ezDPS: An Efficient and Zero-Knowledge Machine Learning Inference Pipeline

Haodi Wang
Beijing Normal University / Virginia Tech
whd@mail.bnu.edu.cn

Thang Hoang
Virginia Tech
thanghoang@vt.edu

ABSTRACT

Machine Learning as a service (MLaaS) permits resource-limited clients to access powerful data analytics services ubiquitously. Despite its merits, MLaaS poses significant concerns regarding the integrity of delegated computation and the privacy of the server's model parameters. To address this issue, Zhang et al. (CCS'20) initiated the study of zero-knowledge Machine Learning (zkML). Few zkML schemes have been proposed afterward; however, they focus on sole ML classification algorithms that may not offer satisfactory accuracy or require large-scale training data and model parameters, which may not be desirable for some applications.

We propose ezDPS, a new efficient and zero-knowledge ML inference scheme. Unlike prior works, ezDPS is a zkML pipeline in which the data is processed in multiple stages for high accuracy. Each stage of ezDPS is harnessed with an established ML algorithm that is shown to be effective in various applications, including Discrete Wavelet Transformation, Principal Components Analysis, and Support Vector Machine. We design new gadgets to prove ML operations effectively. We fully implemented ezDPS and assessed its performance on real datasets. Experimental results showed that ezDPS achieves one-to-three orders of magnitude more efficient than the generic circuit-based approach in all metrics while maintaining more desirable accuracy than single ML classification approaches.

KEYWORDS

Verifiable Machine Learning, Zero-Knowledge Proofs, Principle Component Analysis (PCA), Support Vector Machine (SVM).

1 INTRODUCTION

Machine learning (ML) has grown to become a game-changer for the humane society. A well-trained ML model can effectively aid in performing highly complicated tasks such as medical diagnosis, natural language processing, intrusion detection, or financial forecasting. However, since a powerful ML model requires a large amount of data and computational resources for training, it may not be widely accessible to individuals or small organizations. To address this issue, Machine Learning as a Service (MLaaS) has been proposed, which permits resource-limited clients to access useful ML services (e.g., visualization, training, classification) offered by cloud providers.

Despite its usefulness, MLaaS has posed new integrity and privacy concerns. When the client delegates the ML computation to the MLaaS server, it is not clear if she will receive a reliable response. A corrupted server may process the client data arbitrarily or even substitute it with malicious data, making the outcome untrustworthy. This is especially critical for sensitive applications such as medical diagnosis, intrusion detection, or fraud detection. Computation integrity can be addressed with Verifiable Computation (VC), in which the MLaaS server attaches a *proof* to show that the computation is carried out correctly [19]. However, VC itself may not be sufficient for MLaaS because it only enables computation integrity but not the privacy of the parameters used in the computation. In MLaaS, the server uses its private ML model to process the client data. This sophisticated model may cost significant resources to obtain and, therefore, it is considered the intellectual property of the server. Moreover, such models may also be trained from sensitive training data (e.g., medical). As a result, it is undesirable that the MLaaS server leak any information about its private ML models when processing the client query.

The above privacy concern in MLaaS can be addressed by adding the *zero-knowledge* property to the VC proof, which permits verifiable computation without leaking any information other than the computation result [22]. Preliminary zero-knowledge VC (zkVC) protocols are computation and communication expensive with strong assumptions. Thanks to the recent advancements in cryptography, recent zkVC protocols have become more practical. Recently, Zhang et al. [71] have initiated zero-knowledge ML (zkML) research. In zkML inference, the server first commits to its ML model parameters and then provides an interface for the client to process her data sample. Given a client data sample, the server returns the ML computation result along with a zero-knowledge proof, which permits the client to verify the ML computation regarding the committed model without learning the model parameters in the proof.

Few zkML schemes have been proposed such as zero-knowledge Decision Tree (zkDT) [71], and zero-knowledge deep learning [40, 45]. Although the decision tree (DT) is simple with the lightweight model parameters, it offers limited accuracy for predicted outcomes. Deep neural networks (DNNs) permit a high accuracy rate, however, it may require a large amount of training data and heavy model parameters and, therefore, may not be ideal for some applications. Most zkML schemes (e.g., [40, 45, 71]) also focus solely on the final ML inference phase, while the data is generally processed via a so-called ML pipeline with multiple processing phases (e.g., (pre)processing, feature extraction, and classification) to achieve a desirable performance. Thus, there is a need to develop a zero-knowledge ML pipeline to achieve balanced performance and model complexity for some applications.

Proceedings on Privacy Enhancing Technologies 2023(2), 430–448

https://doi.org/10.56553/popets-2023-0061

Research Objective. The objective of this paper is to design an efficient and zero-knowledge ML pipeline, which permits the data to be processed in multiple phases for accuracy while at the same time, permitting the verifiability without leaking private model parameters at every processing phase.

Our Contributions. In this paper, we propose ezDPS, an efficient and zero-knowledge ML inference pipeline, which offers desirable security properties (e.g., zero-knowledge, verifiability) along with high accuracy for MLaaS. ezDPS comprises typical phases of an ML pipeline, including data (pre)processing, feature extraction, and classification. In ezDPS, we instantiate with established classical ML algorithms including, Discrete Wavelet Transformation (DWT) [66] for preprocessing, Principal Components Analysis (PCA) [69] for feature extraction, and Support Vector Machines (SVM) [7] for classification due to their popularity and wide adoption in many applications [47, 48]. To our knowledge, we are the first to propose a zero-knowledge ML inference pipeline. Our concrete contributions are as follows.

- **New gadgets for critical ML operations.** We create new gadgets for proving essential ML operations in arithmetic circuits such as exponentiation, absolute value, and max/min in an array (§4.1.2). These gadgets are necessary for proving concrete ML algorithms in our proposed scheme but also for other ML operations such as deep learning.
- **New zero-knowledge ML inference pipeline scheme with high accuracy.** Built on top of our proposed gadgets, we design ezDPS, an efficient and zero-knowledge ML inference pipeline, permits the data to be processed with effective ML algorithms for high accuracy (§4.2). We design new methods to prove DWT, PCA, and multi-class SVM with different kernel functions via an optimal set of arithmetic constraints. ezDPS significantly outperforms the generic approaches both in asymptotic and concrete performance metrics. ezDPS is designed to be compatible with any zkVC backend (similar to [71]), thus, its concrete efficiency can be further improved when adopted with a more efficient zkVC. We also propose a zero-knowledge proof-of-accuracy scheme to enable public validation of the effectiveness of the committed ML model on public datasets (§4.2.5).
- **Formal security analysis.** We present a formal security model for zero-knowledge ML inference pipeline (§3) and rigorously analyze the security of our scheme. We prove that ezDPS satisfies the security of a zero-knowledge ML inference pipeline (§5).
- **Full-fledged implementation, evaluation, and comparison.** We fully implemented our proposed techniques (§6) and conducted a comprehensive experiment to evaluate their performance in real-world environments. (§7). Experiments on real datasets showed that ezDPS achieves one-to-three orders of magnitude more efficient than the generic circuit approaches in all performance metrics (i.e., proving time, verification time, proof size). Our implementation is available at

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

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., [10, 20, 34, 43, 56]), which preserves the privacy of the client and server against each other but not computation integrity (see §8 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 [24], where some ML models are claimed to achieve high accuracy without having a proper way to validate them. 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 §4.2.5), and the verifier can verify that statement efficiently in zero knowledge.

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 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., [35]) 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 [61] (with Hyrax [67] as the underlying polynomial commitment scheme) as the backend zkPC-based CP-ZKP protocol due to its succinctness properties (e.g., linear proving time, sublinear verification time, and proof size), transparent setup, and support generic Rank-1 Constraint System (R1CS). 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 [61]). *Let $\mathbb{F}$ be a finite field and $C_{\mathbb{F}}$ 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 [31]. An effective ML pipeline contains three main phases, including data preprocessing, feature extraction, and ML training/inference as illustrated in Figure 1. In data preprocessing, raw samples $\mathbf{x} \in \mathbb{F}^m$ are collected, and then some preprocessing technique is used to reduce the impact of noise in the collection environment. Feature extraction extracts the most prominent dimension of the preprocessed data so that only a small set of features $\mathbf{x}' \in \mathbb{F}^k$ will be fetched for efficient computation and a high convergence rate. Finally, the ML training computes a prediction model $\mathbf{w}'$ from a set of feature vectors $\{\mathbf{x}'_i\}$ as well as their labels $\{y_i\}$, while ML inference computes the label y from the feature vector $\mathbf{x}'$ of a new observation using the prediction model $\mathbf{w}'$.

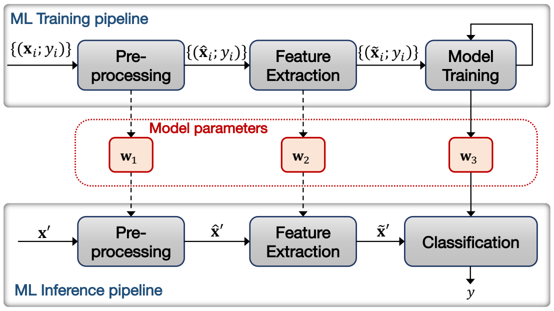


Figure 1: A general ML pipeline.

