TFE.

*3) MP2ML:* We select MP2ML, a hybrid framework that uses the CKKS HE [55] scheme to evaluate linear functions and ABY to support non-linear layers.

*4) SecureQ8:* SecureQ8 [34] is an SMPC framework that implements eight out of 30 protocols proposed by the MP-SPDZ library [39]. Four of them provide semi-honest security while four others provide malicious security. We exclude the protocols that provide malicious security from our evaluation as most other frameworks do not provide security against this threat model. We select the remaining four protocols for our evaluation and denote SecureQ8 with OTSemi2k, OTSemiPrime, Replicated2k, and ReplicatedPrime by Q8-0 to Q8-3, respectively. During the time of experiments, there was an issue with parsing quantized models preventing the correct loading of model weights. We experimented with different quantization methods provided by multiple TensorFlow versions, however, none of the resulting quantized weights were parsed correctly by SecureQ8. Hence, we run SecureQ8 with randomly initialized weights. A comparison with a trained plaintext classifier with the same weights is therefore not possible. While this prevents the measurement of the approximation error caused by this framework, we are still able to evaluate it in the context of our trade-off study (cf. Section IV).

*5) CrypTen:* We do not choose CrypTen [33] for our evaluation as it only supports Amazon Web Service (AWS) machines for evaluating remote settings.

In total, we thus selected seven privacy-preserving ML framework configurations (PySyft, TF-E, MP2ML, Q8-0, Q8-1, Q8-2, Q8-3) for further evaluation.

We set the internal parameters of each framework to the default and recommended values given in the respective literature. For our evaluation, we operate each framework in its respective remote setting and each entity involved in the CaaS scenario in its own process to ensure that network communication is a necessity. However, all entities are executed on the same machine and use the loopback interface for communication, which minimizes potential network delay. Note that the reported inference latency for each framework can therefore be considerably longer in a real-world environment.

### *D. Selected State-of-the-Art Classifiers*

In this section, we present the state-of-the-art classifiers that we implement in the selected privacy-preserving ML frameworks. Since the deep learning classifiers outperform feature-based approaches for DGA detection, we choose three deep learning classifiers that differ in their architecture. However, we also adapt one feature-based approach for comparison.

For classification, all deep learning models start by processing integer encoded domain names using an embedding layer that includes semantics into the encoding. The embedded domain names are then processed by different hidden layers depending on the utilized model. Below we give a brief overview of the different models. More detailed information can be found in the respective literature [3], [13], [56].

*1) Inline:* Yu et al. [56] proposed a model based on a one-dimensional convolutional layer with 1,000 filters for DGA binary classification.

*2) NYU:* The authors of [13] also proposed a CNN-based model, but it consists of two stacked one-dimensional convolutional layers with 128 filters.

*3) ResNet:* Drichel et al. [3] proposed a ResNet-based classifier consisting of a single residual block that introduces skip connections between convolutional layers to counteract the vanishing gradient problem. Likewise to the NYU model, the ResNet model uses two one-dimensional convolutional layers with 128 filters within the residual.

*4) Feature-Based Approach:* FANCI [11] is a feature-based DGA detection approach that implements a random forest (RF) and a support vector machine (SVM) classifier. Both classifier extract for each domain name to be classified, 21 features that are categorized into linguistic, statistical, and structural features. As all selected privacy-preserving ML frameworks are only applicable to neural network classifiers, we create a deep learning classifier out of FANCI by passing extracted features to a dense layer consisting of 32 neurons with ReLU activation function. Similar to the deep learning models, the dense layer's output is then passed to a single output neuron with sigmoid activation to perform the final logistic regression for binary classification.

*5) Model Modifications:* One-dimensional convolutional layers as used by the three deep learning models are not supported in PySyft, TF-E, and SecureQ8. Hence, we construct two-dimensional variants of the Inline, NYU, and ResNet models by replacing all one-dimensional convolutional layers with two-dimensional convolutions. For compatibility, we add an additional dimension to the input data which we pad with zeros. Provided sufficient training, these variants achieve the same accuracy as the unmodified models.

During testing, PySyft, SecureQ8, and MP2ML frequently crashed, timed out, or refused to run due to the complexity of the models. Therefore, additional modifications are necessary to enable an evaluation of these frameworks. To do this, we reduce layer dimensionality to eliminate these stability problems while preserving model architectures as much as possible. In detail, we reduce the dimension of the convolutional layer for the Inline model from 1,000 to 100. Further, we reduce the dimension of the convolutional layers included in the NYU and ResNet model from 128 to 32. For the ResNet model, we additionally reduce the kernel size of the max pooling layer from 4 to 3. We determined these values by iteratively reducing the dimensionality of the layers until all models run stable in all frameworks. We calculate the potential decline in classification performance caused by the model changes by training unmodified and modified models with the data set described in Section III-B3. We present the achieved accuracies (ACC), true positive rates (TPRs), and false positive rates (FPRs) in Table II.

