An Efficient and Zero-Knowledge Classical Machine Learning Inference Pipeline

Haodi Wang, Rongfang Bie, Thang Hoang

Abstract— Machine Learning as a Service (MLaaS) offers powerful data analytics services to clients with limited resources. However, it still raises concerns about the integrity of delegated computation and the privacy of the server's model parameters. To address these issues, zero-knowledge Machine Learning (zkML) has been suggested for computation verifiability with privacy guarantee for ML models. Nevertheless, the existing zkML schemes focus on only one classical ML classification algorithm or deep neural networks, which may not achieve satisfactory accuracy or require large-scale training data and model parameters, thus limiting their usefulness in certain applications.

In this paper, we propose ezDPS, an efficient and zero-knowledge scheme for classical ML inference that processes data in multiple stages for improved accuracy. Unlike prior works, each stage of the ezDPS pipeline is based on a well-established classical ML algorithm, including Discrete Wavelet Transformation, Zero-Score Normalization, Principal Components Analysis, and Support Vector Machine. We design new gadgets to prove various ML operations effectively. Our implementation of ezDPS has been fully tested on real datasets, and experimental results show that it is up to three orders of magnitude more efficient than generic circuit-based approaches, while also maintaining greater accuracy than single ML classification approaches.

Index Terms—Classical Machine Learning, Zero Knowledge Proofs, Inference Pipeline.

1 Introduction

Machine learning (ML) has become a promising paradigm due to its ability to perform highly complicated tasks in classification, object detection, pattern recognition, and natural language processing. However, training a sophisticated machine learning model requires a large amount of resources and relative expertise, making it non-trivial for individuals or small organizations to perform. To address this issue, Machine Learning as a Service (MLaaS) has been proposed in which a cloud server provides an API of ML services for resource-limited clients to access. Despite recent progress, it has been shown that existing MLaaS designs have computation integrity issues.

In the following, we outline the main concerns of MLaaS and the limitations of the existing solutions. Then we present our research objective toward mitigating some of these limitations.

1.1 Research Gap and Problem Statement

MLaaS provides a potential approach to alleviate the computing limitations of the clients. Nevertheless, it is difficult for the client to know whether the results she receives are reliable responses. A reckless server may give out incorrect computation results by mistake. Moreover, a corrupted server may arbitrarily manipulate or substitute the client data to produce a malicious outcome. This computation integrity problem is especially essential in sensitive scenarios, e.g., medical diagnosis, intrusion detection, and financial forecasting.

Some previous work utilized Verifiable Computation (VC) to solve this integrity problem. VC requires the server to generate a *proof* along with the ML inference results and send them to the client. The latter can validate the proof to check the correctness of the results [18]. However, VC is insufficient in the MLaaS scenario because it only guarantees the computation integrity but not the privacy of the server. More concretely, the server in MLaaS utilizes private datasets to train the models. Thus the model parameters contain private information of the server, which may cost quantities of resources to acquire. Revealing the model weights using VC violates the interests of the server. To address this issue, it is viable to add the Zero-Knowledge property to the VC method (zkVC [21]), which permits the server to hide the model parameters when generating the proof. However, due to the heavy computation overhead, it is nontrivial to apply zkVC to MLaaS. Zhang *et al.* initiated the first work [74] that utilized zkVC to solve the integrity and privacy problems in ML prediction. In their design, the server first commits to all the fixed model parameters after training. The client can submit an image via the MLaaS interface for classification service. Given the input, the server computes the inference result and generates a proof using zkVC, which permits the client to verify the inference result regarding the inputs and the committed parameters without obtaining any information about the model parameters.

There are some researches proposed in the literature of zkML that are designed for single stage or simple models [40], [43], [74]. However, in real-world applications, the data is usually processed by several phases called ML pipeline instead of a single inference algorithm. Thus, it is necessary to design a zero-knowledge ML pipeline (e.g., composed

- *Haodi Wang and Rongfang Bie are with the School of Artificial Intelligence, Beijing Normal University, China.E-mail: whd@mail.bnu.edu.cn, rfbie@bnu.edu.cn*
- *Thang Hoang is with Virginia Tech (corresponding author). Email: thanghoang@vt.edu*

Manuscript received xxx; revised xxx

of denoising, normalization, feature extraction, and classification) to balance the model complexity and performance under certain situations.

The objective of this paper is to design a zero-knowledge scheme for an ML pipeline that comprises a complete ML inference, including data denoising, normalization, feature extraction, and classification. The clients can verify the inference results without knowing the private model parameters at every processing stage.

1.2 Our Contributions

In this paper, we craft a reliable and efficient scheme for verifying the inference results from an outsourced ML pipeline. The proposed scheme, called ezDPS, enables practical verification with zero-knowledge confidentiality. Our design consists of four main stages of a typical ML inference pipeline, i.e., data denoising, feature extraction, and ML classification. More concretely, we initialize these stages with established classical ML algorithms, including Discrete Wavelet Transformation (DWT) [67] and Normalization for preprocessing, Principal Components Analysis (PCA) [71] for feature extraction, and Support Vector Machine (SVM) [8] for classification. These ML algorithms are widely adopted in various applications [45], [46] due to their effectiveness. To the best of our knowledge, this paper takes the first step toward establishing a zero-knowledge ML inference pipeline. Our concrete contributions are as follows.

- **New gadgets for critical ML operations.** We propose several new gadgets to transform ML computation into arithmetic circuits, e.g., exponentiation, absolute value, max/min in an array, and square root computation (§4.1). The gadgets can be utilized to prove our ML inference pipeline and other ML operations such as deep learning.
- **New zero-knowledge ML inference pipeline scheme.** Based on the proposed gadgets, we propose a new zero-knowledge ML inference pipeline called ezDPS (§4.2). Unlike the existing zkML schemes, ezDPS permits efficient and practical integrity proof for a complete ML pipeline, including data preprocessing, feature extraction, and classification. We design an optimal set of constraints for each stage and present multiple optimizations to reduce the model size, making ezDPS outperform the baseline methods both in asymptotic and concrete performance metrics. Note that our ezDPS is designed to be compatible with any zkVC backend. Thus, the efficiency can be further improved when a better zkVC is adopted. We also provide a zero-knowledge proof-of-accuracy scheme for validating the effectiveness of the ML pipeline (§4.2.6).
- **Formal security analysis.** We illustrate the security model and rigorously analyze the security strengths of ezDPS. The rigorous security analysis shows that ezDPS satisfies the definition of a zero-knowledge ML inference pipeline (§5).
- **Full-fledged implementation, evaluation, and comparison.** We fully implement our ezDPS in Python and Rust programming language (§6) and conduct comprehensive experiments to evaluate the performance of our method (§7). The experimental results on real-world datasets demonstrate that ezDPS achieves one-to-three orders of magnitude more efficiently than the baseline method in all performance matrices (i.e., proving time, verification time, and proof size).

Remark. In this paper, we focus on the verifiability of the ML inference task and the privacy of the server model in the integrity proof. Our technique does not permit client data privacy, in which the client sends plaintext data to the server for computation. This model is different from the standard privacy-preserving ML inference (PPMLI) (e.g., [11], [19], [35], [42], [54], [57]), which preserves the privacy of the client and server against each other but not computation integrity (see §9 for more details). To our knowledge, it is not clear how to combine zero-knowledge with PPMLI efficiently to enable both client and server privacy plus computation integrity. We leave such an investigation as our future work.

Application use-cases. Our zkML inference scheme can be found useful in various applications. First, it can be used to enable *proof-of-genuine* ML services, in which the service provider can prove that its ML model is of high quality, and the inference result is computed from the same model. Another application is a fair ML model trading platform with try-before-buy, in which the buyer can attest to the ML model quality before purchase, while the sellers do not want to reveal their model first. Finally, our technique can partially address the *reproducibility* problem in ML [26], where some ML models are claimed to achieve high accuracy without having a proper way to validate them. As for more concrete examples, our zkML can be adopted in scenarios including federal training of anti-money laundering models [24], [25], artificial diagnoses [3], [72], and student learning behavior analysis [73]. In these use cases, the model-holder can prove the inference results without revealing the private model parameters. In all, our technique can offer a solution to this issue, in which the model owner can prove that there exists an ML model that can achieve such accuracy (see §4.2.6), and the verifier can verify that statement efficiently in zero knowledge.

Improvements over the PETs'23 conference version. This article is the extended version of [23], which concentrates on the classical ML algorithms rather than the deep learning methods. The classical ML pipeline provides better performance for small or middle-sized datasets, which can also effectively avoid possible overfit caused by deep learning models. To prove the classical ML inference pipeline more efficiently, we propose new gadgets as building blocks and add various optimizations to reduce the size of the arithmetic circuits. We also select three more suitable datasets to evaluate our ezDPS. More concretely, the improvements over the conference version include the following parts. First, we focus on the classical ML pipeline and present an efficient proving scheme for inference integrity upon four main stages, i.e., Discrete Wavelet Transformation, Z-Score Normalization, Principal Components Analysis, and Support Vector Machines. These classical ML algorithms have been widely adopted in various tasks and are capable of obtaining high accuracy without incurring possible overfit. Second, from the algorithmic viewpoint, we propose a new gadget for square root computation, which supports more complicated computation in machine learning regions.

The square root gadget can be effectively exerted on the proving procedure of Z-Score Normalization. Moreover, we also propose several optimizations of the ML pipeline based on the new gadget to increase the accuracy of the ML pipeline while reducing the model size. Finally, from the experimental perspective, we have revised all the experiments of our ezDPS scheme in the PETs'23 version with two new datasets that are more appropriate for the classical ML inference scenario. In our preliminary version, we reported the performance of ezDPS on LFW and Cifar-100 image datasets, e.g., the proving time, verification time, proof size, and accuracy loss. Our method requires hundreds of minutes to prove and obtains relatively unsatisfying accuracy. In this extended version, we fixed this issue by adopting two new datasets, i.e., the British Birdsong and KDD-1999. The revised experiments cost only several minutes to prove and achieve over 90% accuracy with little accuracy loss. We have released the improved source-code of our scheme for public use and adaptation. The code is publicly available at

https://github.com/vt-asaplab/ezDPS/tree/extended_ezDPS

2 Preliminaries

Notations. For $n \in \mathbb{N}$, we denote $[1, n] = \{1, \ldots, n\}$. Let λ be the security parameter and $\mathsf{negl}(\cdot)$ be the negligible function. We denote a finite field as $\mathbb{F}$. PPT stands for Probabilistic Polynomial Time. We use bold letters, e.g., $\mathbf{a}$ and $\mathbf{A}$, to denote vector and matrix, respectively. $\mathbf{A}^\top$ means the transpose of $\mathbf{A}$. We write $\mathbf{ab}$ (or $\mathbf{a} \cdot \mathbf{b}$) to denote dot product and $\mathbf{A} \circ \mathbf{B}$ to denote Hadamard (entry-wise) product. We use $\overset{c}{\approx}$ to denote that two quantities are computationally indistinguishable.

2.1 Commit-and-Prove Argument Systems

Argument of knowledge. An argument of knowledge for an NP relation $\mathcal{R}$ is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, in which $\mathcal{P}$ convinces $\mathcal{V}$ that it *knows* a witness w for some input in an NP language $x \in \mathcal{L}$ such that $(x, w) \in \mathcal{R}$. Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote a pair of PPT interactive algorithms. A zero-knowledge argument of knowledge is a tuple of PPT algorithms $\mathsf{zkp} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ that satisfies the following properties.

- *Completeness.* For any $(x, w) \in \mathcal{R}$ and $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, it holds that
$$\langle \mathcal{P}(w, \mathsf{pp}), \mathcal{V}(\mathsf{pp}) \rangle(x) = 1$$
- *Knowledge soundness.* For any PPT prover $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that given the access to the entire execution process and the randomness of $\mathcal{P}^*$, $\mathcal{E}$ can extract a witness w such that $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \pi^* \leftarrow \mathcal{P}^*(x, \mathsf{pp}), w \leftarrow \mathcal{E}^{\mathcal{P}^*}(x, \pi^*, \mathsf{pp})$ and
$$\Pr\left[(x, w) \notin \mathcal{R} \wedge \mathcal{V}(x, \pi^*, \mathsf{pp}) = 1\right] \leq \mathsf{negl}(\lambda)$$
- *Zero-knowledge.* There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, auxiliary input $z \in \{0, 1\}^*$, $(x, w) \in \mathcal{R}$, $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$:
$$\mathsf{view}(\langle \mathcal{P}(w, \mathsf{pp}), \mathcal{V}^*(z, \mathsf{pp}) \rangle(x)) \overset{c}{\approx} \mathcal{S}^{\mathcal{V}^*}(x, z)$$
where $\mathsf{view}(\langle \cdot, \cdot \rangle(x))$ denotes the distribution of the transcript of interaction.

Commit-and-Prove zero-knowledge proof. Commit-and-Prove (CP) Zero-Knowledge Proof (ZKP) permits the prover to prove the NP-statements on the committed witness. Most generic ZKP protocols support CP paradigm and the most efficient CP-ZKP protocols harness the succinct polynomial commitment scheme (e.g., [36]) to achieve succinctness properties. The prover first commits to the witness w using a zero-knowledge polynomial commitment scheme before proving an NP statement, and the verifier takes the committed value as an additional input for verification. We denote the commitment algorithm for CP-ZKP as $\mathsf{cm}_w \leftarrow \mathsf{zkp.Com}(w, r, \mathsf{pp})$, where r is the randomness chosen by the prover.

In our framework, we use Spartan [59] (with Hyrax [68] as the underlying polynomial commitment scheme) as the backend zkPC-based CP-ZKP protocol. We choose Spartan because it is a fully implemented zero-knowledge proof scheme, which is transparent and supports generic Rank-1 Constraint Systems. Spartan is also effective with linear proving time, sub-linear verification time, and proof size. Generally speaking, Spartan supports NP statements expressed as R1CS, which shows that there exists a vector $z = (x, 1, w)$ such that $\mathbf{A}z \circ \mathbf{B}z = \mathbf{C}z$, where $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are matrices for the arithmetic circuits, x is the public input (statement), w is the witness of the prover. All the witnesses are encoded as a polynomial on the Lagrange basis. Since it is easy to convert arithmetic statements into R1CS, our main focus is to create arithmetic constraints for proving algorithms in the ML pipeline efficiently that can be realized with Spartan or any CP-ZKP backend.

Theorem 1 (Spartan ZKP [59]). *Let $\mathbb{F}$ be a finite field and $\mathcal{C}_\mathbb{F}$ be a family of the arithmetic circuit over $\mathbb{F}$ of size n. Under standard cryptographic hardness assumptions, there exists a family of succinct argument of knowledge for the relation*
$$\mathcal{R} = \{(C, x; w) : C \in C_\mathbb{F} \wedge C(x; w) = 1\}$$
where x and w are the public input and the auxiliary input to the circuit C, respectively, and the prover incurs $O(n)$ to $O(n \log n)$ overhead, the verifier's time and communication costs range from $O(\log^2 n)$ to $O(\sqrt{n})$ depending on the underlying polynomial commitment schemes being used for multilinear polynomials.

Note that since Spartan is established on the polynomial commitment schemes, it can support CP-ZKP paradigm.