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., preprocessing, 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 $y \leftarrow \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x})$, where $\mathbf{x} \in \mathbb{F}^m$ 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 $\mathbf{x}$ 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.
- $\mathsf{cm} \leftarrow \mathsf{zkMLIP.Com}(\mathbf{w}, r, \mathsf{pp})$: Given MLIP parameters $\mathbf{w}$, it outputs a commitment cm under randomness r.
- $(y, \pi) \leftarrow \mathsf{zkMLIP}.\mathcal{P}(\mathbf{w}, \mathbf{x}, \mathsf{pp})$: Given MLIP model parameters $\mathbf{w}$ and a data sample $\mathbf{x}$, it outputs the inference result $y = \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x})$ and the proof π.
- $\{0, 1\} \leftarrow \mathsf{zkMLIP}.\mathcal{V}(\mathsf{cm}, \mathbf{x}, y, \pi, \mathsf{pp})$: Given a commitment cm, a sample $\mathbf{x}$, an inference result y, and a proof π, it outputs 1 if π is the valid proof for $y = \mathcal{F}_{\mathsf{mlip}}(\mathbf{w}, \mathbf{x})$ 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* $\mathbf{x} \in \mathbb{F}^m$, $\mathsf{pp} \leftarrow \mathsf{zkMLIP}.\mathcal{G}(1^\lambda, n)$, $\mathsf{cm} \leftarrow \mathsf{zkMLIP.Com}(\mathbf{w}, r, \mathsf{pp})$, $(y, \pi) \leftarrow \mathsf{zkMLIP}.\mathcal{P}(\mathbf{w}, \mathbf{x}, \mathsf{pp})$, *it holds that*

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

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

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

- ***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*

$$\Pr\left[\mathcal{A}(\mathsf{cm}, \mathbf{x}, y, \pi, \mathsf{pp}) = 1 \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{zkMLIP}.\mathcal{G}(1^\lambda, n) \\ \mathsf{cm} \leftarrow \mathsf{zkMLIP.Com}(\mathbf{w}, r, \mathsf{pp}) \\ \mathbf{x} \leftarrow \mathcal{A}(\mathsf{cm}, \mathsf{pp}) \\ (y, \pi) \leftarrow \mathsf{zkMLIP}.\mathcal{P}(\mathbf{w}, \mathbf{x}, \mathsf{pp}) \end{array}\right] \overset{c}{\approx}$$

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

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 [6, 64] 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., [6, 30, 36, 41, 64]) 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., return only the predicted label like our scheme currently offers), it makes the attack become 50-100× more difficult [64]. We elaborate all these approaches in Appendix E. 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. For curious readers, we also show how π may leak significant information about $\mathbf{w}$ if it is not zero-knowledge in Appendix F.

We also do not consider model poisoning/backdoor attacks (e.g., [57, 58]), 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., [46], 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 three processing phases, including data (pre)processing, feature extraction, and ML classification, as shown in Figure 1. We adopt ML algorithms for each phase including Discrete Wavelet Transformation (DWT) [66] for data preprocessing, Principal Components Analysis (PCA) [69] for feature extraction, and Support Vector Machine (SVM) [7] for classification. We focus on these algorithms because they were well-established in various systems and applications with high efficiency [47, 48]. ezDPS permits to verify a data sample was computed correctly with DWT, PCA, and SVM without leaking the parameters at each phase including, for example, low-pass 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 §4.2.5.

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. They can be used to prove other useful ML kernels (Appendix C) and deep learning components (Appendix D). We then present our techniques for proving DWT, 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.

4.1.1 Building Blocks.

4.1.1 Building Blocks. We first present building block gadgets that were previously proposed.

Permutation Gadget [71]. Given two vectors $\mathbf{v}, \mathbf{v}' \in \mathbb{F}^n$, the permutation $\mathsf{Perm}(\mathbf{v}, \mathbf{v}')$ 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 [60], the soundness error of the permutation test is $\frac{n}{|\mathbb{F}|} = \mathsf{negl}(\lambda)$.

Binarization Gadget [59]. 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. We now construct new gadgets that are needed in our ezDPS scheme. These gadgets can be used to prove other ML algorithms that incur the same operations.

Exponent Gadget. 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 $a \in \mathbb{F}$[1]. 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 $\mathbf{a}$. 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

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

Table 1: Notation table.

Variables	**Description**
DWT components	
$\mathbf{x} \in \mathbb{F}^m$	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
PCA components	
$\hat{\mathbf{x}} \in \mathbb{F}^m$	Sample input of size m to PCA
$\bar{\mathbf{x}} \in \mathbb{F}^m$	Mean vector
$\mathbf{V} = [\mathbf{v}_1^T, \ldots, \mathbf{v}_m^T]$	Eigenvectors
$(\lambda_1, \ldots, \lambda_m)$	Eigenvalues
k	Size of PCA output
SVM components	
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

4.2 Our Proposed Scheme

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 [66] 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 $\mathbf{x} \in \mathbb{F}^m$ 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

[1] The exponent gadget was briefly mentioned in [71], but no concrete constraints were given. We give concrete arithmetic constraints for proving exponent in arithmetic circuits.

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

for $i \in [1, t_\ell]$, where $\mathbf{h}, \mathbf{g} \in \mathbb{F}^c$ are low-pass and high-pass filters respectively, and $\mathbf{z}_0 = \mathbf{x}$. 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

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

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

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

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

$$\begin{aligned}\sum_{i=1}^{t_\ell} \alpha^i \mathbf{z}_\ell[i]^{(1)} &= \sum_{i=1}^{t_\ell} \alpha^i \cdot \sum_{k=1}^{c/2} \mathbf{h}[2k-1] \cdot \mathbf{z}_{\ell-1}[(2k+2i-3)_{\bmod t'_\ell}] \\ \sum_{i=1}^{t_\ell} \alpha^i \mathbf{z}_\ell[i]^{(2)} &= \sum_{i=1}^{t_\ell} \alpha^i \cdot \sum_{k=1}^{c/2} \mathbf{h}[2k] \cdot \mathbf{z}_{\ell-1}[(2k+2i-2)_{\bmod t'_\ell}] \\ \sum_{i=1}^{t_\ell} \alpha^i \mathbf{z}_\ell[i+t_\ell]^{(1)} &= \sum_{i=1}^{t_\ell} \alpha^i \cdot \sum_{k=1}^{c/2} \mathbf{g}[2k-1] \cdot \mathbf{z}_{\ell-1}[(2k+2i-3)_{\bmod t'_\ell}] \\ \sum_{i=1}^{t_\ell} \alpha^i \mathbf{z}_\ell[i+t_\ell]^{(2)} &= \sum_{i=1}^{t_\ell} \alpha^i \cdot \sum_{k=1}^{c/2} \mathbf{g}[2k] \cdot \mathbf{z}_{\ell-1}[(2k+2i-2)_{\bmod t'_\ell}]\end{aligned} \tag{5}$$

We convert (5) to the product of sums as

$$\begin{aligned}\sum_{i=1}^{t_\ell} \alpha^{\frac{c}{2}+i-2}\mathbf{z}_\ell[i] = &\sum_{k=1}^{c/2} \alpha^{\frac{c}{2}-k}\mathbf{h}[2k-1] \cdot \sum_{i=1}^{t_\ell} \alpha^{i-1}\mathbf{z}_{\ell-1}[2i-1] \\ &+ \sum_{k=1}^{c/2} \alpha^{\frac{c}{2}-k} \cdot \mathbf{h}[2k] \cdot \sum_{i=1}^{t_\ell} \alpha^{i-1}\mathbf{z}_{\ell-1}[2i] + (\alpha^{t_\ell}-1)\sum_{q=1}^{\frac{c}{2}-1} \alpha^{q-1} \\ &\cdot \sum_{p=1}^{q} (\mathbf{z}_{\ell-1}[2p]\mathbf{h}[c-2q+2p] + \mathbf{z}_{\ell-1}[2p-1]\mathbf{h}[c-2q+2p-1]) \\ \sum_{i=1}^{t_\ell} \alpha^{\frac{c}{2}+i-2}\mathbf{z}_\ell[i+t_\ell] = &\sum_{k=1}^{c/2} \alpha^{\frac{c}{2}-k}\mathbf{g}[2k-1] \cdot \sum_{i=1}^{t_\ell} \alpha^{i-1}\mathbf{z}_{\ell-1}[2i-1] \\ &+ \sum_{k=1}^{c/2} \alpha^{\frac{c}{2}-k}\mathbf{g}[2k] \cdot \sum_{i=1}^{t_\ell} \alpha^{i-1}\mathbf{z}_{\ell-1}[2i] + (\alpha^{t_\ell}-1)\sum_{q=1}^{\frac{c}{2}-1} \alpha^{q-1} \\ &\cdot \sum_{p=1}^{q} (\mathbf{z}_{\ell-1}[2p]\mathbf{g}[c-2q+2p] + \mathbf{z}_{\ell-1}[2p-1]\mathbf{g}[c-2q+2p-1])\end{aligned} \tag{6}$$

In (6), the number of constraints for proving DWT decomposition is reduced from mc to $c(\frac{c}{2}-1)+4$. To aid understanding, we present a toy example of our split technique in Appendix A. To prove the thresholding computation in (2), we employ the GT gadget, such that for $i \in [1, t_\ell]$:

$$\begin{cases}\mathsf{GT}(\mathbf{z}_\ell[i+t_\ell], \eta) \text{ for all } \mathbf{z}'_\ell[i+t_\ell] \neq 0 \\ \mathsf{GT}(\eta, \mathbf{z}_\ell[i+t_\ell]) \text{ for all } \mathbf{z}'_\ell[i+t_\ell] = 0 \\ \mathbf{z}'_\ell[i] - \mathbf{z}_\ell[i] = 0\end{cases} \tag{7}$$

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

$$
\begin{aligned}
&\sum_{k=1}^{t_\ell} \bar{\alpha}^{\frac{c}{2}+i-2}\hat{\mathbf{x}}_\ell[(2k+1)_{\mathrm{mod}\ t'_\ell}] = \\
&\sum_{k=1}^{c/2} \bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{h}}[2k-1] \cdot \sum_{i=1}^{t_\ell} \bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i] + \sum_{k=1}^{c/2} \bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{h}}[2k] \cdot \sum_{i=1}^{t_\ell} \bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i+t_\ell] \\
&+ (\bar{\alpha}^{t_\ell}-1) \cdot \sum_{q=1}^{\frac{c}{2}-1} \bar{\alpha}^{q-1} \cdot \sum_{p=1}^{q} \Big(\mathbf{z}'_{\ell-1}[p]\bar{\mathbf{h}}[c-2q+2p] + \mathbf{z}'_{\ell-1}[p+t_\ell]\bar{\mathbf{h}}[c-2q+2p-1]\Big) \\
&\sum_{k=1}^{t_\ell} \bar{\alpha}^{\frac{c}{2}+i-2}\hat{\mathbf{x}}_\ell[(2k)_{\mathrm{mod}\ t'_\ell}] = \\
&\sum_{k=1}^{c/2} \bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{g}}[2k-1] \cdot \sum_{i=1}^{t_\ell} \bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i] + \sum_{k=1}^{c/2} \bar{\alpha}^{\frac{c}{2}-k}\bar{\mathbf{g}}[2k] \cdot \sum_{i=1}^{t_\ell} \bar{\alpha}^{i-1}\mathbf{z}'_{\ell-1}[i+t_\ell] \\
&+ (\bar{\alpha}^{t_\ell}-1) \sum_{q=1}^{\frac{c}{2}-1} \bar{\alpha}^{q-1} \cdot \sum_{p=1}^{q} \Big(\mathbf{z}'_{\ell-1}[p]\bar{\mathbf{g}}[c-2q+2p] + \mathbf{z}'_{\ell-1}[p+t_\ell]\bar{\mathbf{g}}[c-2q+2p-1]\Big)
\end{aligned}
\tag{8}
$$

