SHIELD: A Framework for Efficient and Secure Machine Learning Classification in Constrained Environments

Jan Henrik Ziegeldorf, Jan Metzke, Klaus Wehrle
Communication and Distributed Systems (COMSYS), RWTH Aachen University, Germany
{ziegeldorf,metzke,wehrle}@comsys.rwth-aachen.de

ABSTRACT

Machine learning classification has enabled many innovative services, e.g., in medicine, biometrics, and finance. Current practices of sharing sensitive input data or classification models, however, causes privacy concerns among the users and business risk among the providers. In this work, we resolve the conflict between privacy and business interests using Secure Two-Party Computation. Concretely, we propose SHIELD, a framework for efficient, and accurate machine learning classification with security in the semi-honest model. Building on SHIELD, we realize several widely used classifiers and real-world use cases that compare favorably against related work. Departing definitively from prior works, all of SHIELD's protocols are designed from the ground up to enable secure outsourcing to untrusted computation clouds enabling even constrained devices to handle our most complex use cases in (milli)seconds.

ACM Reference Format:
Jan Henrik Ziegeldorf, Jan Metzke, Klaus Wehrle. 2018. SHIELD: A Framework for Efficient and Secure Machine Learning Classification in Constrained Environments. In *2018 Annual Computer Security Applications Conference (ACSAC '18), December 3–7, 2018, San Juan, PR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3274694.3274716

1 INTRODUCTION

With "data [...] the oil of the 21st century economy" [53], a growing number of services employ machine learning classification to refine it: Speech and handwriting recognition, biometric identification, or medical diagnosis are only a few examples. These applications are often deployed *as-a-service* in the cloud with access conveniently integrated into users' devices and applications. Users are required to send their data to the service providers who perform classifications using proprietary models, neglecting that users' inputs are often highly sensitive and must be protected. Speech recognition, e.g., not only reveals the user's searches to the service provider but allows creating voice profiles to impersonate users [59, 84].

A simple solution would be to perform classifications locally on the user's device (if it is has enough resources). Machine learning models are, however, expensive to train and their quality creates competitive edge. Service providers hence treat and protect them as their intellectual property. Data protection legislation presents a second reason why handing out models is not a viable solution. E.g., models for a genetic disease testing service may be trained over confidential patient records and residual risk remains that the learned models still leak information about individual patients [31] – sharing such models could be unlawful, e.g., according to the U.S. Health Insurance Portability and Accountability Act of 1996.

Hence, classification services–especially those involving highly sensitive data–face a conflict of business interests, regulatory issues, and privacy concerns. A promising solution is Secure Two-Party Computation (STC), which allows a user and service provider to compute classifications under strong security and privacy guarantees such that neither party is able to learn the other party's input [85]. First yet very specialized efforts in this direction have been made for Naive Bayes [75], Support Vector Machines (SVMs) [81], linear classifiers [39], face recognition [34, 70] and logistic regression [19]. Bost et al. [20] presented a first general framework for secure classification supporting hyperplane classifiers, Naive Bayes, and decision trees. Due to the success of deep learning, secure evaluation of Artificial Neural Networks (ANNs) has been the focus of related works [29, 52, 66, 68], while others [4, 36, 86] tackle secure pattern recognition with Hidden Markov Models (HMMs).

From these works, we identify three difficult challenges when designing STC protocols for classification. i) *Efficiency:* STC involves large numbers of cryptographic and interactive operations that may cause infeasible overheads in real-world applications. ii) *Accuracy:* many classification algorithms entail computations over very small probabilities that cause numerical instabilities [64]–the fact that all established STC approaches build on cryptographic primitives defined over *discrete* algebraic structures renders numerical accuracy even more challenging. iii) *Mobility:* traditional STC approaches assume high-powered hosts connected over stable, high-bandwidth, low-latency networks–mobile scenarios, however, typically involve resource-constrained devices and networks.