2.2 Machine Learning Pipeline

ML pipeline is an end-to-end process that consists of multiple data processing phases to train an ML model from a large-scale dataset effectively and to predict an inference result for a new observation accurately [33]. Without loss of generality, an efficient ML pipeline contains four main parts, including data denoising, normalization, feature extraction, and classification. The pipeline is illustrated in Figure 1. In data denoising, raw samples $\mathbf{x}_{\mathsf{in}} \in \mathbb{F}^m$ are collected, and then some algorithm is used to reduce the impact of noise in the collection environment. The output is denoted as $\mathbf{x}_{\mathsf{dn}} \in \mathbb{F}^m$. Normalization takes $\mathbf{x}_{\mathsf{dn}}$ as input and generate $\mathbf{x}_{\mathsf{nl}} \in \mathbb{F}^m$, which shifts and scales the data into a standard distribution. Feature extraction extracts the most prominent dimension of $\mathbf{x}_{\mathsf{nl}}$ so that only a small set of features $\mathbf{x}_{\mathsf{fe}} \in \mathbb{F}^k, k < m$ will be fetched for efficient computation and a high convergence rate. Finally, the ML classification algorithm trains upon $\mathbf{x}_{\mathsf{fe}}$ to simulate the data distribution with a set of model parameters. After the training procedure is completed, the classification model can predict the label y for a new observation.

In this paper, we focus on the ML inference pipeline (MLIP), in which the client collects raw data, and the server processes the data in multiple stages (i.e., denoising, normalization, feature extraction, ML classification) to obtain the final inference result. At each stage, the server can employ its private ML model parameters obtained from its training pipeline to process the client data. We denote such MLIP functionality as $y \leftarrow \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x}_{\mathsf{in}})$, where $\mathbf{x}_{\mathsf{in}} \in \mathbb{F}^m$ is the data sample, $\mathbf{w} \in \mathbb{F}^n$ is MLIP model parameters in all stages, and $y \in \mathbb{F}$ is the inference result.

3 Models

System and threat models. Our system consists of two parties, including the client and the server. The server holds well-trained MLIP model parameters $\mathbf{w}$ and provides an interface for the client to classify her data sample $\mathbf{x}_{\mathsf{in}}$ using its model $\mathbf{w}$.

We consider the client and server to mutually distrust each other. The adversarial server can be malicious, in which it may process the client's query arbitrarily. On the other hand, the client is semi-honest, in which she is curious about the server's model parameters. In this setting, we aim to achieve inference integrity and model privacy. To enable inference integrity, the server first commits to its model $\mathbf{w}$. Given a client request, the server computes the inference result y along with a proof π to convince the client that the result is indeed computed from the committed model rather than an arbitrary answer. To ensure model privacy, the proof π should not leak any information about the model $\mathbf{w}$.

Formally speaking, a zero-knowledge MLIP is a tuple of algorithms $\mathsf{zkMLIP} = (\mathcal{G}, \mathsf{Com}, \mathcal{P}, \mathcal{V})$ as follows

- $\mathsf{pp} \leftarrow \mathsf{zkMLIP}.\mathcal{G}(1^\lambda, n)$: Given a security parameter λ and a bound on the size of the MLIP model parameters n, it outputs public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{zkMLIP}.\mathsf{Com}(\mathbf{w}, r, \mathsf{pp})$: Given MLIP parameters $\mathbf{w}$, it outputs a commitment cm under randomness r.
- $(y, \pi) \leftarrow \mathsf{zkMLIP}.\mathcal{P}(\mathbf{w}, \mathbf{x}_{\mathsf{in}}, \mathsf{pp})$: Given MLIP model parameters $\mathbf{w}$ and a data sample $\mathbf{x}_{\mathsf{in}}$, it outputs the inference result $y = \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x}_{\mathsf{in}})$ and the proof π.
- $\{0,1\} \leftarrow \mathsf{zkMLIP}.\mathcal{V}(\mathsf{cm}, \mathbf{x}_{\mathsf{in}}, y, \pi, \mathsf{pp})$: Given a commitment cm, a sample $\mathbf{x}_{\mathsf{in}}$, an inference result y, and a proof π, it outputs 1 if π is the valid proof for $y = \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x}_{\mathsf{in}})$ and $\mathsf{cm} = \mathsf{Com}(\mathbf{w}, r, \mathsf{pp})$; otherwise it outputs 0.

Security model. We define the security definition of zero-knowledge MLIP that captures inference integrity and model privacy in the integrity proof as follows.

Definition 1 (zero-knowledge MLIP). *A scheme is zero-knowledge MLIP if it satisfies the following properties.*

- ***Completeness.*** *For any* $\mathbf{w} \in \mathbb{F}^n$ *and* $\mathbf{x}_{\mathsf{in}} \in \mathbb{F}^m$, $\mathsf{pp} \leftarrow \mathsf{zkMLIP}.\mathcal{G}(1^\lambda, n)$, $\mathsf{cm} \leftarrow \mathsf{zkMLIP}.\mathsf{Com}(\mathbf{w}, r, \mathsf{pp})$, $(y, \pi) \leftarrow \mathsf{zkMLIP}.\mathcal{P}(\mathbf{w}, \mathbf{x}_{\mathsf{in}}, \mathsf{pp})$, *it holds that*

$$\Pr\left[\mathsf{zkMLIP}.\mathcal{V}(\mathsf{cm}, \mathbf{x}_{\mathsf{in}}, y, \pi, \mathsf{pp}) = 1\right] = 1$$

- ***Soundness.*** *For any* PPT *adversary* $\mathcal{A}$, *it holds that*

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{zkMLIP}.\mathcal{G}(1^\lambda, n) \\ (\mathsf{cm}^*, \mathbf{w}^*, \mathbf{x}_{\mathsf{in}}, y^*, \pi^*, r) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{cm}^* = \mathsf{zkMLIP}.\mathsf{Com}(\mathbf{w}^*, r, \mathsf{pp}) \\ \mathsf{zkMLIP}.\mathcal{V}(\mathsf{cm}^*, \mathbf{x}_{\mathsf{in}}, y^*, \pi^*, \mathsf{pp}) = 1 \\ \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}^*, \mathbf{x}_{\mathsf{in}}) \neq y^* \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- ***Zero-knowledge.*** *For any MLIP model* $\mathbf{w} \in \mathbb{F}^n$ *and PPT algorithm* $\mathcal{A}$, *there exists simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that*

$$\Pr\left[\mathcal{A}(\mathsf{cm}, \mathbf{x}_{\mathsf{in}}, y, \pi, \mathsf{pp}) = 1 \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{zkMLIP}.\mathcal{G}(1^\lambda, n) \\ \mathsf{cm} \leftarrow \mathsf{zkMLIP}.\mathsf{Com}(\mathbf{w}, r, \mathsf{pp}) \\ \mathbf{x}_{\mathsf{in}} \leftarrow \mathcal{A}(\mathsf{cm}, \mathsf{pp}) \\ (y, \pi) \leftarrow \mathsf{zkMLIP}.\mathcal{P}(\mathbf{w}, \mathbf{x}_{\mathsf{in}}, \mathsf{pp}) \end{array}\right] \overset{c}{\approx}$$
$$\Pr\left[\mathcal{A}(\mathsf{cm}, \mathbf{x}_{\mathsf{in}}, y, \pi, \mathsf{pp}) = 1 \;\middle|\; \begin{array}{r} (\mathsf{cm}, \mathsf{pp}) \leftarrow \mathcal{S}_1(1^\lambda, n, r) \\ \mathbf{x}_{\mathsf{in}} \leftarrow \mathcal{A}(\mathsf{cm}, \mathsf{pp}) \\ (y, \pi) \leftarrow \mathcal{S}_2^{\mathcal{A}}(\mathsf{cm}, \mathbf{x}_{\mathsf{in}}, r, \mathsf{pp}), \textit{given} \\ \textit{oracle access to } y = \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x}_{\mathsf{in}}) \end{array}\right]$$

Out-of-scope attacks. Our security definition captures the inference integrity and the model privacy in the integrity proof π. There exist model stealing attacks [7], [65] that target only the inference result y to reconstruct the model $\mathbf{w}$. In this paper, we do not focus on addressing such vulnerabilities. It is because there exist independent studies that address these vulnerabilities (e.g., [7], [32], [37], [41], [65]) and, with some efforts, they can be integrated orthogonally into our scheme to protect $\mathbf{w}$ from both y and π. For example, by simply limiting the inference result information (i.e., returning only the predicted label like our scheme currently offers), it makes the attack become 50-100$\times$ more difficult [65]. We elaborate on all these approaches in §8.

Our main goal is to ensure $\mathbf{w}$ is not leaked from π via zero-knowledge so that the leakage from y can be sealed or mitigated independently by these techniques.

We also do not consider model poisoning/backdoor attacks (e.g., [55], [56]), in which the adversarial server may target adversarial behaviors on certain data samples while maintaining an overall high level of accuracy. Mitigating such attacks requires analyzing the model parameters (e.g., [44], which may be highly challenging in our setting, where the model privacy is preserved. Thus, we leave this threat model as an open research problem for future investigation.

4 Our Proposed Zero-Knowledge MLIP Framework

In this section, we present the detailed construction of our framework. We start by giving an overview.

Overview. Our ezDPS framework contains four processing phases, including data denoising, normalization, feature extraction, and ML classification, as shown in Figure 1. We adopt ML algorithms for each phase including Discrete Wavelet Transformation (DWT) [67] for data denoising, Z-Score Normalization for normalization, Principal Components Analysis (PCA) [71] for feature extraction, and Support Vector Machine (SVM) [8] for classification. We focus on these algorithms because they were well-established in various systems and applications with high efficiency [45], [46]. ezDPS permits to verify a data sample was computed correctly with DWT, SN, PCA, and SVM without leaking the parameters at each phase including, for example, low-pass

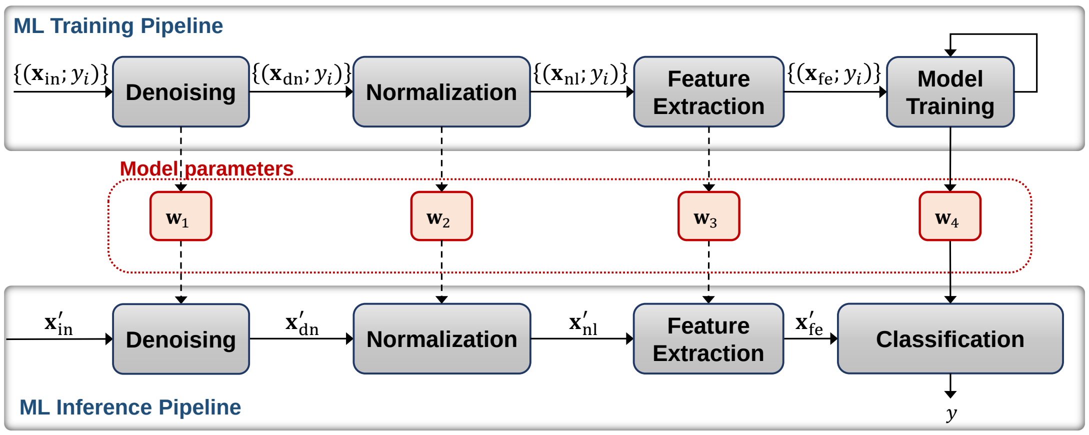

ML Training Pipeline
$\{(\mathbf{x}_{\text{in}}; y_i)\}$
Denoising
$\{(\mathbf{x}_{\text{dn}}; y_i)\}$
Normalization
$\{(\mathbf{x}_{\text{nl}}; y_i)\}$
Feature Extraction
$\{(\mathbf{x}_{\text{fe}}; y_i)\}$
Model Training
Model parameters
$\mathbf{w}_1$
$\mathbf{w}_2$
$\mathbf{w}_3$
$\mathbf{w}_4$
$\mathbf{x}'_{\text{in}}$
Denoising
$\mathbf{x}'_{\text{dn}}$
Normalization
$\mathbf{x}'_{\text{nl}}$
Feature Extraction
$\mathbf{x}'_{\text{fe}}$
Classification
y
ML Inference Pipeline

Fig. 1: A general ML pipeline.

and high-pass filters in DWT; mean vector and eigenvectors in PCA; and support vectors in SVM.

In ezDPS, the server first commits to the model parameters of each ML algorithm and provides an interface for the client to process her data sample based on the committed parameters. To demonstrate the validity of the committed model, the server can publish a zero-knowledge Proof-of-Accuracy (zkPoA) to demonstrate that the committed model maintains a desirable accuracy on public datasets with ground truth labels. zkPoA permits the client to attest to the genuineness and the effectiveness of the server's committed model before using the inference service on her data sample. zkPoA can be derived from zero-knowledge proof of inference of individual samples. We show how to construct zkPoA for our scheme in §4.2.6.

In the following sections, we first present new gadgets for critical ML operations (e.g., max/min, absolute). Notice that our proposed gadgets are not limited to the ML algorithms selected above and can be used to prove other useful ML kernels. We then present our techniques for proving DWT, SN, PCA, and SVM more efficiently than the generic approaches. Finally, we show how to construct a zkPoA scheme to attest to the effectiveness of the committed model on public datasets.

4.1 Gadgets

A gadget is an intermediate constraint system consisting of a set of arithmetic constraints for proving a particular statement in the higher-level protocols. We present the gadgets that are needed in our ezDPS sheme, which can also be applied to other ML algorithms. Note that in this section, we first formalize the previously proposed permutation and binarization gadgets as our building blocks, and then present the gadgets that we designed.

4.1.1 Building Blocks

Permutation gadget [74]. Given two vectors $\mathbf{v}, \mathbf{v}' \in \mathbb{F}^n$, $\mathsf{Perm}(\mathbf{v}, \mathbf{v}')$ permits to prove that $\mathbf{v}$ is the permutation of $\mathbf{v}'$, i.e., $\mathbf{v}[i] = \mathbf{v}'[\sigma(i)]$ for $i \in [1, n]$ according to some permutation σ. This can be done by showing that their characteristic polynomial evaluates to the same value at a random point α chosen by the verifier as

$$\prod_{i=1}^{n}(\mathbf{v}[i] - \alpha) = \prod_{i=1}^{n}(\mathbf{v}'[i] - \alpha)$$

Due to Schwartz-Zippel Lemma [58], the soundness error of the permutation test is $\frac{n}{|\mathbb{F}|} = \mathsf{negl}(\lambda)$.

Binarization gadget . Given a vector $\mathbf{v} \in \mathbb{F}^n$ and a value $a \in \mathbb{F}$, binarization gadget $\mathsf{Bin}(a, \mathbf{v}, n)$ permits to prove that $\mathbf{v}$ is a binary representation of a. This can be done by showing that

$$\begin{cases} \mathbf{v}[i] \times \mathbf{v}[i] = \mathbf{v}[i] \text{ for } i \in [1, n] \\ \sum_{i=1}^{n} \mathbf{v}[i] \cdot 2^{i-1} = a \end{cases}$$

4.1.2 New Gadgets for Zero-Knowledge MLIP

Exponent gadget. Note that the idea to prove the exponentiation was first proposed by Zhang et al. [74]. In this paper, we present the concrete constraints and formulate them in the gadget format. Given two values $b, x \in \mathbb{F}$, we propose a gadget $\mathsf{Exp}(b, a, x)$ to prove $b = a^x$ for public value $a \in \mathbb{F}$ [74]. This can be done using the multiplication tree and the binarization gadget (Bin). Let $\mathbf{v} \in \mathbb{F}^n$ be an auxiliary witness. It suffices to show that

$$\begin{cases} \mathsf{Bin}(x, \mathbf{v}, n) \\ b = \prod_{i=1}^{n}(a^{2^{i-1}} \cdot \mathbf{v}[i] + (1 - \mathbf{v}[i])) \end{cases}$$

GreaterThan gadget. Given two values $a, b \in \mathbb{F}$, we create a gadget $\mathsf{GT}(a, b)$ to prove that $a > b$. The main idea is to compute an auxiliary witness $c := 2^n + (a - b)$, where n is the length of the binary representation of a and b, and show that the most significant bit of c is equal to 1. Let $\mathbf{c} \in \mathbb{F}^{n+1}$ and $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ be additional auxiliary witnesses. The set of arithmetic constraints to prove $a > b$ is

$$\begin{cases} c = 2^n + a - b \\ \mathsf{Bin}(a, \mathbf{a}, n) \\ \mathsf{Bin}(b, \mathbf{b}, n) \\ \mathsf{Bin}(c, \mathbf{c}, n+1) \\ \mathbf{c}[n+1] = 1 \end{cases}$$

Maximum/Minimum gadget. Given a value $v \in \mathbb{F}$ and an array $\mathbf{a} \in \mathbb{F}^n$, we create a gadget $\mathsf{Max}(v, \mathbf{a})$ (resp. $\mathsf{Min}(v, \mathbf{a})$) to prove that v is the maximum (resp. minimum) value in

TABLE 1: Notation table.