The resulting loss of accuracy is insignificant, since the accuracy of all three modified models decreases by less than

TABLE II
ACCURACY IMPACT OF MODEL MODIFICATIONS

| Classifier | Unmodified | Modified |
|---|---|---|
| | ACC / TPR / FPR [%] | ACC / TPR / FPR [%] |
| Inline | 99.84 / 99.83 / 0.15 | 99.77 / 99.76 / 0.22 |
| NYU | 99.81 / 99.78 / 0.16 | 99.81 / 99.80 / 0.18 |
| ResNet | 99.86 / 99.85 / 0.13 | 99.85 / 99.82 / 0.12 |

0.07% in accuracy, if it decreases at all. In the following, we always refer to the modified models in our experiments.

*6) Unsupported Operations:* While we can work around the problem described above with unsupported one-dimensional convolutional layers, there are additional limitations imposed by the frameworks.

In detail, embedding layers are not supported by any of the frameworks (cf. Table I) and feature extraction as used in the FANCI-based model can only be performed on plaintext domains. Further, sigmoid functions are not supported by all frameworks. Therefore, we exclude the execution of embedding layers, feature extraction, and sigmoid activation functions from the private inference process.

This implies that the data owner must perform either feature extraction or embed domain names before secret sharing or data encryption, and must apply the sigmoid function after the final inference result has been received and decrypted. While this setup enables an unbiased evaluation of all frameworks, it leaks the embedding layers and the feature extraction process to the data owner in a real-world setting.

Moreover, TF-E and SecureQ8 do not support custom layers such as the residual layer of the ResNet model. Hence, we only evaluate the ResNet model in PySyft and MP2ML.

Finally, due to the lack of support for long short-term memory (LSTM) layers in all frameworks, we exclude the well-known RNN-based model [12] from our evaluation.

## E. Hardware

Our test system consists of an AMD Ryzen 5 3600 CPU with six cores@3.6GHz and 16GB of RAM. We execute each model on the CPU because the frameworks do not support GPU computation. Additionally, we run each framework in a separate Docker container to ensure a similar overhead.

## IV. EVALUATION

In this section, we first provide an overview of the different evaluations and subsequently present the evaluation results.

### A. Evaluation Overview

In a plaintext inference setting, the performance impact of parameters, such as on the inference latency, is often neglected because they are usually tuned for the best accuracy. However, in a privacy-preserving inference setting, inference latency and communication complexity determine the feasibility of a model. Thus, foregoing a small accuracy penalty for a significant reduction in inference latency or communication complexity can ensure the feasibility of a privacy-preserving classifier. In this work, we investigate the impact of the embedding dimension, ReLU activation functions, and max pooling layers on the inference latency, communication complexity, and the accuracy.

First, we present the inference latency (i.e., the time delay between starting the inference and computing the prediction) and the communication complexity (i.e., the amount of data that is transmitted between data and model owner) for each of the various models and frameworks in Section IV-B.

Subsequently, we analyze the effect of the embedding dimension, ReLU activation functions, and max pooling layers on the inference latency and communication complexity in Section IV-C1, IV-C2, and IV-C3, respectively. We present the cumulative relative improvement in inference latency and communication complexity after applying all three proposed model simplifications in Section IV-C4.

In Section IV-D, we present the impact of all model simplifications on the classification performance (ACC, TPR, FPR).

Finally, in Section IV-E, we measure the relative approximation error between plaintext predictions and corresponding private predictions caused by each framework.

In our evaluation, we consider four models and seven privacy-preserving ML framework configurations. In addition, there are up to 16 variants per model through a combination of four different embedding dimensions, two options for activation functions, and two options for pooling layers. However, not all 16 variants are applicable to all four models, as some models do not contain any embedding or pooling layers, which leads to a total of 298 different privacy-preserving inference scenarios.

Due to this amount of scenarios and high inference latencies (of up to 13 min/sample), running every scenario on the complete test set is infeasible. Hence, we limit the amount of test samples depending on the inference latency of each scenario. Specifically, we randomly draw 1,000 test samples for frameworks running in a three-party setting and 100 samples for two-party settings. Note, reduced sample sizes are only used for privacy-preserving performance metrics, i.e., inference latency, communication complexity, and approximation error. The accuracy evaluation (Section IV-D) is performed on the entire dataset (cf. Section III-B3).

Moreover, privacy-preserving classifiers operate on secret shared or encrypted data, i.e., essentially random values. Thus, the inference computation process is independent of plaintext domain names, resulting in little variance in the performance metrics. Consequently, a smaller sample size also enables a meaningful comparative evaluation.