The main thrust of previous works has been to optimize efficiency while maintaining accuracy–none of them considers the outlined challenges posed by mobile usage scenarios such as resource constraints, network dynamics, and connectivity. In this paper, we introduce the SHIELD framework that allows two mutually distrustful parties to securely, efficiently, and accurately compute or outsource classifications using a range of state-of-the-art classifiers. SHIELD thereby enables *Secure Classification as a Service*, conceptually complementing existing Machine Learning as a Service cloud offers [6, 38, 54] that typically violate both the user's and the service provider's privacy. The following are our main contributions:

Framework for Secure Classification. We analyze core building blocks of classification algorithms and propose efficient hybrid STC protocols. Our building blocks provide tunable accuracy and are flexibly composable which we demonstrate by realizing a range

ACSAC '18, December 3–7, 2018, San Juan, PR, USA

ACM ISBN 978-1-4503-6569-7/18/12…$15.00
https://doi.org/10.1145/3274694.3274716

of established classifiers with distinctly different characteristics, i.e., linear classifiers such as SVMs, Bayesian classifiers with discrete and continuous features, ANNs with arbitrary activation functions, and the Viterbi algorithm on different HMM architectures.

Performance Evaluation. We evaluate SHIELD on different datasets and real-world use cases, e.g., bioinformatics sequence alignments and indoor localization. Our evaluation shows that SHIELD accurately handles large classification problems and, despite being designed for wide applicability, provides competitive performance even when compared to specialized approaches.

Secure Outsourcing. Departing significantly from related work, all of SHIELD's protocols are designed from the ground up to allow securely outsourcing protocol execution to untrusted computation clouds to cope with common resource limitations and challenges in mobile scenarios. Our evaluation shows that overheads for outsourcing range only in the order of milliseconds of processing and kilobytes of communication, rendering SHIELD applicable even for very constrained and challenged deployment scenarios.

2 PROBLEM STATEMENT

2.1 Scenario and Requirements

We consider two parties, a service provider S who holds a trained classification model M and a user U who holds a feature vector $\vec{x}$. Together, U and S want to compute $F(M, \vec{x})$, i.e., classify $\vec{x}$ using the model M. Due to privacy concerns, business interests, regulatory or legal requirements, neither party is willing to share her inputs with the other or any third party. In this paper, we thus show how U and S can compute $F(M, \vec{x})$ using STC without learning each other's inputs. This problem scenario is ubiquitous in different application areas of classification and pattern recognition, e.g., genetic disease testing [37], speech recognition [59, 72], biometric authentication [34, 58, 70], and localization [87]. Surveying these and further related works, we distill core requirements and design goals for secure classification and pattern recognition in the following.

Efficiency. Efficiency is generally of high importance in the surveyed applications. In user-centric applications, such as speech recognition [64] or localization [87], low latency, i.e., the time it takes to classify a single data record, is paramount. Data mining applications [29], in contrast, require high throughput to classify large batches of data records efficiently. While optimizing for latency automatically increases throughput the opposite is not necessarily true, e.g., when using Single-Instruction-Multiple-Data operations [29]. To ensure the wide applicability of secure classification, we thus priorize latency over throughput where necessary.

Accuracy. Secure classification protocols should ideally compute results identical to their insecure counterparts. In practice, certain degrees of inaccuracy are tolerable, e.g., in speech recognition where one is only interested in the best matching word but not the exact probabilities [58, 64]. Since the required numerical accuracy depends on the actual use case, secure classification protocols should allow trading accuracy against performance, ideally without the need to modify classification models or feature spaces.

Mobile Users and Constrained Environments. The increasing number of *mobile* users, e.g., using speech-to-text services, poses additional challenges to secure classification. Due to limited processing, communication, and energy resources, mobile users may not be able to execute STC protocols themselves. One of the most promising solutions to cater to such constrained deployment and operation scenarios is the outsourcing of costly computations from constrained user devices to more capable (cloud) peers. Without precluding other approaches, we hence pay special attention to the support for outsourcing in our analysis of related work as well as in the design of our SHIELD framework.

Security. We define the capabilities of the user, service provider, and potential cloud peers to attack the computation by the semi-honest model [51]. Shortly put, a *semi-honest* attacker correctly follows the protocol but may try to infer additional information from the transcript. Semi-honest behavior is not only the standard choice in the related literature [20, 28, 66]. It is also a widely used security model for outsourcing, arguing that cloud computation providers must preserve their reputation and thus have a strong interest in executing outsourced computations correctly [2, 44, 60]. Compared to security against stronger *malicious adversaries*, the semi-honest model allows for much more efficient protocols while still protecting against insiders and outsiders.

2.2 Analysis of Related Work

We analyze to which extent prior works address the stated requirements. We first present related work on secure classification and pattern recognition then briefly discuss orthogonal works. A quantitative evaluation and comparison against related work is provided separately for each classifier in Sections 5.1, 6.1, 7.1, and 8.1.