4.2.2 PCA-Based Feature Extraction. PCA [69] is a method to reduce the dimensionality of the data input by representing the most significant characteristics of $\hat{\mathbf{x}} \in \mathbb{F}^m$ 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 $\mathrm{S} = \frac{1}{N}\sum_{i=1}^N(\hat{\mathbf{x}}_i - \bar{\mathbf{x}})(\hat{\mathbf{x}}_i - \bar{\mathbf{x}})^\top$. 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 S such that $\mathrm{S} \times \mathbf{V} = \mathbf{V} \times \mathbf{\Lambda}$ where $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$. 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

$$
\tilde{\mathbf{x}} = (\hat{\mathbf{x}} - \bar{\mathbf{x}}) \times \mathbf{V}' \tag{9}
$$

Proving PCA Computation. There are $O(m \cdot k)$ constraints in (9), 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, (9) is equivalent to

$$
\begin{aligned}
\tilde{\mathbf{x}}[1] &= (\hat{\mathbf{x}} - \bar{\mathbf{x}}) \times \mathbf{V}'[1] \\
&\cdots \\
\tilde{\mathbf{x}}[k] &= (\hat{\mathbf{x}} - \bar{\mathbf{x}}) \times \mathbf{V}'[k]
\end{aligned}
\tag{10}
$$

where $\mathbf{V}'[k]$ 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 (10). Specifically, the prover can prove (10) holds by proving that

$$
\begin{aligned}
\sum_{i=1}^{k} \alpha^i \tilde{\mathbf{x}}[i] &= (\hat{\mathbf{x}} - \bar{\mathbf{x}}) \times \sum_{i=1}^{k} \alpha^i \mathbf{V}'[i] \\
&= \sum_{j=1}^{m} \left(\sum_{i=1}^{k} \alpha^i \mathbf{V}'[i] \right) \cdot (\hat{\mathbf{x}}[j] - \bar{\mathbf{x}}[j])
\end{aligned}
\tag{11}
$$

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.

4.2.3 SVM Classification. SVM [7] 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) [7] $\phi_{\mathrm{rbf}}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \cdot ||\mathbf{x}_i - \mathbf{x}_j||^2}$ 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

$$
y = \underset{\hat{c}}{\mathrm{argmax}} \sum_{i \in \mathcal{I}^{(\hat{c})}} \delta_i^{(\hat{c})} y_i^{(\hat{c})} \phi(\tilde{\mathbf{x}}, \mathbf{x}_i^{(\hat{c})}) + b^{(\hat{c})} \tag{12}
$$

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 (12), we harness Exp and Max gadgets in §4.1.2 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 [71] 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

$$
\begin{aligned}
p^{(\hat{c})} &= f^{(\hat{c})} + \beta \cdot \hat{c} \\
\bar{p}^{(\hat{c})} &= \bar{f}^{(\hat{c})} + \beta \cdot \sigma(\hat{c})
\end{aligned}
\tag{13}
$$

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

- $\underline{\mathsf{pp} \leftarrow \mathsf{ezDPS}.\mathcal{G}(1^\lambda)}$*: Output* $\mathsf{pp} \leftarrow \mathsf{zkp}.\mathcal{G}(1^\lambda)$
- $\underline{\hat{\mathsf{cm}} \leftarrow \mathsf{ezDPS.Com}(\mathbf{w}, r, \mathsf{pp})}$*: 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.*
- $\underline{(y, \pi) \leftarrow \mathsf{ezDPS}.\mathcal{P}(\mathbf{w}, \mathbf{x}, \mathsf{pp})}$*:*
  (1) *The server executes Algorithm 1 to compute* $y \leftarrow \mathsf{DPS}(\mathbf{w}, \mathbf{x})$*, and commits to all the auxiliary witnesses* aux *in (6), (7) (8), (11), (14) 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* $\pi \leftarrow \mathsf{zkp}.\mathcal{P}((\mathbf{w}, \mathsf{aux}), \mathbf{x}, y, \mathsf{pp})$*. The server sends (y, π) to the client.*
- $\underline{b \leftarrow \mathsf{ezDPS}.\mathcal{V}(\mathsf{cm}, \mathbf{x}, y, \pi, \mathsf{pp})}$*: Let* $\mathsf{cm} = (\hat{\mathsf{cm}}, \mathsf{cm}')$*, the client invokes* $b \leftarrow \mathsf{zkp}.\mathcal{V}(\mathsf{cm}, \mathbf{x}, y, \pi, \mathsf{pp})$ *and outputs b.*

Figure 2: Our ezDPS **Protocol.**

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 (13), respectively. The set of arithmetic constraints to prove (12) is

$$\begin{cases} k_i^{(\hat{c})} = -\gamma ||\tilde{\mathbf{x}} - \mathbf{x}_i^{(\hat{c})}||^2 \text{ for } i \in \mathcal{I}^{(\hat{c})}, \hat{c} \in [1, s] \\ f^{(\hat{c})} = \sum_{i \in \mathcal{I}^{(\hat{c})}} \delta_i^{(\hat{c})} y_i^{(\hat{c})} l_i^{(\hat{c})} + b^{(\hat{c})} \text{ for } \hat{c} \in [1, s] \\ \mathsf{Exp}(l_i^{(\hat{c})}, e, k_i^{(\hat{c})}) \text{ for } i \in \mathcal{I}^{(\hat{c})}, \hat{c} \in [1, s] \\ \mathsf{Max}(f^{(y)}, [f^{(1)}, \ldots, f^{(s)}]) \\ \mathsf{Perm}(\mathbf{p}, \bar{\mathbf{p}}) \\ f^{(y)} + \beta \cdot y = \bar{p}^{(1)} \end{cases} \tag{14}$$

Proving Other SVM Kernels. Our techniques can be used to prove other SVM kernels such as the polynomial kernel, Sigmoid kernel, etc. The polynomial kernel $\phi_{\mathsf{ply}}(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + a)^b$ can be easily proved via addition and multiplication gates, where γ, a, b are parameters. Although it is relatively easy to prove, the polynomial kernel usually achieves a lower accuracy than the RBF kernel [13]. Due to the space constraint, we show how to prove other kernels in Appendix C.

4.2.4 Putting Everything Together. We combine everything together and present the complete algorithmic description of ezDPS scheme in Protocol 1. We describe the functionality (Algorithm 1) that processes a data sample $\mathbf{x} \in \mathbb{F}^m$ with DWT (Figure 3, lines 1-14), PCA (line 15), and SVM (lines 16-19), and returns an inference result y.

4.2.5 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 $\mathcal{D} = (\mathbf{x}_1, \ldots, \mathbf{x}_M)$ with ground truth labels $\mathbf{T} = (t_1, \ldots, t_M)$. 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 $\mathbf{Y} = (y_1, \ldots, y_M)$ 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 $\psi = \frac{\sum_{i=1}^{M}(y_i \overset{?}{=} t_i)}{M}$ where $0 \le \psi \le 1$.

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 $\mathbf{Y} = \{(y_1, 1), \ldots, (y_M, M)\}$, $\mathbf{T} = \{(t_1, 1), \ldots, (t_M, M)\}$. 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 $\mathbf{Y}' = \{(y_1', \sigma_1(1)), \ldots, (y_M', \sigma_1(M))\}$ and $\mathbf{T}' = \{(t_1', \sigma_2(1)), \ldots, (t_M', \sigma_2(M))\}$, 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

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

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 (6) and reconstruction (8), 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 $16 \log_2 \frac{2m}{c} + (3n + 9)(m - \frac{c}{2})$ constraints for proving DWT. In PCA, the number of constraints is m (11). This is reduced from mk compared with direct proving (9) due to random linear combination. In SVM classification (14), our scheme incurs $(2n + k)t + 2s$ constraints for proving RBF kernel projection, and $(3n + 6)(s - 1) + 2s$ constraints for proving the classification for s classes and t constraints for the final decision function. The permutation trick in our proposed Max gadget permits us to reduce the number of comparisons from $O(s^2)$ in generic circuits to $O(s)$. In total, our scheme incurs $(2n + k)t + 4s + (3n + 6)(s - 1)$ constraints for proving s-class SVM classification with RBF kernel. Table 2 summarizes the complexity of our framework, compared with directly proving DWT, PCA, and SVM computations with generic circuits.

```
**Algorithm 1** ($y \leftarrow$ DPS(w, x)).
**Input**: Data sample $\mathbf{x} \in \mathbb{F}^m$, 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 $y$.
for $\ell = 1$ to $d$ do
    $t_\ell \leftarrow \frac{m}{2^\ell}$ and $t'_\ell \leftarrow \frac{m}{2^{\ell-1}}$
    for $i = 1$ to $t_\ell$ do
        $\mathbf{z}_\ell[i] \leftarrow \sum_{j=1}^{c} \mathbf{h}[j] \cdot \mathbf{z}_{\ell-1}[(2i+j-2)_{\bmod t'_\ell}]$
        $\mathbf{z}_\ell[i+t_\ell] \leftarrow \sum_{j=1}^{c} \mathbf{g}[j] \cdot \mathbf{z}_{\ell-1}[(2i+j-2)_{\bmod t'_\ell}]$
    for $i = 1$ to $t_\ell$ do
        $\mathbf{z}'_\ell[i] \leftarrow \mathbf{z}_\ell[i]$
        if $|\mathbf{z}_\ell[i+t_\ell]| - \eta > 0$ then
            $\mathbf{z}'_\ell[i+t_\ell] \leftarrow \mathsf{sign}(\mathbf{z}_\ell[i+t_\ell])(\mathbf{z}_\ell[i+t_\ell] - \eta)$
        else
            $\mathbf{z}'_\ell[i+t_\ell] \leftarrow 0$
    for $i = 1$ to $t_\ell$ do
        $\hat{\mathbf{x}}_\ell[2i-1] \leftarrow \sum_{j=1}^{c/2} (\bar{\mathbf{h}}[2j-1] \cdot \mathbf{z}'_\ell[(i+j-1)_{\bmod t_\ell}]$
        $+\bar{\mathbf{h}}[2j] \cdot \mathbf{z}'_\ell[t_\ell + (i+j-1)_{\bmod t_\ell}])$
        $\hat{\mathbf{x}}_\ell[2i] \leftarrow \sum_{j=1}^{c/2} (\bar{\mathbf{g}}[2j-1] \cdot \mathbf{z}'_\ell[(i+j-1)_{\bmod t_\ell}]$
        $+\bar{\mathbf{g}}[2j] \cdot \mathbf{z}'_\ell[t_\ell + (i+j-1)_{\bmod t_\ell}])$
$\tilde{\mathbf{x}} \leftarrow (\hat{\mathbf{x}}_d - \bar{\mathbf{x}})\mathbf{V}'$
for $\hat{c} = 1$ to $s$ do
    Let $\mathcal{I}^{(\hat{c})} = \{i : \delta_i^{(\hat{c})} > 0\}$
    $y_{\hat{c}} \leftarrow \sum_{i \in \mathcal{I}^{(\hat{c})}} \delta_i^{(\hat{c})} y_i^{(\hat{c})} \phi(\tilde{\mathbf{x}}, \mathbf{x}_i) + b^{(\hat{c})}$
$y_c \leftarrow \max(y_1, \ldots, y_s)$
return $c$
```