Lastly, we conduct all evaluations with a batch size of one. Note that a larger batch size would result in a significantly higher throughput as there is a constant overhead.

### B. Inference Latency & Communication Complexity

In Table III, we present the inference latency and communication complexity for each framework before applying any model simplifications (i.e., the embedding dimension is not

TABLE III
INFERENCE LATENCY IN (S) / COMMUNICATION COMPLEXITY IN (KB) FOR EACH MODEL AND FRAMEWORK

| Classifier | Three-Party Setting | | | | Two-Party Setting | | |
|---|---|---|---|---|---|---|---|
| | PySyft | TF-E | Q8-2 | Q8-3 | MP2ML | Q8-0 | Q8-1 |
| Inline | 2.282 / 152,721 | 1.341 / 158,080 | 2.071 / 93,363 | 4.209 / 246,473 | 296.641 / 36,886,810 | 98.228 / 60,300,025 | 208.422 / 7,980,211 |
| NYU | 6.942 / 315,674 | 3.679 / 295,319 | 2.480 / 176,717 | 10.288 / 841,378 | 191.371 / 34,614,955 | 374.646 / 234,501,961 | 793.289 / 30,842,328 |
| ResNet | 9.840 / 431,917 | - / - | - / - | - / - | 174.789 / 32,983,456 | - / - | - / - |
| FANCI | 0.099 / 159 | 0.141 / 288 | 0.598 / 521 | 0.609 / 966 | 0.135 / 17,659 | 0.649 / 29,325 | 0.863 / 13,454 |

reduced from 128, ReLU activations are not removed, and max pooling layers are not replaced).

All frameworks and models were evaluated on the test system (cf. Section III-E), with the exception of Inline, NYU, and ResNet in the MP2ML framework. The reason for this is the high RAM requirement of MP2ML exceeding the 16GB RAM of the test system. To enable a comparison of this framework, we ran the three models on a compute cluster with 150GB of RAM. However, lower CPU clock speeds and different numbers of threads result in a significant difference in inference latency. To enable a meaningful comparison, we therefore interpolated the inference latencies for these three models in the MP2ML framework. While this introduces a non-negligible error, it affects all MP2ML settings equally, such that only the absolute inference latency is affected. The communication complexity is not affected by this and thus accurate.

The inference latency for FANCI is below one second for all two- and three-party frameworks due to model simplicity. For Inline, NYU, and ResNet, there is a significant performance difference between two- and three-party frameworks. All three-party frameworks achieve an inference latency of less than eleven seconds for all models, while the two-party frameworks perform significantly worse and require up to 13 min/inference. Moreover, MP2ML's inference latency for the Inline model is significantly higher than the latency of both two-party SMPC-based approaches, while MP2ML performs comparably better for NYU and ResNet. The reason for this is the high input dimension of the first linear layer in the Inline model which poses a significantly higher latency impact in MP2ML's HE-based approach compared to SecureQ8's SMPC-based approaches.

The communication complexity in the two-party setting is also significantly higher compared to the three-party setting. The communication complexity for the NYU model is the highest and ranges for a single data item from 841MB in a three-party setting to 234GB in a two-party setting. FANCI on the other hand results in the lowest communication complexity with only 159kB in PySyft and 29MB in Q8-0.

Generally, the communication complexity of HE-based approaches is expected to be significantly lower compared to SMPC-based approaches [36]. However, this is not the case when comparing MP2ML with Q8-1. The reason for this is that the embedding layers are evaluated before the inference process, which leads to high dimensional input data for privacy-preserving classifiers and thus to large transmitted ciphertexts. The encoding in the CKKS HE scheme used by MP2ML requires more space than the corresponding secret shares in an SMPC setting. In addition, the implemented CKKS scheme only supports linear operations. For this reason, MP2ML uses a two-party SMPC protocol to evaluate non-linear layers, which increases the communication complexity even further.

Overall, the communication complexity is high, especially in the two-party setting. In addition, even if we completely ignore the natural network delay, the inference latencies exceed the limits necessary for real-time detection ($405\mu s$ to $430\mu s$). Thus, additional measures are required to enable the feasibility of privacy-preserving DGA detection classifiers.

## C. Model Simplification Analysis

In this section, we investigate the impact of the embedding dimension, ReLU activation functions, and max pooling layers on the performance of a classifier.

*1) Embedding Dimension:* We run every model that contains an embedding layer with an embedding dimension of 128, 64, 32, and 16. Higher embedding dimensions may result in better accuracy, but higher data dimensionality also increases inference latency and communication complexity.

In the upper part of Table IV, we display the relative performance impact of an embedding dimension reduction from 128 to 16. The ResNet model profits most from the embedding dimension reduction. The inference latency reduces by 64% and the communication complexity by 63% in a three-party setting and by 86% and 79% in a two-party setting, respectively. The reduction of the embedding dimension reduces the input dimension of subsequent layers which saves additional computational cost. The improvements are significantly higher in a two-party setting than in a three-party setting as the former requires more communication.