Variables	Description
DWT components	
$\mathbf{x}_{\mathsf{in}} \in \mathbb{F}^m$	Sample input of size m to DWT
$\mathbf{h}, \bar{\mathbf{h}} \in \mathbb{F}^c$	low-pass filter of size c and its inverse
$\mathbf{g}, \bar{\mathbf{g}} \in \mathbb{F}^c$	high-pass filter of size c and its inverse
η	Filter threshold
Z-Score components	
$\mathbf{x}_{\mathsf{dn}} \in \mathbb{F}^m$	Sample input of size m to Z-Score
ν	mean of the data sample
$\varkappa$	standard deviation of the data sample
PCA components	
$\mathbf{x}_{\mathsf{nl}} \in \mathbb{F}^m$	Sample input of size m to PCA
$\bar{\mathbf{x}} \in \mathbb{F}^m$	Mean vector
$\mathbf{V} = [\mathbf{v}_1^T, ..., \mathbf{v}_m^T]$	Eigenvectors
$(\lambda_1, ..., \lambda_m)$	Eigenvalues
k	Size of PCA output
SVM components	
$\mathbf{x}_{\mathsf{fe}} \in \mathbb{F}^k$	Sample input of size m to SVM
ϕ	kernel function
γ	RBF kernel parameter
$\mathbf{x}_i^{(\hat{c})}$	Support vectors for class $\hat{c}$
$\mathbf{w}^{(\hat{c})}, b^{(\hat{c})}$	Weights and bias for class $\hat{c}$
$y_i^{(\hat{c})} \in \{0, 1\}$	Label of class $\hat{c}$
$\delta^{(\hat{c})}$	Coefficients of class $\hat{c}$ in RBF kernel
$f^{(\hat{c})}$	Decision function of class $\hat{c}$
Proof components	
σ	Permutation function
λ	Security parameter
π	Proof
$\mathbf{w}$	Witness
aux	Auxiliary witness
cm	Commitment
$\alpha, \bar{\alpha}, \beta$	Random challenges

$\mathbf{a}$. The idea is to harness Perm and GT gadgets to prove that v is equal to the first element of the permuted array of $\mathbf{a}$, whose first element is the largest (resp. minimum) value. Specifically, to prove $v = \max(\mathbf{a})$, it suffices to show (i) $v = \mathbf{a}'[1]$, (ii) $\mathbf{a}'[1] > \mathbf{a}'[i]$ for all $i \in [2, n]$, and (iii) $\mathbf{a}'$ is the permutation of $\mathbf{a}$. Let $\mathbf{a}' \in \mathbb{F}^n$ be an auxiliary witness. The set of arithmetic constraints to prove a maximum value in an array is

$$\begin{cases} \mathsf{GT}(\mathbf{a}'[1], \mathbf{a}'[i]) \text{ for all } i \in [2, n] \\ v = \mathbf{a}'[1] \\ \mathsf{Perm}(\mathbf{a}, \mathbf{a}') \end{cases}$$

The constraints to prove a minimum value in an array can be defined analogously.

Absolute gadget. Given $a', a \in \mathbb{F}$, we create gadget $\mathsf{Abs}(a', a)$ to prove that a' is the absolute value of a, i.e., $a = a'$ or $-a = a'$. The idea is to compute $c = a + 2^n$, where n is the length of the binary representation of a, and show that the most significant bit of c represents the sign difference of a and a'. Let $\mathbf{c} \in \mathbb{F}^{n+1}$ and $\mathbf{a} \in \mathbb{F}^n$ be auxiliary witnesses, the set of arithmetic constraints to show that a' is the absolute value of a is

$$\begin{cases} c = a + 2^n \\ \mathsf{Bin}(a, \mathbf{a}, n) \\ \mathsf{Bin}(c, \mathbf{c}, n+1) \\ \mathbf{c}[n+1](a + a') + \mathbf{c}[n+1](a - a') = 0 \end{cases}$$

SquareRoot gadget. Given $a, b \in \mathbb{F}$, we propose gadget $\mathsf{Sqr}(b, a)$ to prove that b is the square root of a, i.e., $b = \sqrt{a}$. Due to the fixed-point representation in ZKP, there is a difference between a and b^2. The essential observation is to harness the reverse square computation and a range constraint, i.e., $a - b^2$ is smaller than a small constant. Let $s, c \in \mathbb{F}$ be the auxiliary witness, the set of constraints to show that b is the square root of a is

$$\begin{cases} b^2 + c = a \\ \mathsf{GT}(1, s \cdot c) \end{cases}$$

4.2 ezDPS Framework

We now give the detailed construction of our ezDPS scheme with DWT, PCA, and SVM algorithms. We provide the overview of each algorithm and show how to prove it with a small number of constraints. We summarize all the variables and notation being used for our detailed description in Table 1.

4.2.1 DWT-Based Data Preprocessing

DWT [67] exerts the wavelet coefficients on the raw data sample to project it to the wavelet domain for efficient preprocessing. A DWT algorithm contains three main operations, including decomposition, thresholding, and reconstruction. The decomposition transforms the raw input from the spatial/time domain to the wavelet domain consisting of approximation and detail coefficients. The thresholding is then applied to filter some detail coefficients, which generally contain noise. Finally, the reconstruction is applied to reconstruct the original data after noise reduction. Such decomposition and thresholding processes can be applied recursively until a small constant number of coefficients is obtained. Let $\mathbf{x}_{\mathsf{in}} \in \mathbb{F}^m$ be the input data sample of length m, $t_\ell := \frac{m}{2^\ell}$, $t'_\ell := \frac{m}{2^{\ell-1}}$. The DWT computes the frequency component $\mathbf{z}_\ell \in \mathbb{F}^{t'_\ell}$ at the recursion level $\ell \geq 1$ as

$$\begin{aligned} \mathbf{z}_\ell[i] &= \sum_{j=1}^{c} \mathbf{h}[j] \cdot \mathbf{z}_{\ell-1}[(2i + j - 2)_{\mathrm{mod}\ t'_\ell}] \\ \mathbf{z}_\ell[i + t_\ell] &= \sum_{j=1}^{c} \mathbf{g}[j] \cdot \mathbf{z}_{\ell-1}[(2i + j - 2)_{\mathrm{mod}\ t'_\ell}] \end{aligned} \tag{1}$$

for $i \in [1, t_\ell]$, where $\mathbf{h}, \mathbf{g} \in \mathbb{F}^c$ are low-pass and high-pass filters respectively, and $\mathbf{z}_0 = \mathbf{x}_{\mathsf{in}}$. The thresholding is applied to compute high-frequency components (i.e., detail coefficients) as

$$\begin{aligned} \mathbf{z}'_\ell[i] &= \mathbf{z}_\ell[i] \\ \mathbf{z}'_\ell[i + t_\ell] &= \begin{cases} \mathsf{sign}(\mathbf{z}_\ell[i + t_\ell])(\mathbf{z}_\ell[i + t_\ell] - \eta) & \text{if } |\mathbf{z}_\ell[i + t_\ell]| - \eta > 0 \\ 0 & \text{if } |\mathbf{z}_\ell[i + t_\ell]| - \eta < 0 \end{cases} \end{aligned} \tag{2}$$

for $i \in [1, t_\ell]$, where η is the public threshold parameter, $\mathsf{sign}(x)$ returns the sign of x (i.e., 1 if $x \geq 0$, and -1 otherwise). The decomposition and thresholding can be applied recursively until $t_\ell < c$, or the number of rounds reaches a set value. Finally, the reconstructed data $\hat{\mathbf{x}}_\ell \in \mathbb{F}^{t'_\ell}$ at recursion level ℓ is computed as

$$\hat{\mathbf{x}}_\ell[2i-1] = \sum_{j=1}^{c/2}(\bar{\mathbf{h}}[2j-1]\cdot \mathbf{z}'_\ell[(i+j-1)_{\mathrm{mod}\ t_\ell}] + \bar{\mathbf{h}}[2j]\cdot \mathbf{z}'_\ell[t_\ell + (i+j-1)_{\mathrm{mod}\ t_\ell}])$$
$$\hat{\mathbf{x}}_\ell[2i] = \sum_{j=1}^{c/2}(\bar{\mathbf{g}}[2j-1]\cdot \mathbf{z}'_\ell[(i+j-1)_{\mathrm{mod}\ t_\ell}] + \bar{\mathbf{g}}[2j]\cdot \mathbf{z}'_\ell[t_\ell + (i+j-1)_{\mathrm{mod}\ t_\ell}]) \tag{3}$$

for $i \in [1, t_\ell]$, $\bar{\mathbf{h}}, \bar{\mathbf{g}} \in \mathbb{F}^c$ are the coefficients of the inverse low-pass and high-pass filters, respectively. In summary, the DWT model parameters are $\mathbf{h}, \mathbf{g}, \bar{\mathbf{h}}, \bar{\mathbf{g}}, \eta$. The size of the model parameter is $4c+1$, where c depends on the concrete DWT algorithm used in practice, e.g., $c = 4$ in DB-4 algorithm.

Proving DWT computation. We can see that (1) incurs $8m(1-\frac{1}{2^l})$ constraints, where m is the length of the data sample, l is the number of recursion levels. We propose a novel method to prove DWT computation in a more efficient way using our proposed split technique along with the product of sums and random linear combination. Our optimization reduces the complexity of proving the decomposition and reconstruction from $O(m)$ to $O(\log m)$. Furthermore, if the recursion level l is set to a constant, the complexity can be reduced to $O(1)$. Specifically, we first split each element in $\mathbf{z}_\ell \in \mathbb{F}^{t'_\ell}$ into two parts as

$$\mathbf{z}_\ell[i]^{(1)} = \sum_{k=1}^{c/2}\mathbf{h}[2k-1]\cdot \mathbf{z}_{\ell-1}[(2k+2i-3)_{\mathrm{mod}\ t'_\ell}]$$
$$\mathbf{z}_\ell[i]^{(2)} = \sum_{k=1}^{c/2}\mathbf{h}[2k]\cdot \mathbf{z}_{\ell-1}[(2k+2i-2)_{\mathrm{mod}\ t'_\ell}]$$
$$\mathbf{z}_\ell[i+t_\ell]^{(1)} = \sum_{k=1}^{c/2}\mathbf{g}[2k-1]\cdot \mathbf{z}_{\ell-1}[(2k+2i-3)_{\mathrm{mod}\ t'_\ell}]$$
$$\mathbf{z}_\ell[i+t_\ell]^{(2)} = \sum_{k=1}^{c/2}\mathbf{g}[2k]\cdot \mathbf{z}_{\ell-1}[(2k+2i-2)_{\mathrm{mod}\ t'_\ell}] \tag{4}$$

for $i \in [1, t_\ell]$. Let $\alpha \in \mathbb{F}$ be a random scalar chosen by the verifier, the prover can prove (4) holds such that

$$\sum_{i=1}^{t_\ell}\alpha^{\frac{c}{2}+i-2}\mathbf{z}_\ell[i] = \sum_{k=1}^{c/2}\alpha^{\frac{c}{2}-k}\mathbf{h}[2k-1]\cdot\sum_{i=1}^{t_\ell}\alpha^{i-1}\mathbf{z}_{\ell-1}[2i-1]$$
$$+\sum_{k=1}^{c/2}\alpha^{\frac{c}{2}-k}\cdot\mathbf{h}[2k]\cdot\sum_{i=1}^{t_\ell}\alpha^{i-1}\mathbf{z}_{\ell-1}[2i] + (\alpha^{t_\ell}-1)\sum_{q=1}^{\frac{c}{2}-1}\alpha^{q-1}$$
$$\cdot\sum_{p=1}^{q}(\mathbf{z}_{\ell-1}[2p]\mathbf{h}[c-2q+2p] + \mathbf{z}_{\ell-1}[2p-1]\mathbf{h}[c-2q+2p-1])$$
$$\sum_{i=1}^{t_\ell}\alpha^{\frac{c}{2}+i-2}\mathbf{z}_\ell[i+t_\ell] = \sum_{k=1}^{c/2}\alpha^{\frac{c}{2}-k}\mathbf{g}[2k-1]\cdot\sum_{i=1}^{t_\ell}\alpha^{i-1}\mathbf{z}_{\ell-1}[2i-1]$$
$$+\sum_{k=1}^{c/2}\alpha^{\frac{c}{2}-k}\mathbf{g}[2k]\cdot\sum_{i=1}^{t_\ell}\alpha^{i-1}\mathbf{z}_{\ell-1}[2i] + (\alpha^{t_\ell}-1)\sum_{q=1}^{\frac{c}{2}-1}\alpha^{q-1}$$
$$\cdot\sum_{p=1}^{q}(\mathbf{z}_{\ell-1}[2p]\mathbf{g}[c-2q+2p] + \mathbf{z}_{\ell-1}[2p-1]\mathbf{g}[c-2q+2p-1]) \tag{5}$$

In (5), the number of constraints for proving DWT decomposition is reduced from mc to $c(\frac{c}{2}-1)+4$. To prove the thresholding computation in (2), we employ the GT gadget, such that for $i \in [1, t_\ell]$:

$$\begin{cases}\mathsf{GT}(\mathbf{z}_\ell[i+t_\ell], \eta) \text{ for all } \mathbf{z}'_\ell[i+t_\ell] \neq 0\\ \mathsf{GT}(\eta, \mathbf{z}_\ell[i+t_\ell]) \text{ for all } \mathbf{z}'_\ell[i+t_\ell] = 0\\ \mathbf{z}'_\ell[i] - \mathbf{z}_\ell[i] = 0\end{cases} \tag{6}$$

In our protocol, the prover provides $|\mathbf{z}_\ell[i]|$ and $\mathsf{sign}(\mathbf{z}_\ell[i])$ as the auxiliary witnesses so that the number of constraints reduces from $5n+14$ to $3n+9$ for each $\mathbf{z}_\ell[i+t_\ell]$, where n is the length of the binary representation of $\mathbf{z}_\ell[i+t_\ell]$.

The final step is proving the DWT reconstruction, which is analog to proving the decomposition. Let $\bar{\alpha} \in \mathbb{F}$ be a random challenge chosen by the verifier. The prover can prove DWT reconstruction in (3) such that

$$\sum_{k=1}^{t_\ell}\bar{\alpha}^{\frac{c}{2}+i-2}\hat{\mathbf{x}}_\ell[(2k+1)_{\mathrm{mod}\ t'_\ell}] = \sum_{k=1}^{c/2}\bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{h}}[2k-1]\cdot\sum_{i=1}^{t_\ell}\bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i]$$
$$+\sum_{k=1}^{c/2}\bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{h}}[2k]\cdot\sum_{i=1}^{t_\ell}\bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i+t_\ell] + (\bar{\alpha}^{t_\ell}-1)\cdot\sum_{q=1}^{\frac{c}{2}-1}\bar{\alpha}^{q-1}$$
$$\cdot\sum_{p=1}^{q}(\mathbf{z}'_{\ell-1}[p]\bar{\mathbf{h}}[c-2q+2p] + \mathbf{z}'_{\ell-1}[p+t_\ell]\bar{\mathbf{h}}[c-2q+2p-1])$$
$$\sum_{k=1}^{t_\ell}\bar{\alpha}^{\frac{c}{2}+i-2}\hat{\mathbf{x}}_\ell[(2k)_{\mathrm{mod}\ t'_\ell}] = \sum_{k=1}^{c/2}\bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{g}}[2k-1]\cdot\sum_{i=1}^{t_\ell}\bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i]$$
$$+\sum_{k=1}^{c/2}\bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{g}}[2k]\cdot\sum_{i=1}^{t_\ell}\bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i+t_\ell] + (\bar{\alpha}^{t_\ell}-1)\sum_{q=1}^{\frac{c}{2}-1}\bar{\alpha}^{q-1}$$
$$\cdot\sum_{p=1}^{q}(\mathbf{z}'_{\ell-1}[p]\bar{\mathbf{g}}[c-2q+2p] + \mathbf{z}'_{\ell-1}[p+t_\ell]\bar{\mathbf{g}}[c-2q+2p-1]) \tag{7}$$

We present a toy example in Figure 2 to further explain our split technique. We denote by $\mathbf{x}_{\mathsf{dn}}$ the reconstruction results in the last round of iteration.