Figure 3: MLIP with DWT, PCA and SVM algorithms.

Table 2: Complexity of ezDPS **vs. generic circuit (baseline).**

		ezDPS	**Generic circuit**
DWT	Decomposition	$8\log_2 \frac{2m}{c}$	$8m - 4c$
	Thresholding	$(3n+9)(m - \frac{c}{2})$	$(5n+12)(m - \frac{c}{2})$
	Reconstruction	$8\log_2 \frac{2m}{c}$	$8m - 4c$
PCA		m	mk
Multi-class SVM (w/ RBF)		$(2n+k)t + 2s$ $+(3n+6)(s-1) + 2s$	$(2n+k+2)t + s$ $+(s^2 - s)(2n+5) + 2s - 2$

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. See Appendix B □

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 [66]. We used `sklearn` [55] to implement 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 for generating 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 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 §4.1.2, our gadget precomputes $a^{2^{i-1}}$, where $a = e^{-\gamma}$ and i is the index of the binary representation of the exponent. 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).

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` [62] and then invoked its library APIs to create proofs and verification. Concretely, we used Splartan$_{\mathsf{DL}}$ scheme, which implements *(i)* Hyrax polynomial commitment [67], *(ii)* `curve25519-dalek` [27] 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` [12] for non-interactive proof via Fiat-Shamir transformation.

Our implementation is available at https://github.com/vt-asaplab/ezDPS.

7 EXPERIMENTAL EVALUATION

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 did not make use of 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 ECG dataset in UCR Time Series Classification Archive (UCR-ECG)[11], Labeled Faces in the Wilds (LFW) [26], and Cifar-100 [38]. UCR-ECG contains 1800 records of ECG signals, each being of length 750. LFW contains 5749 human faces, where each image is of size 125 × 94 bits. Cifar-100 contains 100 classes, and the dimension of the samples is 3072. We used the subset of each dataset for the different number of classes.

Parameters. We used standard parameters as suggested in Spartan [61] (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 3). For LFW dataset, we scaled the dimension of the image inputs to 4200 when the number of classes is small (i.e., 8 and 16), and to 5655 for many classes (> 32). For DWT processing, we set the number of recursion levels to be 1 for noise reduction and $\eta = 0.2$ for processing the detail coefficients.

Table 3: Detailed model parameters.

UCR-ECG				Cifar-100				LFW			
m	k	s	t	m	k	s	t	m	k	s	t
750	33	4	54	3072	98	4	676	5655	119	8	1005
750	34	8	115	3072	108	8	1967	5655	121	16	1236
750	57	16	317	3072	121	16	2950	5655	123	32	1533
750	55	32	795	3072	120	32	3354	5655	125	64	1718
750	47	42	1061	3072	112	64	4627	5655	120	128	1384
				3072	108	100	6623	5655	106	256	4895
								5655	102	512	3862
								5655	121	1024	3233
								5655	118	2048	2645

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

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 3. Finally, we used the Grid Search method to find the best parameters for SVM and set $C = 1$, $\gamma = 0.001$.

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, PCA, and SVM) in our framework. Thus, we chose to compare with the naïve approach, in which we hardcore the whole DWT, 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). Note that for such cryptographic performance evaluation, we only used a reduced dataset of Cifar-100 and LFW that yield concrete model parameters after training as presented in Table 3. For UCG-ECG, we used the whole set as it is already small. We did not not evaluate on the whole set of Cifar-100 and LFW due to our limited hardware and the expensive cryptographic overhead incurred by the baseline. Instead, we report the accuracy of plain ML techniques and estimate the performance of our scheme when tested on the whole dataset (§7.4).

7.2 Overall Results

ezDPS is one to three orders of magnitudes more efficient than the baseline in *all* metrics. Figure 4 presents the performance of our technique compared with the baseline approach in terms of proving time, verification time, and proof size, in three datasets with different sizes. For example, on UCR-ECG dataset, our proving time is from 321 to 518 seconds for 4 to 42 classes, while it takes from 1429 to 2807 seconds if using the baseline approach. The gap between our scheme and the baseline is more significant when the number of classes increases. Specifically, on LFW dataset, with 8 classes, our scheme achieves 6.75× faster proving time, where it only takes 1702 seconds, compared with 11491 seconds in the baseline. With 2048 classes, our proving time is 6977 seconds, approximately 1842× faster than the baseline, which takes 2439811 seconds. The verification time and proof size follow a similar trend, in which ezDPS achieves an order of magnitude faster verification time and smaller proof size than the baseline. Specifically, on LFW dataset, the verification time is 6.6 seconds for 16 classes and 19.2 seconds in the baseline. The proof size is 3059 KB in our scheme, compared with 11946 KB in the baseline. On the LFW dataset with 2048 classes, our verification time is 9 seconds, and the proof size is 4411 KB, while it takes 123.6 seconds for verification with 56856 KB proof size in the baseline. This results in around 12× faster on the verification time and 14× smaller proof size, respectively.

We can also see the verification and bandwidth in ezDPS are highly efficient, i.e., less than 10 seconds 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 4 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 $O(s)$. Such asymptotic improvement helps to achieve one to three orders of magnitude faster computation time and lower communication overhead on real datasets.

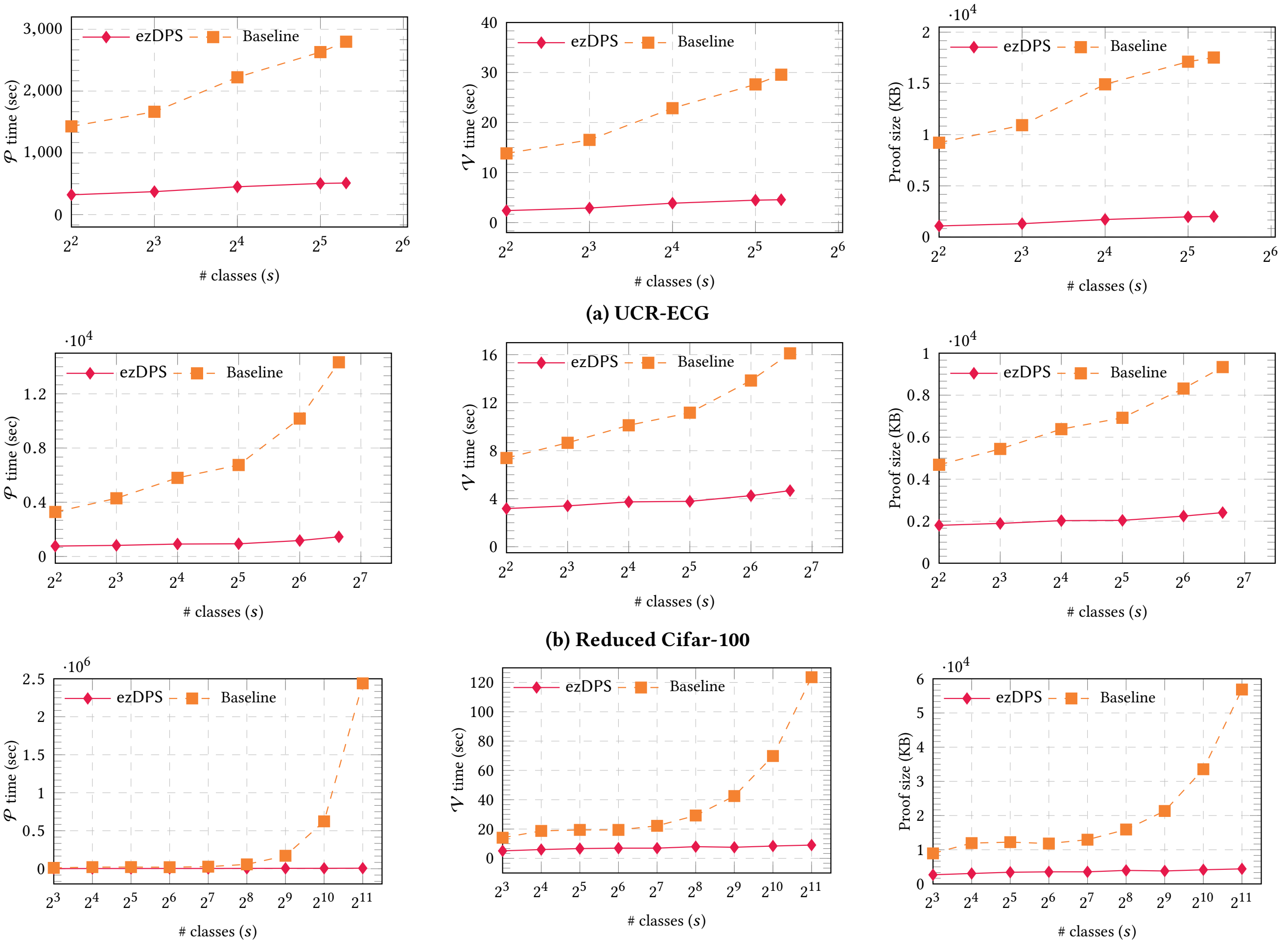
Finally, we report the performance of zkPoA scheme proposed in §4.2.5. Since zkPoA is derived from the proof of inference for individual samples, our scheme maintains the same ratio of performance gain over the baseline as reported in §7.2. Concretely, we tested zkPoA on the reduced LFW dataset with 64 samples. As shown in Figure 6, we achieve 6× to 9× faster on the prover's time, 3× faster on the verifier's time compared with the baseline. Regarding proof size, our scheme incurs 171392–226432 KB, which is three times smaller than the baseline that requires 576148–827968 KB. The complexity of zkPoA is linear with the number of samples, and its main overhead stems from the inference proof of individual samples.