*2) ReLU Activation Functions:* The ReLU function is non-linear and therefore computationally expensive to evaluate privately. Hence, we investigate the performance impact of replacing ReLU activations with the identity function $h(x) \mapsto x$.

We display the relative performance impact of removing ReLU activations in the second row of Table IV. In the two-party setting, there are almost no improvements except for the communication complexity for FANCI in Q8-0 and for all models in MP2ML. In MP2ML, the communication complexity reduces by up to 55%. As mentioned earlier, MP2ML uses a two-party SMPC protocol to evaluate non-linear layers. By removing ReLU activations, fewer non-linear calculations have to be performed and therefore fewer ciphertexts have to be exchanged, which leads to a significant reduction in communication complexity in MP2ML.

TABLE IV
MODEL SIMPLIFICATION ANALYSIS

| Simplification | Classifier | Three-Party Setting | | | | Two-Party Setting | | |
|---|---|---|---|---|---|---|---|---|
| | | **PySyft** | **TF-E** | **Q8-2** | **Q8-3** | **MP2ML** | **Q8-0** | **Q8-1** |
| **Embedding Dimension Reduction** | Inline | 0.08 / 0.03 | 0.06 / 0.03 | 0.06 / 0.01 | 0.16 / 0.01 | 0.61 / 0.40 | 0.67 / 0.68 | 0.64 / 0.63 |
| | NYU | 0.16 / 0.16 | 0.18 / 0.04 | 0.11 / 0.01 | 0.27 / 0.01 | 0.66 / 0.43 | 0.76 / 0.77 | 0.72 / 0.72 |
| | ResNet | 0.64 / 0.63 | - / - | - / - | - / - | 0.86 / 0.79 | - / - | - / - |
| **ReLU Removal** | Inline | 0.61 / 0.43 | 0.75 / 0.45 | 0.01 / 0.00 | -0.01 / 0.00 | 0.04 / 0.54 | 0.00 / 0.00 | 0.00 / 0.00 |
| | NYU | 0.45 / 0.46 | 0.34 / 0.53 | 0.00 / 0.00 | 0.00 / 0.00 | 0.03 / 0.28 | 0.00 / 0.00 | 0.00 / 0.00 |
| | ResNet | 0.22 / 0.24 | - / - | - / - | - / - | 0.01 / 0.55 | - / - | - / - |
| | FANCI | 0.71 / 0.63 | 0.71 / 0.68 | 0.00 / 0.02 | 0.00 / 0.10 | 0.08 / 0.48 | 0.03 / 0.37 | 0.00 / 0.02 |
| **Pooling Layer Exchange** | NYU | 0.34 / 0.34 | 0.48 / 0.40 | 0.00 / -0.18 | 0.19 / 0.30 | -0.59 / -0.35 | 0.05 / 0.05 | 0.06 / 0.06 |
| | ResNet | 0.46 / 0.51 | - / - | - / - | - / - | -0.35 / 0.23 | - / - | - / - |
| **Cumulative Improvements** | Inline | 0.68 / 0.46 | 0.80 / 0.48 | 0.05 / 0.01 | 0.16 / 0.01 | 0.68 / 0.94 | 0.67 / 0.68 | 0.64 / 0.64 |
| | NYU | 0.95 / 0.96 | 0.87 / 0.97 | 0.14 / -0.17 | 0.45 / 0.30 | 0.73 / 0.89 | 0.81 / 0.82 | 0.78 / 0.78 |
| | ResNet | 0.93 / 0.94 | - / - | - / - | - / - | 0.84 / 0.92 | - / - | - / - |
| | FANCI | 0.71 / 0.63 | 0.71 / 0.68 | 0.00 / 0.02 | 0.00 / 0.10 | 0.08 / 0.48 | 0.03 / 0.37 | 0.00 / 0.02 |

In the three-party setting, PySyft and TF-E are heavily affected by the removal of ReLU activations leading to an improvement of up to 75% in inference latency and 68% in communication complexity. In contrast, the Q8 approaches are almost unaffected.

*3) Pooling Layers:* Similar to the ReLU activation function, the max pooling operation, which is used by NYU and ResNet, is non-linear and therefore computationally expensive to evaluate. Evaluating a linear average pooling layer can instead be less computationally expensive. Here, we investigate the performance impact of replacing max pooling layers with average pooling layers.

We present the relative performance impact in the third row of Table IV. In the three-party setting, we achieve improvements for PySyft, TF-E, and Q8-3 of up to 46% and 51% in inference latency and communication complexity, respectively. For Q8-2, however, the communication complexity increases by 18%, suggesting that the linear operations in average pooling layers require more communication than the comparisons in max pooling layers in this particular protocol.