4.2.2 *Z-Score Normalization*

Z-Score normalization is one of a widely adopted normalization method in the machine learning literature. It is utilized to adjust the data by shifting and scaling so that all the data samples follow the normal distribution. Z-Score normalization assist the ML pipeline to better extract the features of the dataset and obtain a higher inference accuracy. For each data sample $\mathbf{x}_{\mathsf{dn}}$, Z-Score normalization transform it into $\mathbf{x}_{\mathsf{nl}} \in \mathbb{F}^m$, such that

$$\mathbf{x}_{\mathsf{nl}} = \frac{\mathbf{x}_{\mathsf{dn}} - \nu}{\varkappa} \tag{8}$$

where $\nu, \varkappa$ are the mean and standard deviation of the data, respectively.

Proving Z-Score normalization. The main obstacle in proving the Z-Score normalization is the computation of standard deviation, which incurs a square root operation. More concretely, different from the integers, the square root is nonequivalent to the inverse square computation for fractional numbers in the finite field. To address this issue, we harness the SquareRoot gadget Sqr in proving the standard deviation. We also require the prover to provide

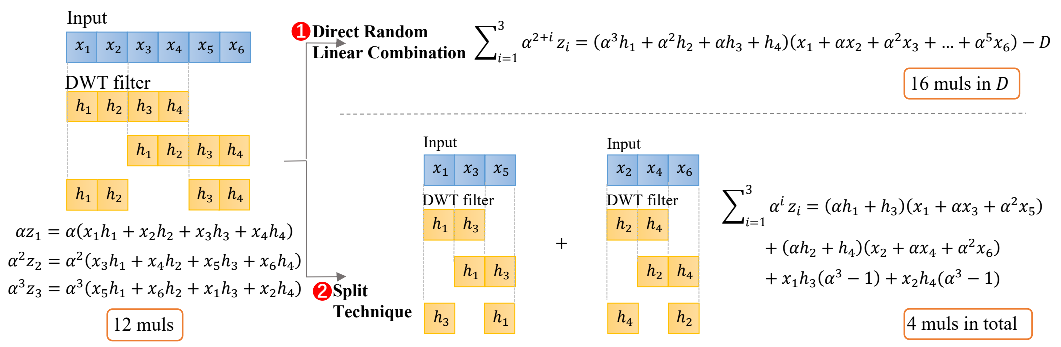

Input
x_1 x_2 x_3 x_4 x_5 x_6
DWT filter
h_1 h_2 h_3 h_4
$\alpha z_1 = \alpha(x_1h_1 + x_2h_2 + x_3h_3 + x_4h_4)$
$\alpha^2 z_2 = \alpha^2(x_3h_1 + x_4h_2 + x_5h_3 + x_6h_4)$
$\alpha^3 z_3 = \alpha^3(x_5h_1 + x_6h_2 + x_1h_3 + x_2h_4)$
12 muls
1 Direct Random Linear Combination
$\sum_{i=1}^{3} \alpha^{2+i} z_i = (\alpha^3 h_1 + \alpha^2 h_2 + \alpha h_3 + h_4)(x_1 + \alpha x_2 + \alpha^2 x_3 + \ldots + \alpha^5 x_6) - D$
16 muls in D
2 Split Technique
$\sum_{i=1}^{3} \alpha^i z_i = (\alpha h_1 + h_3)(x_1 + \alpha x_3 + \alpha^2 x_5)$
$+ (\alpha h_2 + h_4)(x_2 + \alpha x_4 + \alpha^2 x_6)$
$+ x_1 h_3(\alpha^3 - 1) + x_2 h_4(\alpha^3 - 1)$
4 muls in total

Fig. 2: Example of split technique applied to DWT decomposition vs. directly using random linear combination.

$\varkappa' := 1/\varkappa$ as the auxiliary witness, which also needs to be proved. The set of arithmetic constraints to prove (8) is

$$\begin{cases} \nu \cdot m = \sum_{i=1}^{m} \mathbf{x}_{\mathsf{dn},i} \\ \mathsf{Sqr}(\varkappa, \frac{1}{m} \cdot \sum_{i=1}^{m} (\mathbf{x}_{\mathsf{dn},i} - \nu)^2) \\ \mathbf{x}_{\mathsf{nl}} = (\mathbf{x}_{\mathsf{dn}} - \nu) \cdot \varkappa' \\ \varkappa \cdot \varkappa' + a = 1 \\ \mathsf{GT}(s', a) \end{cases} \tag{9}$$

where a is the difference between $\varkappa \cdot \varkappa'$ and 1 due to the truncation error, which is supposed to be less than a small constant s'.

4.2.3 PCA-Based Feature Extraction

PCA [71] is a method to reduce the dimensionality of the data input by representing the most significant characteristics of $\mathbf{x}_{\mathsf{nl}} \in \mathbb{F}^m$ in a smaller feature vector with minimal information loss (i.e., eigenvalues). The PCA training computes a mean vector $\bar{\mathbf{x}} \in \mathbb{F}^m$ for all data samples $\{\hat{\mathbf{x}}_i\}_{i=1}^{N}$ as $\bar{\mathbf{x}} = \frac{\sum_i \hat{\mathbf{x}}_i}{N}$, where N is the number of samples in the training set. A covariance matrix is then computed as $\mathbf{S} = \frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{x}}_i - \bar{\mathbf{x}})(\hat{\mathbf{x}}_i - \bar{\mathbf{x}})^\top$. The PCA training aims at finding eigenvectors $\mathbf{V} = [\mathbf{v}_1^\top, \ldots, \mathbf{v}_m^\top]$ and eigenvalues $(\lambda_1, \ldots, \lambda_m)$ of $\mathbf{S}$ such that $\mathbf{S} \times \mathbf{V} = \mathbf{V} \times \mathbf{\Lambda}$ where $\mathbf{\Lambda} = \mathsf{diag}(\lambda_1, \ldots, \lambda_m)$. To reduce the dimension while retaining the most information about data distribution, we select k eigenvectors $\mathbf{V}' = [\mathbf{v}_{i_1}^\top, \ldots, \mathbf{v}_{i_k}^\top]$ corresponding with k largest eigenvalues $(\lambda_{i_1}, \ldots, \lambda_{i_k})$. To this end, the server retains the eigenvectors $\mathbf{V}'$ and the mean vector $\bar{\mathbf{x}}$ as model parameters. In the inference phase, given a new observation $\hat{\mathbf{x}}$, the feature vector of $\hat{\mathbf{x}}$ can be computed via PCA as

$$\tilde{\mathbf{x}} = (\hat{\mathbf{x}} - \bar{\mathbf{x}}) \times \mathbf{V}' \tag{10}$$

Proving PCA computation. There are $O(m \cdot k)$ constraints in (10), where m is the input dimension and k is the feature vector dimension. We reduce the number of constraints of proving PCA computation from $O(m \cdot k)$ to $O(m)$ using the random linear combination by using the powers of a random challenge chosen by the verifier. This transformation converts variables' multiplication to constant multiplication, where the latter comes for free in R1CS, therefore reducing the computing complexity. Specifically, (10) is equivalent to

$$\tilde{\mathbf{x}}[i] = (\hat{\mathbf{x}} - \bar{\mathbf{x}}) \times \mathbf{V}'[i] \tag{11}$$

where $i \in [1, k]$, $\mathbf{V}'[k]$ is the kth term in $\mathbf{V}'$, e.g., $\mathbf{V}'[k] = \mathbf{v}_{ik}^\top$. Let $\alpha \in \mathbb{F}$ be a random challenge chosen by the verifier. We apply the random linear combination to combine constraints in (11). Specifically, the prover can prove (11) holds by proving that

$$\sum_{i=1}^{k} \alpha^i \tilde{\mathbf{x}}[i] = \sum_{j=1}^{m} \left(\sum_{i=1}^{k} \alpha^i \mathbf{V}'[i] \right) \cdot (\hat{\mathbf{x}}[j] - \bar{\mathbf{x}}[j]) \tag{12}$$

where α^i is the power of the random challenge α computed by the prover, $\mathbf{V}'$ is the eigenvector and $\bar{\mathbf{x}}$ is the mean vector.

4.2.4 SVM Classification

SVM [8] is a supervised ML for classification problems by finding optimal hyperplane(s) that maximizes the separation of the data samples to their potential labels. Suppose the number of samples in the training set is N. Let $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{F}^k$ be the feature vector of data samples and $y_1, \ldots, y_N \in \{1, \ldots, s\}$ be its corresponding label. To deal with data non-linearity, kernel SVM projects $\mathbf{x}_i$ to a higher dimension using a mapping function $\Phi : \mathbb{F}^m \to \mathbb{F}^{m'}$, where $m' > m$ and applies a kernel function $\phi(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ for training and classifying computation. Radial Basis Function (RBF) [8] $\phi_{\mathsf{rbf}}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \cdot ||\mathbf{x}_i - \mathbf{x}_j||^2}$ is the most popular SVM kernel due to its effectiveness.

SVM was initially designed for binary classification, but it can be extended to multiclass classification by breaking down the multiclass problem into multiple *one-to-rest* binary classification problems. For each class $\hat{c}$, data samples are assigned to two classes, where $y_i^{(\hat{c})} = 1$ if $y_i = \hat{c}$, otherwise $y_i^{(\hat{c})} = 0$.

The trainable parameter of SVM is the tuple $(\mathbf{x}_i^{(\hat{c})}, \delta_i^{(\hat{c})}, b^{\hat{c}})$, where for class $\hat{c}$, $\mathbf{x}_i^{(\hat{c})}$ is the support vector, $\delta_i^{(\hat{c})}$ is the coefficient, and $b^{(\hat{c})}$ is the bias. The range of i depends on $|\mathcal{I}^{(\hat{c})}| := |\{i : \delta_i^{(\hat{c})} > 0\}|$, which equals to the number of the support vectors for class $\hat{c}$. Note that $\delta_i^{(\hat{c})} \leq 0$ are dropped during the training. The tuple $(\mathbf{x}_i^{(\hat{c})}, \delta_i^{(\hat{c})}, b^{\hat{c}})$ acts as the secret of the prover, which will be committed to prove the computation.

Given a new observation $\tilde{\mathbf{x}} \in \mathbb{F}^k$, its label y can be predicted as

$$y = \underset{\hat{c}}{\operatorname{argmax}} \sum_{i \in \mathcal{I}^{(\hat{c})}} \delta_i^{(\hat{c})} y_i^{(\hat{c})} \phi(\tilde{\mathbf{x}}, \mathbf{x}_i^{(\hat{c})}) + b^{(\hat{c})} \tag{13}$$

Proving multi-class SVM classification with RBF kernel. Suppose $f^{(\hat{c})} = \sum_{i \in \mathcal{I}^{(\hat{c})}} \delta_i^{(\hat{c})} y_i^{(\hat{c})} \phi(\tilde{\mathbf{x}}, \mathbf{x}_i^{(\hat{c})}) + b^{(\hat{c})}$ is the decision function's evaluation for each class $\hat{c} \in [1, s]$. To prove the SVM classification in (13), we harness Exp and Max gadgets in §4.1 to prove the exponent in the RBF kernel projection, and the class output being the maximum value among all evaluations, respectively. We adopt the representation in [74] where $f^{(\hat{c})}$ is expanded to a value-index pair, i.e., $\mathbf{f} := \{(f^{(1)}, 1), (f^{(2)}, 2), \ldots, (f^{(s)}, s)\}$. Let

$$\bar{\mathbf{f}} := \{(\bar{f}^{(1)}, \sigma(1)), (\bar{f}^{(2)}, \sigma(2)), \ldots, (\bar{f}^{(s)}, \sigma(s))\}$$

be the permutation of $\mathbf{f}$, where $\sigma(\cdot)$ is the permutation function such that $\bar{f}^{(\hat{c})} = f^{(\sigma(\hat{c}))}$ and $\bar{f}^{(1)}$ is the maximum value in $\mathbf{f}$. The prover provides $\bar{\mathbf{f}}$ as the auxiliary witness and shows that the output label $y = \sigma(1)$. Let β be a random challenge from the verifier, the prover binds each value-index pair in $\mathbf{f}$ and $\bar{\mathbf{f}}$ to a single value as

$$p^{(\hat{c})} = f^{(\hat{c})} + \beta \cdot \hat{c}, \quad \bar{p}^{(\hat{c})} = \bar{f}^{(\hat{c})} + \beta \cdot \sigma(\hat{c}) \tag{14}$$

and invokes a permutation check using Perm gadget, where β is a random number chosen by $\mathcal{V}$. Let $l_i^{(\hat{c})} \in \mathbb{F}$ for $i \in \mathcal{I}^{(\hat{c})}, \hat{c} \in [1, s]$, $[\bar{f}^{(1)}, \ldots, \bar{f}^{(s)}]$ be the auxiliary witness used in the gadget Max. Suppose y is the claimed output label and $f^{(y)}$ is the evaluation of the corresponding decision function. Let $\mathbf{p} = \{p^{(\hat{c})}\}$ and $\bar{\mathbf{p}} = \{\bar{p}^{(\hat{c})}\}$ be intermediate vectors, where $p^{(\hat{c})}$ and $\bar{p}^{(\hat{c})}$ are computed by (14), respectively. The set of arithmetic constraints to prove (13) is

$$\begin{cases} k_i^{(\hat{c})} = -\gamma ||\tilde{\mathbf{x}} - \mathbf{x}_i^{(\hat{c})}||^2 \text{ for } i \in \mathcal{I}^{(\hat{c})}, \hat{c} \in [1, s] \\ f^{(\hat{c})} = \sum_{i \in \mathcal{I}^{(\hat{c})}} \delta_i^{(\hat{c})} y_i^{(\hat{c})} l_i^{(\hat{c})} + b^{(\hat{c})} \text{ for } \hat{c} \in [1, s] \\ \mathsf{Exp}(l_i^{(\hat{c})}, e, k_i^{(\hat{c})}) \text{ for } i \in \mathcal{I}^{(\hat{c})}, \hat{c} \in [1, s] \\ \mathsf{Max}(f^{(y)}, [f^{(1)}, \ldots, f^{(s)}]) \\ \mathsf{Perm}(\mathbf{p}, \bar{\mathbf{p}}) \\ f^{(y)} + \beta \cdot y = \bar{p}^{(1)} \end{cases} \tag{15}$$

Proving other SVM kernels. The RBF kernel achieves competitive inference results, it requires heavy computational resources for training all the model parameters. More importantly, the amount of support vectors significantly influences the size of the arithmetic circuits. Hence a large number of support vectors may lead to high proving/verification time and large proof size. Fortunately, the RBF kernel can be substituted by other SVM kernels for some datasets when the distribution margins are clear or the number of samples is large. These SVM kernels, e.g., the polynomial kernel and Sigmoid kernel, are easier to be proved than the RBF yet may cause lower accuracy [14]. Our techniques can be also used to prove those kernels. Let $c \in \mathbb{F}$ be the output of the kernel function. We present the constraints for other SVM kernels as follows.