We can see that our zkPoA scheme currently only supports plain accuracy verification, meaning the proof is given only for a specific test set. In the ML setting, cross-validation over different test sets is generally applied to report a more reliable accuracy result. It is interesting to explore if an zkPoA scheme can permit accuracy verification with cross-validation without leaking the model privacy due to multiple test sets. We leave it as an open research problem for future investigation.

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 5 presents the detailed cost of ezDPS with three datasets. In ezDPS, the sample was processed in three phases, including DWT noise reduction, 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



(a) UCR-ECG

(b) Reduced Cifar-100

(c) Reduced LFW

Figure 4: Performance of our scheme compared with the baseline.

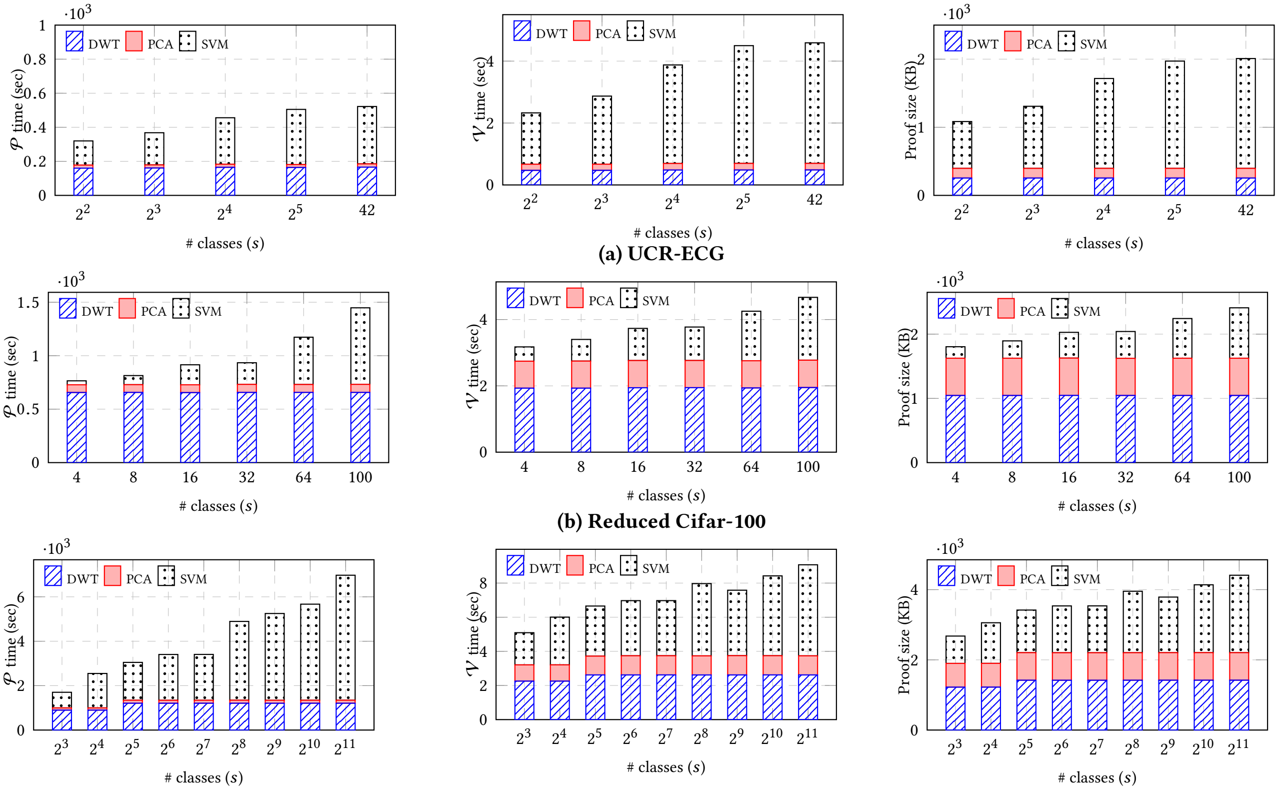
Table 4: Inference accuracy of ML algorithms on whole datasets.

Method	**UCR-ECG**				**Cifar-100**					**LFW**				
# classes	8	16	32	42	8	16	32	64	100	8	16	32	64	128
DT only	0.84±0.09	0.73±0.10	0.65±0.06	0.65±0.01	0.37	0.23	0.17	0.11	0.09	0.47±0.06	0.36±0.08	0.27±0.09	0.21±0.05	0.10±0.03
DWT+PCA+DT	0.79±0.07	0.77±0.07	0.65±0.07	0.66±0.04	0.32	0.22	0.17	0.11	0.09	0.43±0.07	0.32±0.05	0.24±0.05	0.17±0.07	0.15±0.04
SVM only	0.96±0.01	0.96±0.01	0.91±0.04	0.91±0.02	0.13	0.07	0.03	0.02	0.01	0.39±0.07	0.30±0.07	0.23±0.06	0.18±0.07	0.08±0.2
DWT+PCA+SVM	0.99±0.03	0.97±0.04	0.93±0.03	0.92±0.05	0.55	0.41	0.35	0.29	0.24	0.73±0.07	0.60±0.08	0.48±0.06	0.36±0.07	0.20±0.06
DWT+PCA+SVM (FPA)‡	0.97±0.03	0.95±0.04	0.91±0.02	0.91±0.06	0.55	0.4	0.35	0.28	0.24	0.73±0.07	0.6±0.06	0.47±0.06	0.36±0.06	0.19±0.05

‡ FPA stands for fixed-point arithmetic.

of classes. On the UCR-ECG dataset, the proving time is around 160 seconds, and the verification time and proof size are around 0.47 seconds and 256 KB, respectively. On Cifar-100, the proving time, verification time, and proof size are around 656 seconds, 1.94 seconds, and 1046 KB, respectively. On LFW dataset, the performance of the DWT phase ranges from 898 to 1209 seconds, 2.2 to 2.6 seconds, and 676 to 1421 KB, respectively. There is a considerable difference in proving DWT across three datasets. That is because the dimension of inputs varies between different datasets, e.g., m equals 750, 3072, and 4200 (or 5655) on UCR-ECG, Cifar-100, and LFW datasets, respectively.

•*PCA-Based Feature Extraction:* The cost of PCA processing is stable even when the number of classes 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 to s), compared with $O(nm)$ in DWT and $O((n+k)t + ns)$ in SVM. For example, it costs around 17 seconds for proving, 0.198 seconds for the verification, and around 141 KB for the proof size on the UCR-ECG dataset. The cost of proving PCA is nearly negligible



(a) UCR-ECG

(b) Reduced Cifar-100

(c) Reduced LFW

Figure 5: Detailed cost of ezDPS**.**

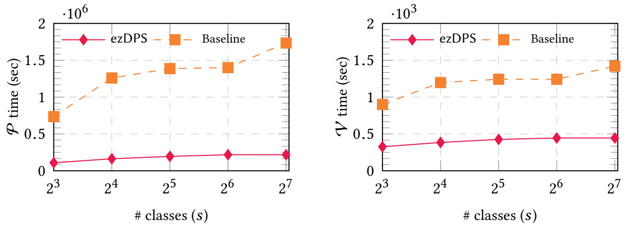


Figure 6: Performance of zkPoA on reduced LFW.

on UCR-ECG and LFW datasets. This is because the number of constraints for PCA is relatively small (i.e., 750 on UCR-ECG and 5655 on LFW) compared with DWT and SVM (e.g., on UCR-ECG, there are 75439 and over 110322 constraints in DWT and SVM, respectively). Since the verification time and proof size is sublinear, the proportion of PCA processing becomes larger relatively compared with DWT and SVM.

•*SVM Classifcation:* SVM computation is the most dominant factor, especially on large datasets (with more than 256 classes), which contributes over 73% to the total proving cost. That is because the cost of SVM is $O((n+k)t+ns)$, and thus it grows linearly with s. Notice that the increase of the number of classes also leads to the increase of the model parameters (t). On UCR-ECG dataset, the proving time of SVM ranges from 142 to 339 seconds. The verification time is from 1.65 to 4.24 seconds, and the proof size is 688 KB to 1623 KB for 4 to 42 classes. On Cifar-100 dataset, the proving time of SVM costs from 43 to 722 seconds, while its verification time and proof size are from 0.4 to 1.925 seconds and 179 KB to 785 KB, respectively, for 4 to 100 classes. On the LFW dataset, the proving time ranges from 704 to 6011 seconds for 8 to 2048 classes, while the verification time ranges from 1.89 to 5.73 seconds, and the proof size ranges from 779 to 2286 KB, respectively. The gap between SVM vs. DWT and PCA looks smaller in the verification time and proof size due to their sublinear growth of complexity by Spartan ZKP.