In the two-party setting, there are slight improvements for the two Q8-based approaches, but in MP2ML there is an enormous deterioration in terms of inference latency and NYU's communication complexity. The reason for this is that average pooling layers increase the required multiplicative depth of the models. This results in an increase in ciphertext size, which offsets the decreased communication complexity caused by eliminating the need for SMPC-based evaluation of max pooling layers. The different effects on the performance of NYU and ResNet are due to the different number of max pooling layers used (2 vs. 1), differences in kernel sizes (2 vs. 3), and the difference in the multiplicative depth caused by the average pooling layers (8 vs. 6).

While we obtained less promising results for MP2ML, in theory, the exchange of the pooling layers could pay off with a smaller input size.

*4) Cumulative Performance Improvements:* In the bottom of Table IV, we present the cumulative relative improvements that are achieved by combining the proposed simplifications.

In the two-party setting, we achieve significant improvements for all frameworks and models of up to 84% in inference latency and 94% in communication complexity. The FANCI-based neural network classifier is the fastest due to its encoding by feature extraction and its model simplicity. It requires approximately 0.124s (originally 0.135s) and 9MB (originally 18MB) of communication for a single data item in MP2ML. The next fastest model is ResNet in MP2ML which requires approximately 28s (originally 175s) and 2.7GB (originally 32.9GB) of communication for private inference.

In the three-party setting, the inference latency is reduced by up to 95%, while the reduction in communication complexity is as high as 97% for PySyft and TF-E. Here, FANCI achieves an inference latency of below one second in all frameworks. In TF-E, it requires approximately 0.029s and 58kB for private inference. The Inline model improves slightly in Q8-2. The NYU model only improves in inference latency and deteriorates in communication complexity caused by the pooling layer exchange. In Q8-3, there are smaller improvements for Inline while NYU profits significantly. In PySyft and TF-E, Inline, NYU, and ResNet achieve an inference latency of below one second, while the inference latency in the Q8 approaches ranges from 2 - 6 seconds. The fastest model after FANCI is Inline with only 0.27s latency in TF-E and requires 8.3MB communication per inference. The NYU model requires 0.47s per inference but has a lower communication complexity of only 0.8MB.

Despite these significant improvements, the required inference latency of $405\mu s$ to $430\mu s$, that is necessary for privacy-preserving real-time classification, is still not achieved.

## D. Classification Performance

While the relative improvements in communication complexity and inference latency are enormous, the impact of the proposed model simplifications on classification performance remains to be determined.

In Table V, we present the classification performance of all models to assess the accuracy penalty caused by the applied model simplifications. In the following, we mainly use the

TABLE V
CLASSIFICATION PERFORMANCE: ACC/TPR/FPR [%]

| Classifier | Pool | ReLU | Embedding Dimension | | | |
|---|---|---|---|---|---|---|
| | | | 16 | 32 | 64 | 128 |
| Inline | - | yes | 99.74 / 99.96 / 0.48 | 99.74 / 99.98 / 0.50 | 99.76 / 99.96 / 0.44 | 99.77 / 99.97 / 0.43 |
| | - | no | 99.66 / 99.97 / 0.65 | 99.65 / 99.96 / 0.66 | 99.69 / 99.98 / 0.60 | 99.66 / 99.95 / 0.63 |
| NYU | Max | yes | 99.74 / 99.93 / 0.45 | 99.77 / 99.96 / 0.42 | 99.79 / 99.95 / 0.37 | 99.81 / 99.98 / 0.36 |
| | | no | 99.76 / 99.91 / 0.39 | 99.80 / 99.96 / 0.36 | 99.80 / 99.98 / 0.38 | 99.83 / 99.98 / 0.32 |
| | Avg | yes | 99.72 / 99.90 / 0.46 | 99.75 / 99.93 / 0.43 | 99.71 / 99.90 / 0.48 | 99.75 / 99.94 / 0.44 |
| | | no | 99.65 / 99.80 / 0.50 | 99.61 / 99.81 / 0.59 | 99.64 / 99.98 / 0.70 | 99.68 / 99.89 / 0.53 |
| ResNet | Max | yes | 99.82 / 99.93 / 0.29 | 99.84 / 99.98 / 0.30 | 99.84 / 99.94 / 0.26 | 99.85 / 99.93 / 0.23 |
| | | no | 99.79 / 99.85 / 0.27 | 99.83 / 99.95 / 0.29 | 99.84 / 99.92 / 0.24 | 99.84 / 99.95 / 0.27 |
| | Avg | yes | 99.80 / 99.90 / 0.30 | 99.82 / 99.94 / 0.30 | 99.84 / 99.96 / 0.28 | 99.83 / 99.98 / 0.32 |
| | | no | 99.68 / 99.99 / 0.63 | 99.66 / 99.64 / 0.32 | 99.72 / 99.86 / 0.42 | 99.73 / 99.89 / 0.43 |
| FANCI | - | yes | 98.58 / 99.24 / 2.08 | | | |
| | - | no | 98.02 / 97.87 / 1.83 | | | |

FPR as a proxy to determine the possible loss in classification performance as a low FPR is the most important attribute of a suitable classifier. Hence, we exclude FANCI from further discussion due to its high FPR (cf. Table V).