- *Polynomial kernel.* $\phi_{\mathsf{ply}}(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + \alpha)^\beta$ can be proven with the following constraints

$$\begin{cases} \gamma \mathbf{x}_i^T \mathbf{x}_j + \alpha = b \\ \mathsf{Exp}(b, \beta, c) \end{cases} \tag{16}$$

  where $b \in \mathbb{F}$ is the intermediate value.
- *Laplace kernel.* $\phi_{\mathsf{la}}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma' ||\mathbf{x}_i - \mathbf{x}_j||}$ can be proven with the following constraints

$$\begin{cases} b = -\gamma' ||\mathbf{x}_i - \mathbf{x}_j|| \\ \mathsf{Exp}(c, e, b) \end{cases}$$

  where $b \in \mathbb{F}$ is intermediate value.
- *Sigmoid kernel.* $\phi_{\mathsf{sig}}(\mathbf{x}_i, \mathbf{x}_j) = tanh[\alpha(\mathbf{x}_i^T \mathbf{x}_j) - \beta]$, where $\alpha, \beta > 0$ are hyper-parameters, can be proven with following constraints

$$\begin{cases} b = \alpha(\mathbf{x}_i^T \mathbf{x}_j) - \beta \\ \mathsf{Exp}(a_1, e, b) \\ a_1 \cdot a_2 = 1 \\ c \cdot (a_1 + a_2) = a_1 - a_2 \end{cases}$$

where $b \in \mathbb{F}$ is the intermediate value, and $a_1, a_2 \in \mathbb{F}$ are auxiliary witnesses.

RBF kernel approximation. Although some datasets can alternatively use simpler kernels (e.g., polynomial kernel as previously described), these kernels are insufficient in complicated datasets and may result in low accuracy. To this end, we further introduce an RBF kernel approximation method as a trade-off between accuracy and proving efficiency. Specifically, we adopt the Nyströem method [63] to approximate the RBF kernel in SVM. The Nyströem method is generally used for low-rank approximations of kernels. The key insight is to carry out an eigendecomposition on a small subset of the data and then expand the results back to the original dimension. More concretely, the kernel-based SVM projects each data sample $\mathbf{x}_i$ to a higher dimension via a kernel function $\phi(\mathbf{x}_i, \mathbf{x}_j)$ such that

$$\phi(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = \sum_k^{N'} \lambda_k \Phi_k(\mathbf{x}_i) \cdot \Phi_k(\mathbf{x}_j) \tag{17}$$

where $N < N' \leq \infty$. We slightly abuse the notation λ_i to denote the eigenvalues of $\Phi(\cdot)$. Equation (17) can be then converted to

$$\int \phi(\mathbf{x}_j, \mathbf{x}_i) \Phi_k(\mathbf{x}_i) p(\mathbf{x}_i) d\mathbf{x}_i = \lambda_k \Phi_k(\mathbf{x}_j) \tag{18}$$

where $p(\mathbf{x}_i)$ is the probability density of the input vector. To approximate Equation (18), Nyströem method replaces the integral over $p(\mathbf{x}_i)$ by an empirical average given an i.i.d. sample $\{\mathbf{x}_1, ..., \mathbf{x}_{N''}\}, N'' < N'$ so that

$$\frac{1}{N''} \sum_m^{N''} \phi(\mathbf{x}_j, \mathbf{x}_m) \Phi_i(\mathbf{x}_m) \approx \lambda_k \Phi_i(\mathbf{x}_j) \tag{19}$$

Given a new observation $\mathbf{x}_{\mathsf{fe}} \in \mathbb{F}^k$, the prediction procedure is the same as Equation (13). However, due to the approximation only adopting a part of the training samples, the number of support vectors is smaller than the plain SVM. Previous works have proved the effectiveness of the Nyströem method [70]. The kernel in the approximation can be heterogeneous. In our work, we choose the RBF Nyströem approximation because of its high accuracy.

Protocol 1 (ezDPS). *Let λ be the security parameter.*

- $\mathsf{pp} \leftarrow \mathsf{ezDPS}.\mathcal{G}(1^\lambda)$: *Output* $\mathsf{pp} \leftarrow \mathsf{zkp}.\mathcal{G}(1^\lambda)$
- $\hat{\mathsf{cm}} \leftarrow \mathsf{ezDPS.Com}(\mathbf{w}, r, \mathsf{pp})$: *Let* $\mathbf{w} = (\mathbf{h}, \mathbf{g}, \bar{\mathbf{h}}, \bar{\mathbf{g}}, \eta, \bar{\mathbf{x}}, \mathbf{V}', \{\mathbf{x}_i, \delta_i^{(\hat{c})}, b^{(\hat{c})}\}_{i\in\mathcal{I}^{(\hat{c})},\hat{c}\in[1,s]}, \gamma)$. *Compute* $\hat{\mathsf{cm}} \leftarrow \mathsf{zkp.Com}(\mathbf{w}, r, \mathsf{pp})$, *where r is randomness chosen by the server.*
- $(y, \pi) \leftarrow \mathsf{ezDPS}.\mathcal{P}(\mathbf{w}, \mathbf{x}_{\mathsf{in}}, \mathsf{pp})$:
  1) *The server executes Algorithm 1 to compute* $y \leftarrow \mathsf{DPS}(\mathbf{w}, \mathbf{x}_{\mathsf{in}})$, *and commits to all the auxiliary witnesses* aux *in (5), (6) (7), (9), (12), (15) as* $\mathsf{cm}' \leftarrow \mathsf{zkp.Com}(\mathsf{aux}, r', \mathsf{pp})$ *under randomness r' chosen by the server.*
  2) *Upon receiving the randomness $\vec{\alpha}$ chosen by the client for checking the random linear combination and maximum value, the server invokes backend ZKP protocol to get the proof as* $\pi \leftarrow \mathsf{zkp}.\mathcal{P}((\mathbf{w}, \mathsf{aux}), \mathbf{x}_{\mathsf{in}}, y, \mathsf{pp})$. *The server sends (y, π) to the client.*
- $b \leftarrow \mathsf{ezDPS}.\mathcal{V}(\mathsf{cm}, \mathbf{x}_{\mathsf{in}}, y, \pi, \mathsf{pp})$: *Let* $\mathsf{cm} = (\hat{\mathsf{cm}}, \mathsf{cm}')$, *the client invokes* $b \leftarrow \mathsf{zkp}.\mathcal{V}(\mathsf{cm}, \mathbf{x}_{\mathsf{in}}, y, \pi, \mathsf{pp})$ *and outputs b.*

Fig. 3: Our ezDPS Protocol.

4.2.5 Putting Everything Together

We combine everything together and present the complete algorithmic description of our ezDPS scheme in Protocol 1. We describe the functionality (Algorithm 1) that processes a data sample $\mathbf{x}_{\mathsf{in}} \in \mathbb{F}^m$ with DWT (Figure 4, lines 1-15), ZS (line 16), PCA (line 17), and SVM (lines 18-21), and returns an inference result y.

4.2.6 Zero-Knowledge Proof of Accuracy

We construct a zkPoA scheme that is derived from the inference of individual samples to attest to the effectiveness of the committed model by demonstrating its accuracy over public dataset $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_M)$ with ground truth labels $\mathbf{T} = (t_1, \dots, t_M)$. zkPoA requires the server to commit to a model with claimed accuracy on public sources. Once the model is committed and zkPoA is generated, it cannot be altered. The server has to use the model that has been committed previously for the successive inference tasks. Let $\mathbf{Y} = (y_1, \dots, y_M)$ be the predicted labels of $\mathcal{D}$, where $y_i \leftarrow \mathsf{DPS}(\mathbf{w}, \mathbf{x}_i)$ for $i \in [1, M]$. The accuracy of MLIP model over $\mathcal{D}$ is $\psi = \frac{\Sigma_{i=1}^{M}(y_i \overset{?}{=} t_i)}{M}$ where $0 \leq \psi \leq 1$.

In our zkPoA, it suffices to show the committed model maintains *at least* ψ accuracy (rather than the precise number) by proving that at least $\psi \cdot M$ samples are classified correctly. This reduces the complexity since the prover does not have to prove some samples are misclassified (which incurs complex circuits for proof of inequality). Our zkPoA is as follows.

We expand $\mathbf{Y}$ and $\mathbf{T}$ to value-index pairs as $\mathbf{Y} = \{(y_1, 1), \dots, (y_M, M)\}$, $\mathbf{T} = \{(t_1, 1), \dots, (t_M, M)\}$. The prover shuffles $\mathbf{Y}$ and $\mathbf{T}$ to $\mathbf{Y}'$ and $\mathbf{T}'$ using permutation functions σ_1, σ_2, respectively, which have two goals: (i) hide which samples are classified correctly, and (ii) reduce the computation cost by rearranging correctly classified samples as first items in $\mathbf{Y}'$ and $\mathbf{T}'$. Therefore, $\mathcal{P}$ needs to prove: (i) first $\psi \cdot M$ items in $\mathbf{Y}'$ and $\mathbf{T}'$ are identical, (ii) $\mathbf{Y}'$ (resp. $\mathbf{T}'$) is a permutation of $\mathbf{Y}$ (resp. $\mathbf{T}$), and (iii) two permutations are the same.

Suppose the permuted sets are $\mathbf{Y}' = \{(y'_1, \sigma_1(1)), \dots, (y'_M, \sigma_1(M))\}$ and $\mathbf{T}' = \{(t'_1, \sigma_2(1)), \dots, (t'_M, \sigma_2(M))\}$, where $y'_i = y_{\sigma_1(i)}$ and $t'_i = t_{\sigma_2(i)}$. The prover provides $\mathbf{Y}'$ and $\mathbf{T}'$ as the auxiliary witnesses. Let ξ be a random challenge chosen by the verifier. To perform the permutation test, $\mathcal{P}$ computes intermediate values $\tilde{\mathbf{Y}} = \{\tilde{y}_i\}, \bar{\mathbf{Y}} = \{\bar{y}_i\}, \tilde{\mathbf{T}} = \{\tilde{t}_i\}$ and $\bar{\mathbf{T}} = \{\bar{t}_i\}$ such that for each $i \in [1, M]$:

$$\tilde{y}_i = y_i + \xi \cdot i \quad \text{and} \quad \bar{y}_i = y'_i + \xi \cdot \sigma_1(i)$$
$$\tilde{t}_i = t_i + \xi \cdot i \quad \text{and} \quad \bar{t}_i = t'_i + \xi \cdot \sigma_2(i)$$

The set of constraints for our zkPoA includes all the constraints to prove each y_i plus the following constraints

$$\begin{cases} y'_i - t'_i = 0 \text{ for } i \in [1, \psi \cdot M] \\ \sigma_1(i) - \sigma_2(i) = 0 \text{ for } i \in [1, M] \\ \mathsf{Perm}(\tilde{\mathbf{Y}}, \bar{\mathbf{Y}}) \\ \mathsf{Perm}(\tilde{\mathbf{T}}, \bar{\mathbf{T}}) \end{cases}$$

Algorithm 1 ($y \leftarrow \mathsf{DPS}(\mathbf{w}, \mathbf{x}_{\mathsf{in}})$).
Input: Data sample $\mathbf{x}_{\mathsf{in}} \in \mathbb{F}^m$, MLIP model parameters $\mathbf{w} = (\mathbf{h}, \mathbf{g}, \bar{\mathbf{h}}, \bar{\mathbf{g}}, \eta, \bar{\mathbf{x}}, \mathbf{V}', \{\mathbf{x}_i, \delta_i^{(\hat{c})}, b^{(\hat{c})}\}_{i\in\mathcal{I}^{(\hat{c})},\hat{c}\in[1,s]}, \gamma)$
Output: Inference result y.

```
1:  for $\ell = 1$ to $d$ do
2:      $t_\ell \leftarrow \frac{m}{2^\ell}$ and $t'_\ell \leftarrow \frac{m}{2^{\ell-1}}$
3:      for $i = 1$ to $t_\ell$ do
4:          $\mathbf{z}_\ell[i] \leftarrow \sum_{j=1}^{c} \mathbf{h}[j] \cdot \mathbf{z}_{\ell-1}[(2i + j - 2)_{\bmod t'_\ell}]$
5:          $\mathbf{z}_\ell[i + t_\ell] \leftarrow \sum_{j=1}^{c} \mathbf{g}[j] \cdot \mathbf{z}_{\ell-1}[(2i + j - 2)_{\bmod t'_\ell}]$
6:      for $i = 1$ to $t_\ell$ do
7:          $\mathbf{z}'_\ell[i] \leftarrow \mathbf{z}_\ell[i]$
8:          if $|\mathbf{z}_\ell[i + t_\ell]| - \eta > 0$ then
9:              $\mathbf{z}'_\ell[i + t_\ell] \leftarrow \mathsf{sign}(\mathbf{z}_\ell[i + t_\ell])(\mathbf{z}_\ell[i + t_\ell] - \eta)$
10:         else
11:             $\mathbf{z}'_\ell[i + t_\ell] \leftarrow 0$
12:     for $i = 1$ to $t_\ell$ do
13:         $\hat{\mathbf{x}}_\ell[2i - 1] \leftarrow \sum_{j=1}^{c/2} (\bar{\mathbf{h}}[2j - 1] \cdot \mathbf{z}'_\ell[(i + j - 1)_{\bmod t_\ell}]$
                $+ \bar{\mathbf{h}}[2j] \cdot \mathbf{z}'_\ell[t_\ell + (i + j - 1)_{\bmod t_\ell}])$
14:         $\hat{\mathbf{x}}_\ell[2i] \leftarrow \sum_{j=1}^{c/2} (\bar{\mathbf{g}}[2j - 1] \cdot \mathbf{z}'_\ell[(i + j - 1)_{\bmod t_\ell}]$
                $+ \bar{\mathbf{g}}[2j] \cdot \mathbf{z}'_\ell[t_\ell + (i + j - 1)_{\bmod t_\ell}])$
15: $\mathbf{x}_{\mathsf{dn}} = \hat{\mathbf{x}}_{t_l}$
16: $\mathbf{x}_{\mathsf{nl}} = (\mathbf{x}_{\mathsf{dn}} - \nu)/\varkappa$
17: $\mathbf{x}_{\mathsf{fe}} \leftarrow (\mathbf{x}_{\mathsf{nl}} - \bar{\mathbf{x}})\mathbf{V}'$
18: for $\hat{c} = 1$ to $s$ do
19:     Let $\mathcal{I}^{(\hat{c})} = \{i : \delta_i^{(\hat{c})} > 0\}$
20:     $y_{\hat{c}} \leftarrow \sum_{i\in\mathcal{I}^{(\hat{c})}} \delta_i^{(\hat{c})} y_i^{(\hat{c})} \phi(\mathbf{x}_{\mathsf{fe}}, \mathbf{x}_i) + b^{(\hat{c})}$
21: $y_c \leftarrow \max(y_1, \dots, y_s)$
22: return $c$
```

Fig. 4: MLIP with DWT, Z-Score, PCA and SVM algorithms.