Estimated Performance on Whole Datasets. Based on the overall results (§7.2) and the above cost analysis on the reduced datasets, we estimated the cryptographic overhead of our scheme when tested on the whole Cifar-100 and LFW. For $X \in \{8, 16, 32, 64, 100\}$ classes in Cifar-100 with the standard train/test method, the proving time of our scheme is estimated to take 8189 to 108698 seconds. The verification time and proof size are estimated to take 8.8-26 seconds and 4154–11247 KB, respectively. In LFW dataset with X most sampled classes, the proving time is estimated to take 5823 to 24772 seconds, while the verification time and proof size is estimated to take 9.24–16.34 seconds and 4487–7424 KB, respectively. The estimated proving time is significant, since the estimation is based on our current hardware (i.e., a laptop without multi-threading). In practice, since the prover is the server that generally has better computational resource (e.g., multi-core CPU with higher frequency and multi-threading), we expect the actual proving time will be significantly faster. For the whole Cifar-100, since the number of support vectors (t) is large, it incurs a large model size, resulting in high proving time. We expect that once some optimization techniques (e.g., [32, 42]) are applied to reduce the model complexity, all the cryptographic overhead will be significantly reduced. We leave such optimization as our future work.

7.4 Accuracy

We report the accuracy of ML algorithms on the whole dataset of UCR-ECG, Cifar-100, and LFW. In Cifar-100, we used all data from classes $0, 1, \ldots, X-1$ for $X \leq 100$ classes and tested with its standard train/test method. For LFW and UCR-ECG, since there is no standard train/test split, we applied the cross-validation to report the accuracy. In LFW, since the number of samples in each class is unbalanced, we selected X classes that have the most data samples. In UCR-ECG, we chose data from classes $0, 1, \ldots, X-1$. Table 4 presents the plain accuracy of ML algorithms on the selected datasets. The last row of Table 4 presents the accuracy of executing DWT+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 LFW, DWT+PCA+SVM with floating-point arithmetic achieves 73% ± 7% and 60% ± 8% accuracy rates for 8 and 16 classes, respectively. The accuracy decreases 1% to 2%, leading to the accuracy rates of 72% ± 7% and 60% ± 6%, respectively. A similar trend is also observed in UCR-ECG and Cifar-100 datasets, where the accuracy loses around 1% to 2% due to FPA.

For curious readers, we conservatively report the best inference accuracy that each of our benchmark datasets currently achieves with different state-of-the-art ML pipeline techniques (without integrity and model privacy). UCR-ECG can achieve 97.5% accuracy by combining Gated Recurrent Unit with Fully Convolutional Network [14]. Cifar-100 can achieve 96.08% accuracy by combining ImageNet pre-trained model with sharpness-aware minimization [16]. Finally, LFW can achieve 99% accuracy using optimized VarGNet [70]. Since these pipeline techniques are highly optimized for each dataset, they yield higher accuracy than our generic framework. We leave the investigation on zero-knowledge proofs for optimization techniques that can be integrated into our framework to further improve the accuracy of our future work.

8 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) [9] and Homomorphic Encryption (HE) [18], or Trusted Execution Environment (TEE) such as Intel-SGX [8]. 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 [4, 28], SVM [65], linear regression [50, 51], logistic regression (LR) [37, 50] and neural networks (NN) [50]. Other frameworks focus on the inference phase such as GAZELLE [34], SWIFT [37], MiniONN [43], XONN [56], CHET [10], Delphi [49], CryptoNets [20] and its variants [3, 25]. Given MPC and FHE incur high costs in large-scale data processing, some studies harnessed Intel-SGX to make PPML more practical [52]. 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 [5, 17, 21, 54, 61]. 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. [72] proposed VeriML, a vML framework for linear regression, LR, NN, SVM, and DT training. Some vML schemes are designed for DNN inference (e.g., [19, 63]) using VC protocols (e.g., [21, 23]) or TEE [8]. On the other hand, zkML, first studied in 2020 [71], 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 [71], followed by a few zero-knowledge DNN inference constructions [15, 40, 45]. Weng et al. proposed Mystique [68], a zkVC compiler for efficient zero-knowledge NN inference.

9 CONCLUSION

We proposed ezDPS, an efficient and zero-knowledge MLIP instantiated with effective ML algorithms including DWT, PCA, and SVM. We introduced new gadgets for proving ML operations in arithmetic circuits more effectively than generic approaches. We fully implemented our ezDPS and evaluated its performance on real-world datasets. Experimental results showed that ezDPS is highly efficient, which achieves orders of magnitudes more efficient than generic approaches.

ACKNOWLEDGEMENT

We would like to thank our shepherd and the anonymous reviewers in PETS 2023 for their insightful comments and suggestions to improve the quality of this paper. Thang Hoang is supported by an unrestricted gift from Robert Bosch, 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 is sponsored by the National Natural Science Foundation of China under Grants 62177007, 62102035, 61571049, 71961022, the Fundamental Research Funds for the Central Universities under Grants 2020NTST32.

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1615–1631. https://www.usenix.org/conference/usenixsecurity18/presentation/adi

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

[3] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low latency privacy preserving inference. In *International Conference on Machine Learning*. PMLR, 812–821.

[4] Paul Bunn and Rafail Ostrovsky. 2007. Secure two-party k-means clustering. In *Proceedings of the 14th ACM conference on Computer and communications security*. 486–497.

[5] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2075–2092.

[6] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring connections between active learning and model extraction. In *29th USENIX Security Symposium (USENIX Security 20)*. 1309–1326.

[7] K. M. Chung, W. C. Kao, C. L. Sun, L. L. Wang, and C. J. Lin. 2003. Radius Margin Bounds for Support Vector Machines with the RBF Kernel. *Neural Computation* 15, 11 (2003).

[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[9] Ronald Cramer, Ivan Bjerre Damgård, et al. 2015. *Secure multiparty computation.* Cambridge University Press.

[10] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 142–156.

[11] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. 2018. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

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

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

[14] Nelly Elsayed, Anthony S Maida, and Magdy Bayoumi. 2018. Deep gated recurrent and convolutional network hybrid model for univariate time series classification. *arXiv preprint arXiv:1812.07683* (2018).

[15] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. ZEN: An Optimizing Compiler for Verifiable, Zero-Knowledge Neural Network Inferences. *Cryptology ePrint Archive* (2021).

[16] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2020. Sharpness-aware Minimization for Efficiently Improving Generalization. In *International Conference on Learning Representations*.

[17] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*. Springer, 626–645.

[18] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Ph. D. Dissertation.

[19] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 4675–4684.

[20] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.

[21] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. 2015. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)* 62, 4 (2015), 1–64.

[22] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.

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

[24] Benjamin J Heil, Michael M Hoffman, Florian Markowetz, Su-In Lee, Casey S Greene, and Stephanie C Hicks. 2021. Reproducibility standards for machine learning in the life sciences. *Nature Methods* 18, 10 (2021), 1132–1135.

[25] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep Neural Networks over Encrypted Data. *CoRR* abs/1711.05189 (2017). arXiv:1711.05189 http://arxiv.org/abs/1711.05189

[26] Gary B Huang, Marwan Mattar, Tamara Berg, and Miller Eric Learned. 2008. 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*.

[27] Agora Lovecruft Isis and de Valence Henry. 2020. A pure-Rust implementation of group operations on Ristretto and Curve25519. https://github.com/dalek-cryptography/curve25519-dalek.

[28] Geetha Jagannathan and Rebecca N Wright. 2005. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *ACM KDD*. 593–599.

[29] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*. 1345–1362.

[30] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. 2021. Entangled Watermarks as a Defense against Model Extraction. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1937–1954. https://www.usenix.org/conference/usenixsecurity21/presentation/jia

[31] Haomiao Jiang, Qiyuan Tian, Joyce Farrell, and Brian A Wandell. 2017. Learning the image processing pipeline. *IEEE Transactions on Image Processing* 26, 10 (2017), 5032–5042.

[32] Ajay J. Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. 2009. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2372–2379. https://doi.org/10.1109/CVPR.2009.5206627

[33] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 512–527.

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

[35] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*. Springer, 177–194.

[36] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. 2018. Model extraction warning in mlaas paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 371–380.

[37] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2651–2668. https://www.usenix.org/conference/usenixsecurity21/presentation/koti

[38] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.

[40] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2020. *vCNN: Verifiable Convolutional Neural Network based on zk-SNARKs.* Technical Report. Cryptology ePrint Archive, Report 2020/584. https://eprint. iacr. org/2020/584.

[41] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. 2019. Defending against neural network model stealing attacks using deceptive perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 43–49.

[42] Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour, Kai Yu, Liangliang Cao, and Thomas Huang. 2011. Large-scale image classification: Fast feature extraction and SVM training. In *CVPR 2011*. 1689–1696. https://doi.org/10.1109/CVPR.2011.5995477

[43] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 619–631.

[44] Tianyi Liu. 2020. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. https://github.com/TAMUCrypto/zkCNN.

[45] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.

[46] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1265–1282.

[47] Eduardo José da S Luz, William Robson Schwartz, Guillermo Cámara-Chávez, and David Menotti. 2016. ECG-based heartbeat classification for arrhythmia detection: A survey. *Computer methods and programs in biomedicine* 127 (2016), 144–164.

[48] Roshan Joy Martis, U Rajendra Acharya, and Lim Choo Min. 2013. ECG beat classification using PCA, LDA, ICA and discrete wavelet transform. *Biomedical Signal Processing and Control* 8, 5 (2013), 437–448.

[49] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2505–2522.

[50] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 35–52.

[51] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*. 19–38.

[52] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious {Multi-Party} Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*. 619–636.

[53] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2020. Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SyevYxHtDB

[54] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*. 238–252.

[55] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[56] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. {XONN}:{XNOR-based} Oblivious Deep Neural Network Inference. In *28th USENIX Security Symposium (USENIX Security 19)*. 1501–1518.

[57] Ahmed Salem, Michael Backes, and Yang Zhang. 2020. Don't Trigger Me! A Triggerless Backdoor Attack Against Deep Neural Networks. *arXiv preprint arXiv:2010.03282* (2020).

[58] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. 2022. Dynamic backdoor attacks against machine learning models. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 703–718.

[59] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.

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

[61] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.

[62] Srinath Setty. 2020. Spartan: High-speed zkSNARKs without trusted setup. https://github.com/microsoft/Spartan.

[63] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287* (2018).

[64] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*. 601–618.

[65] Jaideep Vaidya, Hwanjo Yu, and Xiaoqian Jiang. 2008. Privacy-preserving SVM classification. *Knowledge and Information Systems* 14, 2 (2008), 161–178.

[66] Cédric Vonesch, Thierry Blu, and Michael Unser. 2007. Generalized Daubechies wavelet families. *IEEE Transactions on Signal Processing* 55, 9 (2007), 4415–4429.