Regarding the reduction of the embedding dimension by its own, a clear tendency towards a deteriorated FPR can be observed. Removing ReLU activations but keeping the max pooling layers, results in a significant loss of FPR for Inline. The FPRs of NYU and ResNet, however, change only slightly and even improve in some settings. We attribute the significant drop in Inline's FPR to the violation of a requirement of the universal approximation theorem [57] which states that multilayer neural networks are able to approximate any continuous function to an arbitrary degree of accuracy, provided an appropriate architecture and activation function [57]–[59]. Removing ReLU activations cuts all non-linearity from the Inline model, resulting in a significant loss of approximation capability [60], thus causing a notable FPR penalty. NYU and ResNet are not affected here as both models contain additional non-linearity in the max pooling layers.

Replacing max pooling layers with average pooling layers results in only a negligible change in the FPR. However, combining average pooling layers with the removal of ReLU activations removes any non-linearity from NYU and ResNet and can yield to an FPR penalty similar to the removal of ReLU activations in the Inline model.

### E. Approximation Error

In Table VI, we present the relative errors between plaintext predictions and their corresponding private predictions to provide a notion of the errors caused by each framework. We average the approximation error over the embedding dimensions and pooling layers as both parameters have no effect on the approximation error.

The approximation error of PySyft and TF-E is negligible, such that a privacy-preserving prediction is essentially equal to a plaintext prediction. In MP2ML, the approximation error is significantly increased if a model contains ReLU activations. This is not intuitive as ReLU activations discard any values below zero, thus eliminating potential errors in the intermediate results. We attribute this behavior to the MP2ML's implementation of the ReLU activation that approximates the exact ReLU function.

TABLE VI
EVALUATION: APPROXIMATION ERRORS

| Model | ReLU | PySyft | TF-E | MP2ML |
|---|---|---|---|---|
| Inline | yes | 0.08% | 0.26% | 16.38% |
| | no | 0.15% | 1.49% | 0.01% |
| NYU | yes | 0.01% | 0.01% | 16.12% |
| | no | 0.01% | 0.02% | 0.02% |
| ResNet | yes | 0.01% | - | 14.11% |
| | no | 0.01% | - | 0.24% |
| FANCI | yes | 0.02% | 0.02% | 14.30% |
| | no | 0.02% | 0.02% | 0.01% |

The error caused by MP2ML is large enough to potentially affect a classifier's prediction, depending on the decision threshold and the classifier's certainty. However, the authors of MP2ML conducted an evaluation which showed that the majority of predictions in a binary classification case are not affected by this [36].

## V. CONCLUSION

Our evaluation showed that implementing state-of-the-art DGA detection models in existing privacy-preserving frameworks is not sufficient to construct feasible privacy-preserving classifiers. In this work, we proposed model simplifications that achieve a reduction in inference latency of up to 95%, and up to 97% in communication complexity while causing an accuracy penalty of less than 0.17%. In particular, the accumulated reduction is mostly consistent between two- and three-party settings, and suggests that the proposed simplifications generalize well to other DGA detection models and privacy-preserving frameworks. For the real-world application, framework and model-specific optimizations are still required, and it is suggested to keep a source of non-linearity in a model.

High dimensional data limits the feasibility of privacy-preserving classifiers. Since the embedding layers cannot be executed privately in the selected frameworks, large secret shares or ciphertexts have to be exchanged, especially in MP2ML. The encoding of the developed FANCI-based neural network classifier by feature extraction, leads to smaller data that needs to be exchanged. In addition, the low complexity of the model, leads to inference latencies under one second even in the two-party setting. However, this model has too high an FPR to operate in a real-world scenario. Increasing the model's complexity could lead to a lower FPR but also to a higher inference latency. Removing the embedding layer of the state-of-the-art deep learning classifiers is not an option as prior experiments showed a significant loss in ACC and FPR. However, private evaluation of embedding layers as in PrivFT [25] is technically possible but not implemented in any of the investigated frameworks. This would however increase the computational complexity further, possibly resulting in a significant increase in inference latency.

To classify all NXDs that occur in a large real-world network, an inference latency of $405\mu s$ to $430\mu s$ is required [3], [4], which is not achieved by any of the classifiers. The classifiers in the two-party setting are particularly slow. The fastest model that still achieves an appropriate FPR takes 28s per domain name. In the three-party setting, all of the classifiers take less than a second, but it is not apparent who the third party might be as CaaS is usually considered a two-party setting. Using a larger batch size and DNS caching would make the classifiers more viable. On the other hand, we ignored possible network delay in our analysis, which will increase the inference latency considerably, as huge amounts of data are exchanged.