5 Analysis

Complexity. Let m, k be the dimensions of the raw data sample and the feature vector by PCA, respectively. Let s, t be the number of SVM classes and the number of support vectors for all classes, respectively. In DWT, our scheme requires $8 \log_2 \frac{2m}{c}$ constraints for DWT decomposition (5) and reconstruction (7), while the thresholding (2) incurs $(3n + 9)(m - \frac{c}{2})$ constraints, where n is the size (in bits) of each value per dimension of the raw data sample, c is the dimension of the high-pass and low-pass filters. In total, our scheme requires $16 + (3n + 9)(m - \frac{c}{2})$ constraints for proving DWT. In Z-Score normalization, the number of constraints is $2(m + n_1 + n_2) + 11$, where n_1, n_2 are the different binary vector lengths to reduce the size of the arithmetic circuit. In the real implementation, n_2 is four times larger than n_1 since the scalars are scaled repeatedly during the MLIP computation. In PCA, the number of constraints is m (12). This is reduced from mk compared with direct proving (10) due to random linear combination. In SVM classification (15), the number of constraints is composed of three parts: constraints for kernel projection, decision function, and the final classification. More concretely, ezDPS incurs t constraints for the decision function and $(3n + 6)(s - 1) + 2s$ constraints for proving the classification. The permutation trick in our proposed Max gadget permits us to reduce the number of comparisons from $O(s^2)$ in generic circuits to $O(s)$. The constraints for kernel projection rely on the concrete kernel adopted. For RBF kernel (both in plain SVM and Nyströem approximation), ezDPS requires $(2n + k)t + 2s$ constraints. For polynomial kernel, ezDPS invokes $(15 + k)t(s - 1)$ constraints in total.

For zkPoA, suppose the number of samples in the testing dataset is M, and proving one testing data incurs N constraints. Therefore, our zkPoA incurs $(N + 4)M$ constraints for proving the accuracy.

Security. We analyze the security of our scheme. Specifically, we have the following theorem.

Theorem 2. *Our* ezDPS *scheme in Protocol 1 is a zero-knowledge MLIP as defined in Definition 1 given that the backend CP-ZKP is secure by Theorem 1.*

Proof of Theorem 2. We argue the completeness, soundness, and zero-knowledge properties of our scheme as follows.

Completeness. The circuit in ezDPS.$\mathcal{P}$ outputs 1 if y is the correct inference label of data sample $\mathbf{x}$ by Figure 4 on MLIP parameters $\mathbf{w}$. The correctness of our protocol in Figure 3 follows the correctness of the backend ZKP protocol by Theorem 1.

Soundness. Let C be the arithmetic circuit that represents the computation of MLIP with DWT, PCA, and SVM. By the extractability of commitment used by the backend ZKP, there exists an extractor $\mathcal{E}$ such that given cm, it extracts a witness w^* such that $\mathsf{cm} = \mathsf{zkp.Com}(w^*, r, \mathsf{pp})$ with overwhelming probability. By the soundness of zkMLIP in Definition 1, if $\mathsf{cm} = \mathsf{zkMLIP.Com}(\mathbf{w}, \mathsf{pp}, r)$ and $\mathsf{zkMLIP}.\mathcal{V}(\mathsf{cm}, \mathbf{x}, y, \pi, \mathsf{pp}) = 1$ but $y \neq \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x})$, then there are two scenarios:

- Scenario 1: $w^* = (\mathbf{w}, \mathsf{aux})$ satisfying to $C((\mathsf{cm}, \mathbf{x}, y, \mathbf{r}'); w^*) = 1$. There are three cases for this to happen: (i) $\mathbf{w}$ is not the one committed to cm but passing the verification for cm; (ii) y is not the class label corresponding with the maximum predicted value among the auxiliary witnesses $(f^{(1)}, \ldots, f^{(s)}) \in \mathsf{aux}$ in (15), but passing the max and permutation test; (iii) Some witnesses in aux are not valid, but passing the random linear combination test. The probability of the first case is negligible in λ due to the soundness of the commitment scheme used by the backend ZKP protocol. As Max gadget relies on the permutation test, its soundness error is negligible in λ due to the soundness of the characteristic polynomial check, which achieves the probability of $s/|\mathbb{F}|$ due to Schwartz-Zippel Lemma [58]. Finally, the soundness error of the random linear combination over a small number of constraints is negligible in λ. By the union bound, the probability that $\mathcal{P}$ can generate such w^* is $\mathsf{negl}(\lambda)$.
- Scenario 2: $w^* = (\mathbf{w}, \mathsf{aux})$ and $C((\mathsf{cm}, \mathbf{x}, y, \vec{\alpha}); w^*) = 0$. According to the soundness of the backend ZKP, given a commitment cm*, the probability that $\mathcal{A}$ can generate a proof π_w making $\mathcal{V}$ accept the incorrect witness is negligible in λ.

In overall, the soundness of ezDPS holds except with a negligible probability in λ.

Zero Knowledge. We construct a simulator for Protocol 1 in Figure 5 and show that the following hybrid game is indistinguishable.

- **Hybrid** H_0: H_0 behaves as the honest prover in Protocol 1.
- **Hybrid** H_1: H_1 uses the real ezDPS.Com() in Protocol 1, for the commitment phase, and invokes $\mathcal{S}$ to simulate the proving phase.
- **Hybrid** H_2: H_2 behaves as Simulator 1.

Given the same commitment, the verifier cannot distinguish H_0 and H_1 due to the zero-knowledge property of the backend zero-knowledge protocol, given the same circuit C and public input. If the verifier can distinguish H_1, and H_2, we can find a PPT adversary $\mathcal{A}$ to distinguish whether a commitment of an MLIP with zero strings or not, which is contradictory with the hiding property of the underlying commitment scheme. Thus, the verifier cannot distinguish H_0 from H_2 by the hybrid, which completes the proof of zero-knowledge. □

Simulator 1 (Simulation of Protocol 1). *Let λ be the security parameter, $\mathbb{F}$ be a finite field, $\mathbf{w}$ with n values. Let* $\mathsf{pp} \leftarrow \mathsf{ezDPS}.\mathcal{G}(1^\lambda)$.

- $\hat{\mathsf{cm}} \leftarrow \mathcal{S}_1(n, r, \mathsf{pp})$: *$\mathcal{S}_1$ invokes $\mathcal{S}_{\mathsf{zkp}}$ to generate* $\hat{\mathsf{cm}} = \mathcal{S}_{\mathsf{zkp}}(n, r, \mathsf{pp})$ *where r is randomness generated by* $\mathcal{S}_{\mathsf{zkPC}}$.
- $(y, \pi) \leftarrow \mathcal{S}_2^{\mathcal{A}}(\mathbf{w}, \mathbf{x}, \mathsf{pp})$: *$\mathcal{S}_2$ queries the oracle to get* $y \leftarrow \mathsf{DPS}(\mathbf{w}, \mathbf{x})$. *$\mathcal{S}_2$ shares all public input of C to $\mathcal{S}_{\mathsf{zkp}}$ and invokes* $\mathsf{cm}_w \leftarrow \mathcal{S}_{\mathsf{zkp}}.\mathsf{Com}(\mathsf{pp})$. *Upon receiving randomness $\vec{\alpha}$ from $\mathcal{A}$, $\mathcal{S}_2$ invokes* $\pi \leftarrow \mathcal{S}_{\mathsf{zkp}}.\mathcal{P}(C, (\hat{\mathsf{cm}}, \mathbf{x}, y, \vec{\alpha}), \mathsf{pp})$, *and sends π to $\mathcal{A}$.*
- $b \leftarrow \mathcal{A}(\mathsf{cm}, \mathbf{x}, y, \pi, \mathsf{pp})$: *Let* $\mathsf{cm} = (\hat{\mathsf{cm}}, \mathsf{cm}_w)$, *wait $\mathcal{A}$ for validation.*

Fig. 5: Simulator of Protocol 1.

6 Implementation

We fully implemented our proposed framework in Python and Rust, consisting of approximately 2,500 lines of code in total. For DWT, we implemented the Daubechies DB4 algorithm [67]. We used `sklearn` [53] to implement the Z-Score normalization and the training phase of PCA and SVM. On the other hand, we implemented the inference phase of PCA and SVM from scratch to obtain all the witnesses to generate the proofs. We used fixed-point number representation for all the values being processed in our framework. Each value can be represented by 64 bits, which reserves 1 bit for the sign, 31 bits for the integer part, and 32 bits for the fractional part. We set the dimension of the binary vectors in DWT to be 64, which permits our zkMLIP scheme to be more applicable for various scenarios.

We used the exponent gadget to prove the RBF kernel of the form $e^{\gamma||\mathbf{x}_i-\mathbf{x}_j||^2}$, where the base e^{γ} is public and the exponent $||\mathbf{x}_i - \mathbf{x}_j||^2$ is secret (witness). As shown in §4.1, our gadget precomputes $a^{2^{i-1}}$, where $a = e^{-\gamma}$ and i is the index of the binary representation of the exponent. To reduce the size of the arithmetic circuit for the RBF kernel, we use the polynomial kernel and Nyströem approximation, respectively, corresponding to the characteristics of the datasets. We used a fixed-point arithmetic to represent the exponent. Since it suffices to set $\gamma = 10^{-3}$ for RBF kernel, we used 20 bits to represent the fractional part of the exponent, which suffices to cover most of the cases in our test set. There are few samples that cause the fractional part of the exponent to exceed 20 bits. In this case, we truncated the fractional part of the witness that exceeds 20 bits, leading to a small accuracy loss (see §7.4). The scalars in the finite field grow larger when computation accumulates. We set $n_1 = 24$ and $n_2 = 86$ in proving the Z-score normalization.

In our implementation, we transformed the arithmetic constraints and the witnesses generated from ML algorithms into R1CS relations using the compact encoding method in `libspartan` [60] and then invoked its library APIs to create proofs and verification. Concretely, we used $\texttt{Spartan}_{\texttt{DL}}$ scheme, which implements *(i)* Hyrax polynomial commitment [68], *(ii)* `curve25519-dalek` [29] for curve arithmetic in prime order ristretto group, *(iii)* a separate dot-product proof protocol for each round of the sum-check protocol for zero-knowledge property, and *(iv)* `merlin` [13] for non-interactive proof via Fiat-Shamir transformation. we adapted Spartan to the commit-and-prove paradigm. The high level is to split the commitment in the proof generation of Spartan into two parts in such a way that once the client verifies the proof, they can be combined together. More concretely, let $\mathbf{w}_1 \in \mathbb{F}^P$ be the model parameter and $\mathbf{w}_2 \in \mathbb{F}^Q$ be the witness being generated during inference computation. The prover commits to the model as $\mathsf{cm}_1 = \mathsf{Com}(\mathbf{w}'_1, r_1)$, where $\mathbf{w}'_1 = (\mathbf{w}_1||0^Q)$ and $||$ denotes vector concatenation. During the inference computation, the server generates the proof π using the model and the witness $\mathbf{w} = (\mathbf{w}_1||\mathbf{w}_2)$, and shares with the client the commitment of the witness computed as $\mathsf{cm}_2 = \mathsf{Com}(\mathbf{w}'_2, r_2)$, where $\mathbf{w}'_2 = (0^P||\mathbf{w}_2)$. As π is generated based on $(\mathbf{w}_1||\mathbf{w}_2)$, verifying π requires the commitment $\mathsf{cm} = \mathsf{Com}(\mathbf{w}_1||\mathbf{w}_2)$, which the client can obtain by computing the homomorphic addition as $\mathsf{cm} = \mathsf{cm}_1 \boxplus \mathsf{cm}_2$. Our implementation is available at

https://github.com/vt-asaplab/ezDPS/tree/extended_ezDPS

7 Experimental Evaluation

In this section, we evaluate the performance of ezDPS in comparison with its counterparts on three public datasets that are suitable for the classical ML inference pipeline. Our main evaluation metrics are the proving time, verification time, proof size, and accuracy loss. More concretely, we seek to answer the following main questions.

- How is the performance of our ezDPS compared with the generic circuits on three public datasets? (§7.2)
- How is the detailed cost of each stage in ezDPS framework? (§7.3)
- What is the accuracy loss of ezDPS compared with the floating-point arithmetic, and what is the accuracy of Nyströem approximation method? (§7.4)

We first describe the configuration and methodology to conduct our experiments as follows.

TABLE 2: Detailed model parameters.

KDD 1999				UCR-ECG				British Birdsong			
m	k	s	t	m	k	s	t	m	k	s	t
30	20	4	55	750	57	4	173	169	120	8	150
30	20	8	126	750	60	8	300	169	40	16	270
30	22	16	750	750	56	16	360	169	140	32	450
30	23	23	1999	750	22	32	390	169	125	64	390
				750	86	42	420	169	90	88	330

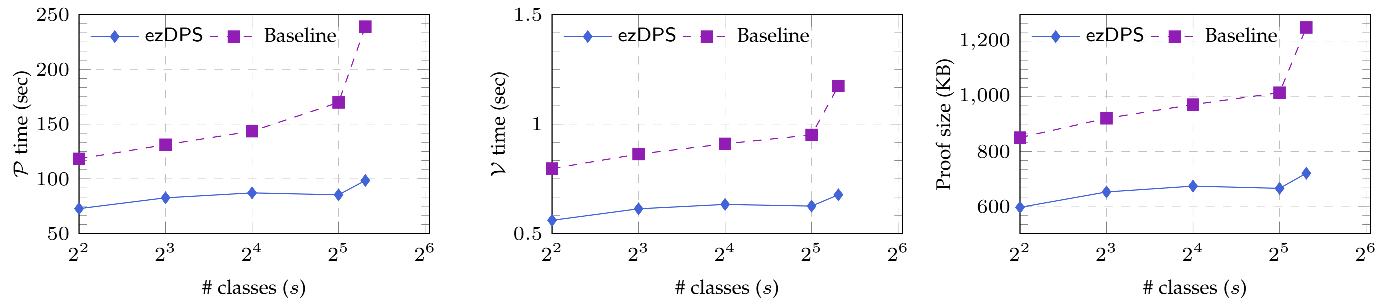
m: dimension of raw data, k: dimension of feature vector by PCA, s: number of distinct class labels, t: number of support vectors in all classes.

7.1 Configuration

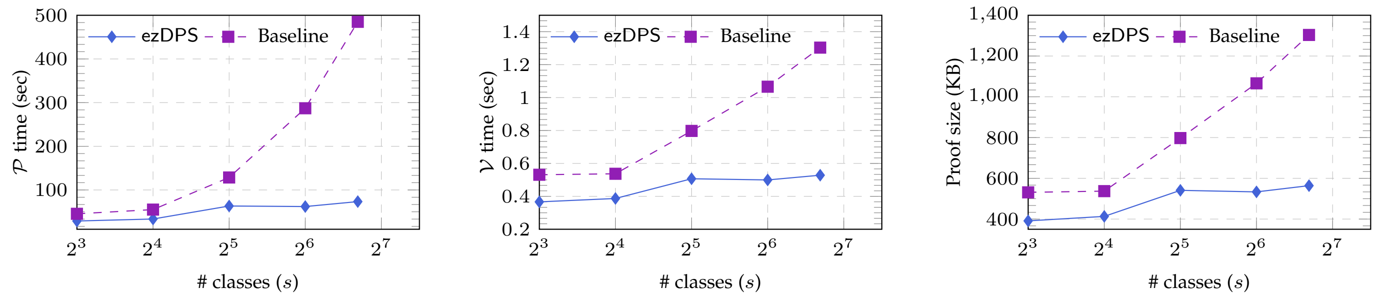
Hardware. We ran all the experiments on a 2020 Macbook Pro, which was equipped with a 2.0 GHz 4-core Intel Core i5 CPU, 16GB DDR4 RAM. Currently, we do not use thread-level parallelization to accelerate the proving/verification time. The experimental results reported in this section are with single-thread computation, which can be further improved once multi-thread parallelization is employed.

Dataset. We evaluated our scheme on three public datasets, including the KDD CUP 1999 dataset [62], ECG dataset in UCR Time Series Classification Archive (UCR-ECG) [12], and the British Birdsong [61]. KDD CUP 1999 is a popular network intrusion dataset used for The Third International Knowledge Discovery and Data Mining Tools Competition, which contains 395,216 data samples for 23 classes. UCR-ECG contains 1800 records of ECG signals, each being of length 750. British Birdsong contains 88 species of birdsong, each audio is of length 169. We used the subset of each dataset for the different number of classes. We selected these datasets because they are more appropriate for the classic machine learning pipelines. The models and our zkMLIP scheme have been proven effective on these datasets, which will be demonstrated in the following sections.