[67] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 926–943.

[68] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 501–518.

[69] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

[70] Mengjia Yan, Mengao Zhao, Zining Xu, Qian Zhang, Guoli Wang, and Zhizhong Su. 2019. Vargfacenet: An efficient variable group convolutional neural network for lightweight face recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 0–0.

[71] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2039–2053.

[72] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. 2021. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems* 32, 10 (2021), 2524–2540.

[73] Huadi Zheng, Qingqing Ye, Haibo Hu, Chengfang Fang, and Jie Shi. 2020. Protecting decision boundary of machine learning model with differentially private perturbation. *IEEE Transactions on Dependable and Secure Computing* (2020).

A EXAMPLE OF SPLIT TECHNIQUE AND APPLICATION

We present a concrete example to demonstrate how the split technique reduces the number of constraints in proving DWT. Suppose the input data is $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]$, the low-pass filter is $\mathbf{h} = [h_1, h_2, h_3, h_4]$. Directly computing the first half of the DWT frequency component $\mathbf{y} = [y_1, y_2, y_3]$, where

$$\begin{aligned} y_1 &= x_1 h_1 + x_2 h_2 + x_3 h_3 + x_4 h_4 \\ y_2 &= x_3 h_1 + x_4 h_2 + x_5 h_3 + x_6 h_4 \\ y_3 &= x_5 h_1 + x_6 h_2 + x_1 h_3 + x_2 h_4 \end{aligned} \tag{15}$$

requires 12 multiplications. The above computation can be combined by adopting the random linear combination, such that

$$\begin{aligned} &(\alpha^3 h_1 + \alpha^2 h_2 + \alpha h_3 + h_4)(x_1 + \alpha x_2 + \alpha^2 x_3 + \ldots + \alpha^5 x_6) \\ &= \alpha^3 y_1 + \alpha^5 y_2 + y_3 - D \end{aligned} \tag{16}$$

where D is the terms that have to be subtracted from the left side of (16) such that

$$\begin{aligned} D = &\, x_1 h_4 + \alpha(x_1 h_3 + x_2 h_4) \\ &+ \alpha^2(x_1 h_2 + x_2 h_3 + x_3 h_4) \\ &+ \alpha^4(x_2 h_1 + x_3 h_2 + x_4 h_3 + x_5 h_4) \\ &+ \alpha^6(x_4 h_1 + x_5 h_2 + x_6 h_3) \\ &+ \alpha^7(x_5 h_1 + x_6 h_2) + \alpha^8 x_6 h_1 - y_3 \end{aligned} \tag{17}$$

We can see that there are 20 multiplications in D as the step of the sliding window between two rounds is two (i.e., computing y_1 starts with x_1, while computing y_2 starts with x_3).

To improve the efficiency, the splitting technique separates the data sample and the low-pass filter into two parts, i.e., the odd part and the even part. Specifically, let $\mathbf{x}^{(1)} = [x_1, x_3, x_5]$, $\mathbf{x}^{(2)} = [x_2, x_4, x_6]$, $\mathbf{h}^{(1)} = [h_1, h_3]$, and $\mathbf{h}^{(2)} = [h_2, h_4]$. Therefore, (15) is equivalent to

$$\begin{aligned} &(x_1 + \alpha x_3 + \alpha^2 x_5)(\alpha h_1 + h_3) + (x_2 + \alpha x_4 + \alpha^2 x_6)(\alpha h_2 + h_4) \\ &= \Sigma_{i=1}^{3} \alpha^i y_i - D' \\ &= \Sigma_{i=1}^{3} \alpha^i y_i - (x_1 h_3(\alpha^3 - 1) + x_2 h_4(\alpha^3 - 1)) \end{aligned}$$

which only requires 4 multiplications to prove compared with 20 in (17). It reduces the number of intermediate terms in D', thereby reducing the number of constraints. We present the above toy example in Figure 7.

Application to zkCNN. We show that the split technique can be used to improve the efficiency of zkCNN [45] in some cases when the sliding step s between two rounds of convolution is larger than 1. Note that $s \geq 2$ is generally adopted in deep learning regions [39].

Suppose the input matrix $\mathbf{X}$ is of size $n \times n$ and the kernel matrix $\mathbf{W}$ is of size $w \times w$. The 2-D convolution between these two matrices is a matrix $\mathbf{U}$ of size $(\frac{n-w}{s} + 1) \times (\frac{n-w}{s} + 1)$ such that

$$\mathbf{U}[i][j] = \sum_{u=0, v=0}^{w-1, w-1} \mathbf{X}[si + u][sj + v] \cdot \mathbf{W}[u][v] \tag{18}$$

for $0 \leq i, j \leq (n/s - 1)$.

By zkCNN, the input and kernel matrices are first transformed to 1-D vectors to reduce the computation. Specifically, let $\bar{\mathbf{x}}, \bar{\mathbf{w}}, \bar{\mathbf{u}} \in \mathbb{F}^{n^2}$ be

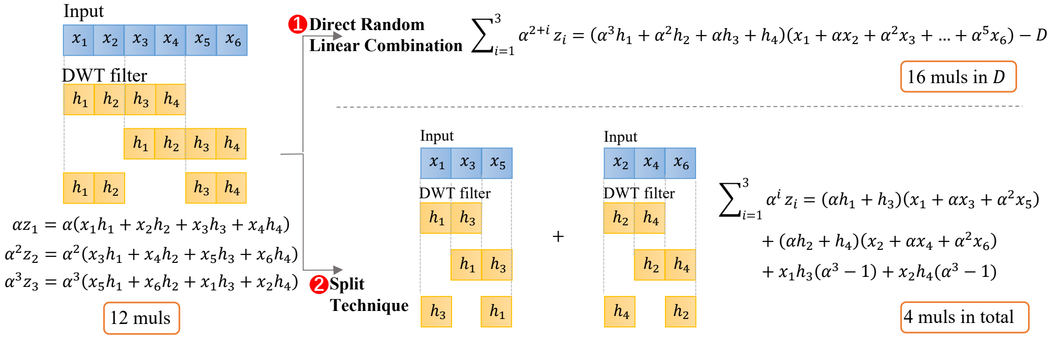

Input
x_1 x_2 x_3 x_4 x_5 x_6
DWT filter
h_1 h_2 h_3 h_4
$\alpha z_1 = \alpha(x_1h_1 + x_2h_2 + x_3h_3 + x_4h_4)$
$\alpha^2 z_2 = \alpha^2(x_3h_1 + x_4h_2 + x_5h_3 + x_6h_4)$
$\alpha^3 z_3 = \alpha^3(x_5h_1 + x_6h_2 + x_1h_3 + x_2h_4)$
12 muls
1 Direct Random Linear Combination
$\sum_{i=1}^{3} \alpha^{2+i}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

Figure 7: Example of split technique applied to DWT decomposition vs. directly using random linear combination.

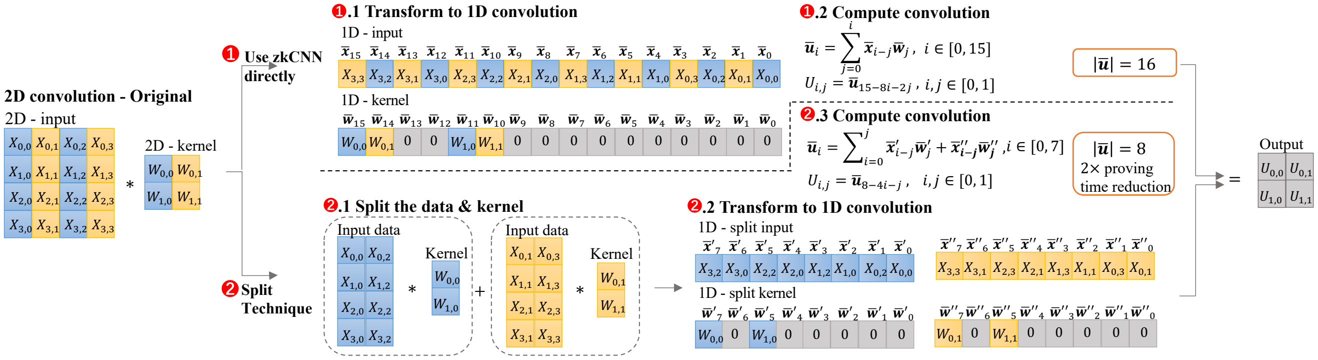
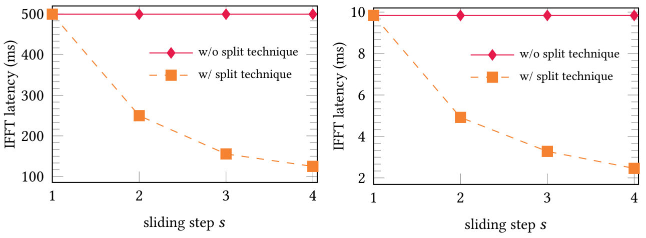

2D convolution - Original
2D - input
2D - kernel
1 Use zkCNN directly
1.1 Transform to 1D convolution
1D - input
1D - kernel
1.2 Compute convolution
$\bar{u}_i = \sum_{j=0}^{i} \bar{x}_{i-j}\bar{w}_j, \ i \in [0, 15]$
$U_{i,j} = \bar{u}_{15-8i-2j}, \ i, j \in [0, 1]$
$|\bar{u}| = 16$
2.3 Compute convolution
$\bar{u}_i = \sum_{i=0}^{j} \bar{x}'_{i-j}\bar{w}'_j + \bar{x}''_{i-j}\bar{w}''_j, i \in [0, 7]$
$U_{i,j} = \bar{u}_{8-4i-j}, \ i, j \in [0, 1]$
$|\bar{u}| = 8$
2× proving time reduction
Output
2 Split Technique
2.1 Split the data & kernel
Input data
Kernel
2.2 Transform to 1D convolution
1D - split input
1D - split kernel

Figure 8: Adopting split technique to convolutions in zkCNN when sliding step $s = 2$.