All in all, feasibility of privacy-preserving DGA detection is questionable due to a combination of high computational complexity and lack of real-world applicability. Future work is needed to improve DGA models and frameworks, for example by enabling private evaluation of embedding layers or by supporting one-dimensional convolutional layers, the absence of which has a significant impact on all SMPC-based frameworks.

## References

[1] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction," *Computational and structural biotechnology*, vol. 13, 2015.

[2] H. C. Tanuwidjaja, R. Choi, S. Baek, and K. Kim, "Privacy-preserving deep learning on machine learning as a service - a comprehensive survey," *IEEE*, vol. 8, 2020.

[3] A. Drichel, U. Meyer, S. Schüppen, and D. Teubert, "Analyzing the real-world applicability of DGA classifiers," in *Conference on Availability, Reliability and Security*. ACM, 2020.

[4] A. Drichel, N. Faerber, and U. Meyer, "First step towards explainable dga multiclass classification," in *Conference on Availability, Reliability and Security*. ACM, 2021.

[5] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of DGA-based malware," in *USENIX Security Symposium*, 2012.

[6] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A passive DNS analysis service to detect and report malicious domains," in *Transactions on Information and System Security*. ACM, 2014.

[7] M. Grill, I. Nikolaev, V. Valeros, and M. Rehak, "Detecting DGA malware using netflow," in *IFIP/IEEE Integrated Network Management*, 2015.

[8] S. Yadav and A. L. N. Reddy, "Winning with dns failures: Strategies for faster botnet detection," in *Security and Privacy in Communication Systems*. Springer, 2011.

[9] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: DGA-based botnet tracking and intelligence," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2014.

[10] Y. Shi, G. Chen, and J. Li, "Malicious domain name detection based on extreme machine learning," in *Neural Processing Letters*, vol. 48, no. 3, 2018.

[11] S. Schüppen, D. Teubert, P. Herrmann, and U. Meyer, "FANCI: Feature-based automated nxdomain classification and intelligence," in *USENIX Security Symposium*, 2018.

[12] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," in *arXiv:1611.00791*, 2016.

[13] B. Yu, J. Pan, J. Hu, A. Nascimento, and M. De Cock, "Character level based detection of DGA domain names," in *International Joint Conference on Neural Networks*. IEEE, 2018.

[14] J. Saxe and K. Berlin, "eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys," in *arXiv:1702.08568*, 2017.

[15] J. Peck, C. Nie, R. Sivaguru, C. Grumer, F. Olumofin, B. Yu, A. Nascimento, and M. De Cock, "Charbot: A simple and effective method for evading dga classifiers," *IEEE Access*, vol. 7, 2019.

[16] J. Spooren, D. Preuveneers, L. Desmet, P. Janssen, and W. Joosen, "Detection of algorithmically generated domain names used by botnets: A dual arms race," in *Symposium On Applied Computing*. ACM, 2019.

[17] M. Joye and F. Salehi, "Private yet efficient decision tree evaluation," in *Data and Applications Security and Privacy XXXII*. Springer, 2018.

[18] D. J. Wu, T. Feng, M. Naehrig, and K. Lauter, "Privately evaluating decision trees and random forests," *Privacy Enhancing Technologies*, vol. 2016, no. 4, 2016.

[19] A. Tueno, Y. Boev, and F. Kerschbaum, "Non-interactive private decision tree evaluation," in *Data and Applications Security and Privacy XXXIV*. Springer, 2020.

[20] A. Akavia, M. Leibovich, Y. S. Resheff, R. Ron, M. Shahar, and M. Vald, "Privacy-preserving decision trees training and prediction." in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2020.

[21] T. Maekawa, T. Nakachi, S. Shiota, and H. Kiya, "Privacy-preserving svm computing by using random unitary transformation," in *Intelligent Signal Processing and Communication Systems*, 2018.

[22] T. van Elsloo, G. Patrini, and H. Ivey-Law, "Sealion: A framework for neural network inference on encrypted data," *arXiv:1904.12840*, 2019.

[23] A. Vizitiu, C. I. Niţă, A. Puiu, C. Suciu, and L. M. Itu, "Applying deep neural networks over homomorphic encrypted medical data," *Computational and mathematical methods in medicine*, vol. 2020, 2020.

[24] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, vol. 48. PMLR, 2016.

[25] A. A. Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung, "Privft: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, 2020.

[26] Q. Lou and L. Jiang, "She: A fast and accurate privacy-preserving deep neural network via leveled tfhe and logarithmic data representation," *arXiv:1906.00148*, 2019.