Parameters. We used standard parameters as suggested in Spartan [59] (e.g., curve25519) for 128-bit security. We evaluated the performance of our proposed methods with varied numbers of classes (s) and PCA dimensions (k) (see Table 2). For DWT processing, we set the number of recursion levels to be 1 for noise reduction and $\eta = 0.02$ for processing the detail coefficients. For the Z-Score normalization, we set s in the SquareRoot gadget as 10^{-4}, and s' as 10^{-5}. For PCA, we selected the number of eigenvectors k such that they can capture at least 90% of the variance. We presented the concrete number of k w.r.t different sizes of the datasets in Table 2. Finally, for SVM classification, we use the polynomial kernel to substitute the RBF kernel for the KDD 1999 dataset because the number of data samples is sufficient. We set the bias of the polynomial kernel $\alpha = 0$ and $n = 4$ in the Exponent gadget. Alternatively, we adopt the Nyströem approximation method to replace the RBF, which significantly reduces the number of support vectors.



(a) UCR-ECG



(b) British Birdsong

Fig. 6: Performance of our scheme compared with the baseline.

Counterpart comparison. To our knowledge, we are the first to propose a zero-knowledge MLIP. There is also no prior work that suggests zero-knowledge proof for each of the ML algorithms (i.e., DWT, Z-Score, PCA, and SVM) in our framework. Thus, we chose to compare with the naïve approach, in which we hardcore the whole DWT, Z-Score, PCA, and SVM computation into the circuit and ran the same CP-ZKP backend (i.e., Spartan). We compared ezDPS with this baseline to demonstrate our advantage in reducing the proving time, verification time, and proof size. We also report the accuracy of ezDPS to demonstrate the advantage of ML pipeline processing.

Evaluation metrics. We assess the performance of our scheme and the baseline approach in terms of proving time, verification time and proof size (§7.2 and §7.3). We use the given training and testing data in UCR-ECG, and adopt the standard train-test split method in sklearn to separate the training and testing samples. We set the testing ratio as 0.2.

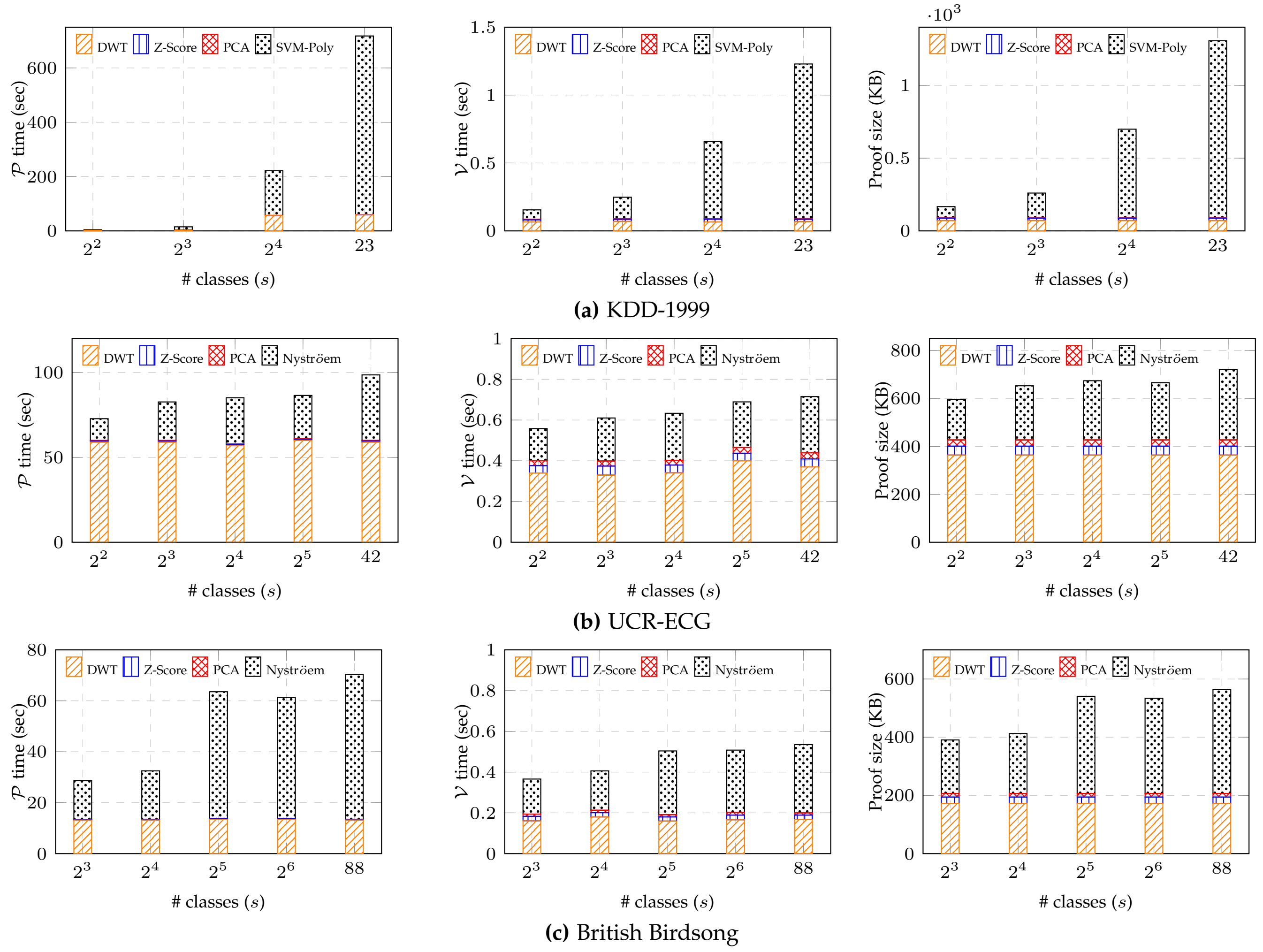
7.2 Overall Results

ezDPS is two to four times more efficient than the baseline in *all* metrics. More concretely, on the KDD-1999 dataset, our proving time is from 4.8 to 660 seconds for 4 to 23 classes, respectively, which is 2 to 30 seconds less than the baseline method. The use of the polynomial kernel in KDD-1999 significantly reduces the number of support vectors. For example, $t = 1999$ for 23 classes using the polynomial kernel, and $t = 11,548$ without the kernel optimization. The gap between our ezDPS and the baseline is more significant on UCR-ECG and British Birdsong datasets. For example, on the UCR-ECG dataset, our proving time is from 72 to 98 seconds while the baseline method requires 118 to 239 seconds. For 8 classes on the British Birdsong dataset, our method incurs 28 seconds of proving time and the generic circuits require 46 seconds. For 88 classes, the proving time using ezDPS is 70 seconds while it costs over 480 seconds if using the baseline method. ezDPS can achieve over 1000× faster in the proving time for larger datasets, e.g., the image dataset LFW [28] we adopted in our prior work [23]. The verification time and proof size follow the same trend as the proving time. Specifically, ezDPS incurs 0.156 to 1.244 seconds for verification and 166 to 1306 KB of the proof size on KDD-1999. As for the baseline, the verification time is from 0.195 to 0.686 seconds and the proof size is from 209 to 1367 KB. Our ezDPS achieves 2 times better performance than the generic circuit on British Birdsong and UCR-ECG. For instance, ezDPS requires 0.528 seconds in verification and 564 KB of proof size for 88 classes on British Birdsong, compared with 1.303 seconds and 1391 KB using the baseline method. We present the performance comparison between our ezDPS and the baseline approach in terms of proving time, verification time, and proof size, in UCR-ECG and British Birdsong with different model sizes.

We can see the verification and bandwidth in ezDPS are highly efficient, i.e., less than one second and 5 MB, respectively, compared with the proving. This is because we use Spartan as the CP-ZKP backend, which offers sublinear verification and proof size overhead.

The concrete end-to-end computation latency and communication in Figure 6 also confirm the efficiency improvement of our optimization techniques. By introducing the split technique and employing the random linear combination, the complexity is reduced from $O(mc + mk)$ to $O(c^2 + m)$, where c is a very small constant in practice (e.g., $c = 4$ for Daubechies DB4 DWT). The most significant improvement in the overall cost is achieved when the number of classes is large. That is due to the employment of Max and Exp gadgets in the SVM phase, which reduces the complexity from $O(s^2)$ to $O(s)$, which also explains the relatively small performance gain on the KDD-1999.

Finally, we report the performance of zkPoA scheme



(a) KDD-1999

(b) UCR-ECG

(c) British Birdsong

Fig. 7: Detailed cost of ezDPS.

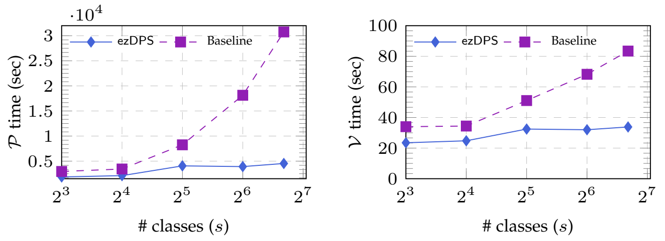


Fig. 8: Performance of zkPoA on British Birdsong.

proposed in §4.2.6. Since zkPoA is derived from the proof of inference for individual samples, our scheme maintains the same ratio of performance gain over the baseline as reported in §7.2. Concretely, we tested zkPoA on the British Birdsong dataset with 64 samples. As shown in Figure 8, we achieve $1.6\times$ to $6.8\times$ faster on the proving time and $1.4\times$ to $2.5\times$ better performance on the verification time and proof size. The complexity of zkPoA is linear with the number of samples, and its main overhead stems from the inference proof of individual samples.

7.3 Detailed Cost Analysis

We dissected the total cost of our scheme to investigate the impact of each data processing on the overall performance. Figure 7 presents the detailed cost of ezDPS with three datasets. In ezDPS, the sample was processed in four phases, including DWT noise reduction, Z-Score normalization, PCA feature extraction, and SVM classification.

•*DWT Processing:* The cost of DWT processing is stable when varying the number of classes (s) and contributes a considerable portion to the overall performance. This is because the complexity of DWT is independent of s, i.e., $O(mn)$, which is bigger than PCA (i.e., $O(m)$), but smaller than SVM (i.e., $O((n+k)t+ns)$) for a large number of classes. On the KDD-1999 dataset, the proving time is around 2.21 seconds, and the verification time and proof size are around 0.066 seconds and 70.5 KB, respectively. On the UCR-ECG dataset, the proving time, verification time, and proof size are around 59 seconds, 0.341 seconds, and 364 KB, respectively. On the British Birdsong dataset, ezDPS incurs 13.18 seconds for proving, 0.161 seconds for verification, and 172 KB of proof size. The difference in proving DWT among three datasets derives from the number of features m. UCR-ECG has the largest number of features, e.g., $m = 750$, hence the DWT denoising is the dominant part of it. On the other hand, $m = 30$ and $m = 169$ on KDD-1999 and British Birdsong, respectively, lead DWT to a small ratio of costs on these two datasets.

•*Z-Score-based Normalization:* The cost of Z-Score normaliza-

TABLE 3: Inference accuracy of ML algorithms on whole datasets.

Method	KDD1999				UCR-ECG					British Birdsong				
# classes	4	8	16	23	4	8	16	32	42	8	16	32	64	88
DWT+PCA+SVM	0.995	0.983	0.986	0.993	0.902	0.937	0.905	0.866	0.875	0.06	0.0	0.03	0.004	0.003
DWT+Z-Score+PCA+SVM	0.998	0.991	0.997	0.998	0.935	0.932	0.867	0.868	0.902	0.861	0.912	0.932	0.877	0.918
DWT+Z-Score+PCA+SVM (FPA)‡	0.98	0.99	0.98	0.98	0.92	0.912	0.85	0.85	0.89	0.85	0.90	0.92	0.87	0.89

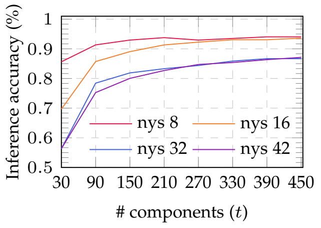
‡ FPA stands for fixed-point arithmetic.

tion is stable across a different number of classes. Similar to the DWT processing, the complexity of Z-Score is also linear to m, i.e., $O(m)$, and independent of the number of classes s. More specifically, the proving time, verification time, and proof size are around 0.11 seconds, 0.015 seconds, and 16 KB on the KDD-1999 dataset. For the UCR-ECG dataset, it costs 0.68 seconds of proving time, 0.038 seconds for verification, and 36.5 KB of proof size. As for the British Birdsong dataset, ezDPS requires 0.23 seconds, 0.02 seconds, and 22.4 KB of proving time, verification time, and proof size, respectively. The cost of Z-Score is nearly negligible on all three datasets, especially compared to DWT and SVM. The reason is that the constraints in proving Z-Score is relatively small.

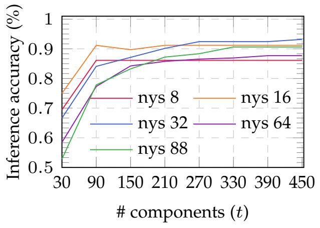
•*PCA-based Feature Extraction:* The cost of PCA processing is stable when s increases and it contributes the least portion to the overall performance of our scheme. This is because the complexity of PCA is $O(m)$ (which is also independent of s), compared with $O(nm)$ in DWT, $O(2m)$ in Z-Score, and $O((n+k)t+ns)$ in SVM. For example, it costs around 0.29 seconds to prove, 0.025 seconds for the verification, and around 26 KB for the proof size on the UCR-ECG dataset. Similar to Z-Score, the cost of proving PCA is also nearly negligible on all three datasets. This is because the number of constraints for PCA is the smallest among the four phases in the MLIP compared with DWT and SVM.

•*SVM Classification:* We use different strategies to reduce the size of the arithmetic circuit for SVM classification. For KDD-1999, we adopt the polynomial kernel as a substitute for the RBF kernel, which significantly reduces the number of support vectors. For example, $t = 11548$ on KDD-1999 with RBF kernel while $t = 1999$ if using the polynomial kernel. Moreover, although proving both kernels requires the Exp gadget, the degree in the polynomial kernel is usually a small integer. Thus, the number of constraints can be further reduced. Specifically, the proving time is from 2.5 seconds to 657 seconds, with respect to the number of classes s. The verification time and proof size range from 0.07 seconds and 75 KB to 1.138 seconds and 1215 KB. Note that the number of samples increases rapidly when s becomes larger, e.g., there are 1844 samples when $s = 4$, while 395216 samples when $s = 88$. Hence the amount of the support vectors grows, which makes SVM the most dominant part when $s \geq 16$.

For the datasets that are unsuitable to use other kernels, e.g., UCR-ECG and British Birdsong, we adopt the Nyströem approximation to replace the RBF kernel. The model parameter t is controlled under 450 thus SVM is not as resource-consuming as our prior work. More concretely, the proving time on UCR-ECG is from 12.79 to 38.52 seconds. The verification time is from 0.159 to 0.275 seconds, and the proof size is 169 to 294 KB. On the British Birdsong dataset, the proving time of SVM costs from 15 to 57 seconds, while its verification time and proof size are from 0.172 to 0.335 seconds and 184 KB to 357 KB, respectively, for 8 to 88 classes.



(a) UCR-ECG dataset



(b) British Birdsong dataset

Fig. 9: Accuracy comparison of ezDPS with Nyströem approximation and RBF kernel.