(a) VGG11 dataset **(b) LeNet dataset**

Figure 9: IFFT delay in zkCNN w/ or w/o split technique.

$$\bar{\mathbf{x}}[un + v] = \mathbf{X}[n - 1 - u][n - 1 - v], 0 \le u < n, 0 \le v < n$$

$$\bar{\mathbf{w}}[un + v] = \begin{cases} \mathbf{W}[u][v], \ 0 \le u, v < w \\ 0, \text{otherwise} \end{cases} \tag{19}$$

$$\bar{\mathbf{u}}[i] = \sum_{j=0}^{i} \bar{\mathbf{x}}[i - j]\bar{\mathbf{w}}[j]$$

(18) becomes

$$\mathbf{U}[i][j] = \bar{\mathbf{u}}[n^2 - 1 - sni - sj] \tag{20}$$

To compute 1-D convolution using the Fast Fourier Transform (FFT) and Inverse FFT (IFFT), $\bar{\mathbf{x}}, \bar{\mathbf{w}}$ are transformed to polynomials $\bar{\mathbf{x}}(\eta), \bar{\mathbf{w}}(\eta)$ with $\bar{\mathbf{x}}, \bar{\mathbf{w}}$ as coefficients, then $\bar{\mathbf{u}}(\eta) = \bar{\mathbf{x}}(\eta)\bar{\mathbf{w}}(\eta)$ by taking $\bar{\mathbf{u}}$ as the first n^2 coefficients. In zkCNN, the convolution $\bar{\mathbf{x}} * \bar{\mathbf{w}}$ can be proven by

$$\bar{\mathbf{u}} = \bar{\mathbf{x}} * \bar{\mathbf{w}} = \mathsf{IFFT}(\mathsf{FFT}(\bar{\mathbf{x}}) \odot \mathsf{FFT}(\bar{\mathbf{w}}))$$

where $\odot$ represents the Hadamard product. Since the size of $\bar{\mathbf{x}}$ and $\bar{\mathbf{w}}$ are n^2, the proving time is $O(n^2)$, the verifier's time and proof size are $O(\log^2 n)$ given oracle access to the multilinear extensions of the input and the output.

We observe that in (20), the majority of terms in $\bar{\mathbf{u}}$ are not the convolutional results when $s \ge 2$. By applying our split technique to (18), we show that the proving time for IFFT can be reduced by s times. Specifically, we split $\mathbf{X}, \mathbf{W}$ for s times respectively, such that

$$\mathbf{U}[i][j] = \sum_{k=1}^{s} \left[\sum_{u=0, v=0}^{w-1, w/s-1} \mathbf{X}[si + u][sj + sv + k] \cdot \mathbf{W}[u][2v + k] \right]$$

Instead of creating $\bar{\mathbf{x}}$ and $\bar{\mathbf{w}}$ of size n^2, we transform $\mathbf{X}[si+u][sj+sv+k]$, $\mathbf{W}[u, 2v+k]$ to $\bar{\mathbf{x}}^{(k)}, \bar{\mathbf{w}}^{(k)}$, respectively, following the same rule as in (19). Then

$$\bar{\mathbf{u}}^{(k)}[i] = \sum_{j=0}^{i} \bar{\mathbf{x}}^{(k)}[i-j] \cdot \bar{\mathbf{w}}^{(k)}[j] \tag{21}$$

The prover could use FFT to prove the correctness of (21) such that

$$\sum_{k=1}^{s} \bar{\mathbf{u}}^{(k)} = \mathsf{IFFT}\left(\sum_{k=1}^{s} (\mathsf{FFT}(\bar{\mathbf{x}}^{(k)}(\eta)) \odot \mathsf{FFT}(\bar{\mathbf{w}}^{(k)}(\eta)))\right)$$

By adopting the split technique to convolution layers in zkCNN, the proving time for the inverse FFT is reduced by s. To further demonstrate how it works, we provide an example in Figure 8 when $n = 4$, $w = 2$, and $s = 2$. As shown in Figure 8, directly transforming the inputs and kernels results in the vectors of size $n^2 = 16$ (case ❶). Adopting the split technique reduces the dimension to $\frac{1}{2}n^2 = 8$ (case ❷). Based on the zkCNN implementation [44], our experiments showed that adopting the split technique reduces the proving latency of IFFT in zkCNN from approximately 2 to 4 times in Lenet and VGG11 datasets (Figure 9).

B SECURITY PROOFS

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 3 on MLIP parameters $\mathbf{w}$. The correctness of our protocol in Figure 2 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 (14), 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 [60]. 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.

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.*

Figure 10: Simulator of Protocol 1.

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 10 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. □

C PROVING OTHER SVM KERNELS

Let $c \in \mathbb{F}$ be the output of the kernel function. We present constraints for other SVM kernels as follows.

- *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

$$\begin{cases} b = -\gamma' ||\mathbf{x}_i - \mathbf{x}_j|| \\ \mathsf{Exp}(c, e, b) \end{cases} \tag{22}$$

  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.

D PROVING DEEP LEARNING TECHNIQUES

In this paper, we mainly focus on designing techniques to prove classical ML algorithms in zero-knowledge. However, we show that they can be used to prove some deep learning techniques as follows.

Convolutional Layers. A convolutional layer computes the dot product between an input vector $\mathbf{x} \in \mathbb{F}^n$ and a small kernel $\mathbf{k} \in \mathbb{F}^c$. In the ith round, it computes the ith entry of the output such that $o[i] = \sum_{j=1}^{c} \mathbf{k}[j] \cdot \mathbf{x}[s(i-1)+j]$, where s is the step between two rounds. Our proposed technique can be applied to the convolutional layers w.r.t different settings of s.

- $s = 1$. This includes only addition and multiplication operations. Thus, the random linear combination can be applied to reduce the number of constraints, or other optimization techniques [40, 45] can be used.
- $s = 2$. Our split technique in §4.2.1 can be applied. Both the kernel and inputs are split into two parts, and a random linear combination can be performed.
- $s \geq 2$. Our split technique can be extended when the step is greater than two. We first split $\mathbf{x}$ and $\mathbf{k}$ to s parts, such that in the lth part,

$$o[i]^{(l)} = \Sigma_{i=1}^{c/s}\mathbf{k}[l+s(j-1)] \cdot \mathbf{x}[l+s(i+j-2)]$$

  and $o[i]$ can be computed as

$$o[i] = \Sigma_{l=1}^{s} o[i]^{(l)}$$

  Then the random linear combination can be utilized as described in §4.2.1.

Activation Layers. Let $c \in \mathbb{F}$ be the output of the activation function. We show how to prove activation functions with our gadgets as follows.

- *Sigmoid activation.* $f_{\mathsf{sig}}(x) = \frac{1}{1+e^{-x}}$ can be proven with following constraints

$$\begin{cases} \mathsf{Exp}(a, e, x) \\ a = (1+a) \cdot c \end{cases}$$

  where $a \in \mathbb{F}$ is the auxiliary witness.
- *ReLU activation.* $f_{\mathsf{relu}}(x) = \max(x, 0)$ can be proven with the gadget as $\mathsf{Max}(c, (x, 0))$.
- *Leaky ReLU activation.* $f_{\mathsf{lrelu}}(x) = \max(0.01x, x)$ can be proven as $\mathsf{Max}(c, (b, x))$ where $b = 0.01x$ is intermediate value.
- *Tanh activation.* $f_{\mathsf{tanh}}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ can be proven with following constraints

$$\begin{cases} \mathsf{Exp}(a_1, e, x) \\ a_1 \cdot a_2 = 1 \\ c \cdot (a_1 + a_2) = a_1 - a_2 \end{cases}$$

  where $a_1, a_2 \in \mathbb{F}$ are auxiliary witnesses.

Pooling Layers. The max pooling layer $c = \max(\mathbf{x})$ can be proven with $\mathsf{Max}(c, \mathbf{x})$ gadget.

E MITIGATING MODEL STEALING ATTACKS

As discussed, model stealing attacks [6, 29, 64] 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 [29]. 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]) [64], or only releasing the class labels [6, 64]. 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. [64] 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. [33] proposed an efficient method to detect whether the adversary is attempting to steal the model by analyzing the distribution of the adversary's queries against the normal (Gaussian) distribution. Kesarwani et al. [36] 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, 30].

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 [6, 41, 64]. 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 [73]. Another approach is to poison the training objective of the adversary by actively perturbing the predictions without impacting the utility for benign users [53].

F MODEL LEAKAGE IN PROOF OF INFERENCE WITHOUT ZERO KNOWLEDGE

We show how the proof of inference, without zero-knowledge, can leak model parameters. Let $\mathbf{w} \in \mathbb{F}^n$ be the MLIP model parameters, $\mathbf{x} \in \mathbb{F}^m$ be the public inputs and outputs, and $s = \lceil \log n \rceil$. According to Spartan, our backend ZKP protocol, the secret parameter $\mathbf{z} = (\mathbf{x}, 1, \mathbf{w})$ is encoded as a function $Z(\cdot) : \{0,1\}^s \to \mathbb{F}$ that the low degree extension of it is a multilinear polynomial $\tilde{Z}(\mathbf{y})$, such that

$$\tilde{Z}(\mathbf{y}) = \Sigma_{\mathbf{e} \in \{0,1\}^s} Z(\mathbf{e}) \cdot \prod_{i=1}^{s} (y_i \cdot e_i + (1-e_i)(1-y_i))$$

To prove the satisfiability of the arithmetic circuits, both parties invoke two sumcheck protocols, where a dot-product-proof protocol [67] is applied to guarantee the zero-knowledge property. Suppose we do not have the zero-knowledge property, the sumcheck protocol would leak the information of the secret parameter $\mathbf{z}$. Specifically, in the first round of the sumcheck protocol, upon receiving a random challenge $\mathbf{r}_x \in \mathbb{F}^s$, $\mathcal{P}$ computes $v_A = \Sigma_{\mathbf{y} \in \{0,1\}^s} \tilde{A}(\mathbf{r}_x, \mathbf{y}) \cdot \tilde{Z}(\mathbf{y})$, where $\tilde{A} : \mathbb{F}^s \times \mathbb{F}^s \to \mathbb{F}$ is a sparse multilinear polynomial, which is the low degree extension of matrix $\mathbf{A}$ in R1CS. Therefore, once acquiring v_A, $\mathcal{V}$ could compute the value of $\tilde{Z}(\mathbf{y})$, which contains private information of the model. This demonstrates the importance of having zero-knowledge in the integrity proof to protect the model parameter privacy.