[27] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "Ngraph-he: A graph compiler for deep learning on homomorphically encrypted data," in *International Conference on Computing Frontiers*. ACM, 2019.

[28] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Computer and Communications Security*. ACM, 2017.

[29] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Design Automation Conference*. ACM, 2018.

[30] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, “Chameleon: A hybrid secure computation framework for machine learning applications,” in *Asia Conference on Computer and Communications Security*. ACM, 2018.

[31] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, “A generic framework for privacy preserving deep learning,” *arXiv:1811.04017*, 2018.

[32] M. Dahl, J. Mancuso, Y. Dupis, B. Decoste, M. Giraud, I. Livingstone, J. Patriquin, and G. Uhma, “Private machine learning in tensorflow using secure computation,” *arXiv:1810.08130*, 2018.

[33] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, “Crypten: Secure multi-party computation meets machine learning,” in *NeurIPS Workshop on Privacy-Preserving Machine Learning*, 2020.

[34] A. Dalskov, D. Escudero, and M. Keller, “Secure evaluation of quantized neural networks,” *Privacy Enhancing Technologies*, vol. 2020, no. 4, 2020.

[35] R. Xu, J. B. Joshi, and C. Li, “Cryptonn: Training neural networks over encrypted data,” in *International Conference on Distributed Computing Systems*, 2019.

[36] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, “Mp2ml: A mixed-protocol machine learning framework for private inference,” in *Availability, Reliability and Security*. ACM, 2020.

[37] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, “Delphi: A cryptographic inference service for neural networks,” in *USENIX Security Symposium*, 2020.

[38] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in *the Twentieth Annual ACM Symposium on Theory of Computing*. ACM, 1988.

[39] M. Keller, “Mp-spdz: A versatile framework for multi-party computation,” in *Computer and Communications Security*. ACM, 2020.

[40] R. Cramer, I. Damgård, and U. Maurer, “General secure multi-party computation from any linear secret-sharing scheme,” in *Advances in Cryptology — EUROCRYPT*. Springer, 2000.

[41] A. C.-C. Yao, “How to generate and exchange secrets,” in *Symposium on Foundations of Computer Science*, 1986.

[42] R. Cramer, I. Damgård, and J. B. Nielsen, “Multiparty computation from threshold homomorphic encryption,” in *Advances in Cryptology — EUROCRYPT*. Springer, 2001.

[43] C. Gentry, S. Halevi, and N. P. Smart, “Homomorphic evaluation of the aes circuit,” in *Advances in Cryptology – CRYPTO*. Springer, 2012.

[44] S. Micali, O. Goldreich, and A. Wigderson, “How to play any mental game,” in *Symposium on Theory of Computing*. ACM, 1987.

[45] D. Demmler, T. Schneider, and M. Zohner, “ABY - a framework for efficient mixed-protocol secure two-party computation,” in *Network and Distributed System Security Symposium*. Internet Society, 2015.

[46] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *Advances in Cryptology – CRYPTO*. Springer, 2012.

[47] E. Boyle, N. Gilboa, and Y. Ishai, “Function secret sharing,” in *Advances in Cryptology - EUROCRYPT*. Springer, 2015.

[48] S. Wagh, D. Gupta, and N. Chandran, “Securenn: 3-party secure computation for neural network training,” *Privacy Enhancing Technologies*, vol. 2019, no. 3, 2019.

[49] J. Liu, S. Mesnager, and L. Chen, “Partially homomorphic encryption schemes over finite fields,” in *Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016.

[50] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, “A comprehensive measurement study of domain generating malware,” in *USENIX Security Symposium*, 2016.

[51] Eduroam, “https://www.eduroam.org/,” World wide education roaming for research and education.

[52] A. Drichel, U. Meyer, S. Schüppen, and D. Teubert, “Making use of NXt to nothing: Effect of class imbalances on DGA detection classifiers,” in *Conference on Availability, Reliability and Security*. ACM, 2020.

[53] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[55] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Advances in Cryptology – ASIACRYPT*. Springer, 2017.

[56] B. Yu, D. L. Gray, J. Pan, M. D. Cock, and A. C. A. Nascimento, “Inline dga detection with deep networks,” in *IEEE International Conference on Data Mining Workshops*, 2017.

[57] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” *Neural Networks*, vol. 6, no. 6, 1993.

[58] S. Sonoda and N. Murata, “Neural network with unbounded activation functions is universal approximator,” *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, 2017.

[59] D.-X. Zhou, “Universality of deep convolutional neural networks,” *Applied and Computational Harmonic Analysis*, vol. 48, no. 2, 2020.

[60] G. Philipp and J. G. Carbonell, “The nonlinearity coefficient-predicting overfitting in deep neural networks,” *arXiv:1806.00179*, 2018.