7.4 Accuracy

We report the accuracy of our MLIP on the whole datasets of KDD-1999, UCR-ECG, and British Birdsong. We use the given training and testing datasets in UCR-ECG. For KDD-1999 and British Birdsong, since there is no standard train/test data, we use the train-test-split method in sklearn with 20% of testing ratio. For different numbers of classes, we use all data from classes $0, 1, \ldots, X-1$ for $X \leq 100$ classes. The second line of Table 3 presents the plain accuracy of our MLIP on the selected datasets. Our optimized MLIP achieves high accuracy with more appropriate datasets, e.g., our method achieves over 99% accuracy on KDD-1999 for all classes and over 86% accuracy on UCR-ECG and British Birdsong. The first row of Table 3 illustrates the model accuracy without the Z-Score normalization stage. The differences between the model performance show that Z-Score effectively improves the model capability. For instance, on KDD-1999 and UCR-ECG, the model accuracy decreases from 0.01 to 0.33 when removing the normalization from the MLIP. On the other hand, Z-Score is proven to be vital in the British Birdsong dataset. When the normalization is removed, the models become entirely ineffective, e.g., the model accuracy is close to zero. The last row of Table 3 presents the accuracy of executing DWT+Z-Score+PCA+SVM inference with Fixed-Point Arithmetic (FPA), which is similar to how our ezDPS works. We can see that FPA leads to an accuracy decrease of around 1% to 2%. In KDD-1999, the pipeline with fixed-point arithmetic achieves around 98% accuracy, which decreases by around 1.8% than the floating-point representation. A similar trend is also observed in UCR-ECG and British Birdsong datasets, where the accuracy loses around 1% to 2% due to FPA.

Finally, we report the effectiveness of the Nyströem kernel approximation. The results are presented in Figure 9.

The solid lines in Figure 9 are the accuracy acquired by the pipeline with Nyströem approximation method, with different settings of parameter t. The accuracy of Nyströem approximation improves rapidly when $t \leq 90$, and grows slower after that. The MLIP with the approximation method achieves over 86% accuracy for all classes on both UCR-ECG and British Birdsong. In our implementation, we select the values of t for higher accuracy with the smallest number of components to reduce the size of the arithmetic circuit, which is a trade-off between accuracy and efficiency.

8 Mitigating Model Stealing Attacks

As discussed, model stealing attacks [7], [31], [65] aim to reconstruct the ML model from the inference result, given that the adversary has black box access to the model parameters. To our knowledge, there is no general defense against these attacks beyond limiting the number of queries the client can make to the model [31]. We present several strategies that can mitigate these attacks, and, with some efforts, they can be integrated orthogonally into our scheme to protect the model privacy for both the inference result and the proof.

Limiting prediction information. The model holder can limit the output information by releasing class probabilities only for high-probabilities classes (e.g., top-5 in ImageNet dataset [39]) [65], or only releasing the class labels [7], [65]. Limiting output information forces the adversary to query more, which permits the model holder to identify them by augmenting adversarial detection methods (see below) that analyze their behaviors against benign users. Tramer et al. [65] showed that by returning the class label without the confidence score (like ezDPS currently offers), the number of required queries to extract the model increases by 50-100 times. Thus, the model holder can increase the cost per query, thereby reducing the profit the adversary can make.

Adversarial detection. Juuti et al. [34] proposed a method to detect whether the adversary is attempting to steal the model by analyzing the distribution of the adversary's queries against the normal distribution. Kesarwani et al. [37] proposed two performance metrics (e.g., the information gain and the coverage of the input space) that quantify the rate of information the adversaries gained from the queries and are used to represent the status of the model extraction process. Another approach is to embed watermark techniques so that if the adversary steals the model, the owner can detect and certify the stolen model [1], [32].

Obfuscating prediction results. Several approaches suggest perturbing or adding noise to the prediction results to prevent the adversary from executing the (supervised) retraining process to reconstruct the model [7], [41], [65]. This can be achieved with Differential Privacy to hide the decision boundary between prediction labels regardless of how many queries are executed by the adversary [76]. Another approach is to poison the training objective of the adversary by actively perturbing the predictions without impacting the utility for benign users [51].

9 Related Work

Privacy-Preserving ML. Privacy-Preserving ML (PPML) permits secure evaluation of ML computation without leaking information about the ML model and training/testing data. Most PPML techniques rely on either secure computation protocols such as Multi-party computation (MPC) [10] and Homomorphic Encryption (HE) [17], or Trusted Execution Environment (TEE) such as Intel-SGX [9]. PPML has been investigated in both training and inference phases. Many PPML training schemes have been proposed for established ML algorithms such as decision tree [2], k-means clustering [5], [30], SVM [66], linear regression [48], [49], logistic regression (LR) [38], [48] and neural networks (NN) [48]. Other frameworks focus on the inference phase such as GAZELLE [35], SWIFT [38], MiniONN [42], XONN [54], CHET [11], Delphi [47], CryptoNets [19] and its variants [4], [27]. Given MPC and FHE incur high costs in large-scale data processing, some studies harnessed Intel-SGX to make PPML more practical [50]. Unlike our ezDPS or zkML, PPML protects the privacy of client and server data but not computation integrity.

Verifiable and zero-knowledge ML. Unlike PPML, verifiable ML (vML) and zkML focus on the integrity of delegated ML computation using VC and zero-knowledge techniques [6], [16], [20], [52], [59]. Both vML and zkML are still in the early development stage, with a limited number of schemes being proposed. In vML, the resource-limited client delegates the training/inference tasks to the server, and later checks if the task has been performed correctly (no privacy guarantee). Zhao et al. [75] proposed VeriML, a vML framework for linear regression, LR, NN, SVM, and DT training. Some vML schemes are designed for DNN inference (e.g., [18], [64]) using VC protocols (e.g., [20], [22]) or TEE [9]. On the other hand, zkML, first studied in 2020 [74], enables integrity and model privacy in the inference phase, where the client can verify if the inference result on her data is indeed computed from the server's committed model without learning the model parameters. Zhang et al. designed a zkDT scheme [74], followed by a few zero-knowledge DNN inference constructions [15], [40], [43]. Weng et al. proposed Mystique [69], a zkVC compiler for efficient zero-knowledge NN inference.

10 Conclusion

We proposed ezDPS, an efficient and zero-knowledge MLIP instantiated with classical ML algorithms including DWT, Z-Score, PCA, and SVM. We introduced new gadgets for proving ML operations in arithmetic circuits more effectively than generic approaches. We also propose several optimizations to reduce the model size. We fully implemented ezDPS and evaluated its performance on real-world datasets. T results showed that ezDPS is highly efficient, which achieves better performance than generic methods.

Acknowledgments

Thang Hoang is supported by an unrestricted gift from Robert Bosch, 4-VA, and the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org. Haodi Wang and Rongfang Bie are supported by the National Science and Technology Major Project (No. 2022ZD0115901), in part by the National Natural Science Foundation of China (No. 62177007, No. 62102035, No. 71961022, No. 62302485).

References

[1] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, Baltimore, MD, Aug. 2018. USENIX Association.

[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450, 2000.

[3] R. AlTawy, H. S. Galal, and A. M. Youssef. Mjolnir: Breaking the glass in a publicly verifiable yet private manner. *IEEE Transactions on Network and Service Management*, 2023.

[4] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821. PMLR, 2019.

[5] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 486–497, 2007.

[6] M. Campanelli, D. Fiore, and A. Querol. Legosnark: modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2075–2092, 2019.

[7] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan. Exploring connections between active learning and model extraction. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1309–1326, 2020.

[8] K. M. Chung, W. C. Kao, C. L. Sun, L. L. Wang, and C. J. Lin. Radius margin bounds for support vector machines with the rbf kernel. *Neural Computation*, 15(11), 2003.

[9] V. Costan and S. Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016, no. 086:1–118, 2016.

[10] R. Cramer, I. B. Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[11] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz. Chet: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 142–156, 2019.

[12] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML. The ucr time series classification archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/, 2018.

[13] H. de Valence. Merlin: composable proof transcripts for public-coin arguments of knowledge. https://docs.rs/merlin/, 2020.

[14] R. Debnath and H. Takahashi. Kernel selection for the support vector machine. *IEICE transactions on information and systems*, 87(12):2903–2904, 2004.

[15] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *Cryptology ePrint Archive*, 2021.

[16] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645. Springer, 2013.

[17] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, 2009.

[18] Z. Ghodsi, T. Gu, and S. Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4675–4684, Red Hook, NY, USA, 2017. Curran Associates Inc.

[19] 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*, pages 201–210. PMLR, 2016.

[20] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[22] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326. Springer, 2016.

[23] W. Haodi and T. Hoang. ezdps: An efficient and zero-knowledge machine learning inference pipeline. In *Proceedings on Privacy Enhancing Technologies*, pages 430–448, 2023.

[24] Y. He and J. Chen. Amlchain: Supporting anti-money laundering, privacy-preserving, auditable distributed ledger. In *International Symposium on Emerging Information Security and Applications*, pages 50–67. Springer, 2021.

[25] Y. He and J. Chen. Amlchain: Supporting anti-money laundering, privacy-preserving, auditable distributed ledger. In W. Meng and S. K. Katsikas, editors, *Emerging Information Security and Applications*, pages 50–67, Cham, 2022. Springer International Publishing.

[26] B. J. Heil, M. M. Hoffman, F. Markowetz, S.-I. Lee, C. S. Greene, and S. C. Hicks. Reproducibility standards for machine learning in the life sciences. *Nature Methods*, 18(10):1132–1135, 2021.

[27] E. Hesamifard, H. Takabi, and M. Ghasemi. Cryptodl: Deep neural networks over encrypted data. *CoRR*, abs/1711.05189, 2017.

[28] G. B. Huang, M. Mattar, T. Berg, and M. E. Learned. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[29] A. L. Isis and d. V. Henry. A pure-rust implementation of group operations on ristretto and curve25519. https://github.com/dalek-cryptography/curve25519-dalek, 2020.

[30] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *ACM KDD*, pages 593–599, 2005.

[31] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1345–1362, 2020.

[32] H. Jia, C. A. Choquette-Choo, V. Chandrasekaran, and N. Papernot. Entangled watermarks as a defense against model extraction. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1937–1954. USENIX Association, Aug. 2021.

[33] H. Jiang, Q. Tian, J. Farrell, and B. A. Wandell. Learning the image processing pipeline. *IEEE Transactions on Image Processing*, 26(10):5032–5042, 2017.

[34] M. Juuti, S. Szyller, S. Marchal, and N. Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.

[35] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.

[36] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, pages 177–194. Springer, 2010.

[37] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta. Model extraction warning in mlaas paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 371–380, 2018.

[38] N. Koti, M. Pancholi, A. Patra, and A. Suresh. SWIFT: Super-fast and robust Privacy-Preserving machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2651–2668. USENIX Association, Aug. 2021.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[40] S. Lee, H. Ko, J. Kim, and H. Oh. vcnn: Verifiable convolutional neural network based on zk-snarks. Technical report, Cryptology ePrint Archive, Report 2020/584. https://eprint.iacr.org/2020/584, 2020.

[41] T. Lee, B. Edwards, I. Molloy, and D. Su. Defending against neural network model stealing attacks using deceptive perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 43–49. IEEE, 2019.

[42] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 619–631, 2017.

[43] T. Liu, X. Xie, and Y. Zhang. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2968–2985, 2021.

[44] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1265–1282, 2019.

[45] E. J. d. S. Luz, W. R. Schwartz, G. Cámara-Chávez, and D. Menotti. Ecg-based heartbeat classification for arrhythmia detection: A

survey. *Computer methods and programs in biomedicine*, 127:144–164, 2016.

[46] R. J. Martis, U. R. Acharya, and L. C. Min. Ecg beat classification using pca, lda, ica and discrete wavelet transform. *Biomedical Signal Processing and Control*, 8(5):437–448, 2013.

[47] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2505–2522, 2020.

[48] P. Mohassel and P. Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 35–52, 2018.

[49] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38, 2017.

[50] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious {Multi-Party} machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, 2016.

[51] T. Orekondy, B. Schiele, and M. Fritz. Prediction poisoning: Towards defenses against dnn model stealing attacks. In *International Conference on Learning Representations*, 2020.

[52] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

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

[54] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar. {XONN}:{XNOR-based} oblivious deep neural network inference. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1501–1518, 2019.

[55] A. Salem, M. Backes, and Y. Zhang. Don't trigger me! a triggerless backdoor attack against deep neural networks. *arXiv preprint arXiv:2010.03282*, 2020.

[56] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang. Dynamic backdoor attacks against machine learning models. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 703–718. IEEE, 2022.

[57] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[58] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

[59] S. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

[60] S. Setty. Spartan: High-speed zksnarks without trusted setup. https://github.com/microsoft/Spartan, 2020.

[61] D. Stowell. British birdsong dataset. https://xeno-canto.org/about/xeno-canto, 2014.

[62] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009.

[63] M. S. Tong and W. C. Chew. *The Nyström Method*, pages 99–106. 2019.

[64] F. Tramer and D. Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

[65] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[66] J. Vaidya, H. Yu, and X. Jiang. Privacy-preserving svm classification. *Knowledge and Information Systems*, 14(2):161–178, 2008.

[67] C. Vonesch, T. Blu, and M. Unser. Generalized daubechies wavelet families. *IEEE Transactions on Signal Processing*, 55(9):4415–4429, 2007.

[68] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018.

[69] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang. Mystique: Efficient conversions for Zero-Knowledge proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518. USENIX Association, Aug. 2021.

[70] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13, 2000.

[71] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[72] Z. Xing, Z. Zhang, J. Liu, Z. Zhang, M. Li, L. Zhu, and G. Russello. Zero-knowledge proof meets machine learning in verifiability: A survey. *arXiv preprint arXiv:2310.14848*, 2023.

[73] X. Xu. Zero-knowledge proofs in education: a pathway to disability inclusion and equitable learning opportunities. *Smart Learning Environments*, 11(1):7, 2024.

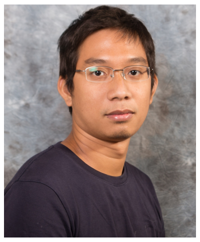
[74] J. Zhang, Z. Fang, Y. Zhang, and D. Song. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2039–2053, 2020.

[75] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems*, 32(10):2524–2540, 2021.

[76] H. Zheng, Q. Ye, H. Hu, C. Fang, and J. Shi. Protecting decision boundary of machine learning model with differentially private perturbation. *IEEE Transactions on Dependable and Secure Computing*, 2020.

Haodi Wang is currently a Ph.D. student at the School of Artificial Intelligence at Beijing Normal University. She received her bachelor's degree in 2018 at Beijing Normal University with Excellent Graduation Thesis Award. Her research interests are zero-knowledge proofs and privacy-preserving machine learning.

Rongfang Bie is currently a Professor at the School of Artificial Intelligence of Beijing Normal University where she received her M.S. degree on June 1993 and Ph.D degree on June 1996. She was with the Computer Laboratory at the University of Cambridge as a visiting faculty from March 2003 for one year. She is the author or co-author of more than 100 papers. Her current research interests include knowledge representation and acquisition for the Internet of Things, dynamic spectrum allocation, big data analysis and application.



Thang Hoang is an Assistant Professor in the Department of Computer Science at Virginia Tech (January 2021). He received his PhD degree in Computer Science from the University of South Florida, in August 2020. He received his MS degree in Computer Science from Chonnam National University, South Korea (2014), and BS degree in Computer Science from the University of Science, VNU HCMC, Vietnam (2010). Prior to joining Virginia Tech, he was a postdoctoral fellow at Carnegie Mellon University from August to December 2020. His research interests include applied cryptography, privacy-enhancing technologies, security, and biometrics.