Secure Classification. Vaidya and Clifton [75] present a secure Naive Bayes classifier based on Homomorphic Encryption (HE) but evaluate neither performance nor accuracy. Yu et al. [81] present HE-based protocols for SVMs which is restricted to binary features and not evaluated. Graepel et al. [39] present an outsourcable secure Fisher's linear discriminant classifier based on Somewhat Homomorphic Encryption (SWHE). Different to our problem scenario, their approach requires that the user holds all inputs and most overheads are due to encryption on the user's side and cannot be outsourced. Bost et al. [20] present secure and efficient hyperplane classifiers, discrete Naive Bayes, and decision tree based on a combination of different HE schemes. Chandran et al. [24] improve upon Bost et al. using a combination of Garbled Circuits (GCs) and Additive Secret Sharings (ASSs). Our secure hyperplane and Naive Bayes classifiers are improvements on these works w.r.t. performance (up to two orders of magnitude faster), functionality (e.g., continuous feature spaces), and outsourcing. A particular focus among related works has been on secure classification using ANNs: Dowlin et al. [29] present a first Fully Homomorphic Encryption (FHE)-based approach that is outsourceable and efficient for batched classifications. Following, Chandran et al. [24], Liu et al. [52], and Riazi et al. [66] propose hybrid protocols (combining GCs, Goldreich-Micali-Wigderson (GMW), and ASS) that greatly reduce computation and communication overheads also for single classifications and conceptually lend themselves to outsourcing. Rouhani et al. [68] present a fully GC-based protocol and an outsourcing scheme based on Boolean secret sharing. Albeit being designed for wide applicability, SHIELD shows competitive performance on ANNs compared to these specialized approaches. Using an intricate combination of FHE and GCs, Juvekar et al.'s

approach [43] achieves another reduction of classification latency by one order of magnitude but cannot be outsourced. Finally, Mohassel and Zhang [55] provide GC and ASS-based protocols for the secure learning of ANNs.

Secure Pattern Recognition. Smaragdis et al. [72] followed by Pathak et al. [57, 59] first considered secure HMM computations in the context of speech recognition. Their approaches are based on HE which causes prohibitive overheads and numerical inaccuracies for all but very small models and requires plaintext knowledge for certain operations which prevents outsourcing. To tackle the challenge of numerical accuracy in secure computation over non-integers, Aliasgari et al. [5], Kamm et al. [45], and Demmler et al. [27] propose secure floating-point primitives. While these primitives could be used to implement secure HMM-based pattern recognition (still requiring additional measures to avoid underflows [64]), none of these works presents a concrete implementation and the performance comparison in [27] indicates high overheads. Franz et al. [36, 37] were first to build a secure HMM Forward algorithm with reasonable performance and accuracy on real-world HMMs based on HE and fixed-point precision arithmetic in logarithmic representation. However, this approach cannot be fully outsourced and scales poorly to long-term security levels due to the use of HE. Aliasgari et al. [4] compute the HMM Viterbi algorithm in the two-party setting using threshold-HE and their secure floating-point primitives [5] (discussed above). The evaluation of their two-party setup indicates prohibitive overheads in the order of hours even for very small HMMs with only five states. Finally, Ziegeldorf et al. [86] provide an efficient secure Forward algorithm based on GC and ASS. We extend on some of their techniques and propose a secure Viterbi algorithm that is faster than previous works by 9.6× to 48.3× and can be used in lieu of the less efficient secure Forward algorithm in the use cases presented in [37, 86].

Orthogonal Work. Different works consider secure training of classifiers on horizontally or vertically partitioned data, e.g., for Naive Bayes [76, 79] or ANNs [18]. The common assumption of these approaches is that learned models are not privacy sensitive and can be handed to the users who then classify locally on the plaintext model and data. In contrast, we assume that also the classification model requires protection, e.g., due to privacy concerns, business interests, or legal requirements. Finally, multiple other works on secure classification and pattern recognition are highly specialized to single use cases. Bos et al. [19] securely predict cardiovascular diseases based on logistic regression. The authors assume that the classifier is public knowledge and only the user's input must be hidden during classification. The proposed algorithms thus do not apply to our setting where nothing must be learned about the model and the input other than what is implied in the computed result. Barni et al. [11] securely evaluate linear branching programs and neural networks specialized to the classification of electrocardiograms using GCs and HE. The provided runtime estimates are two order of magnitudes higher than the state of the art and their use of HE prevents outsourcing. Finally, there have been multiple proposals specialized to secure face recognition using HE [34], GCs [70], or Oblivious Transfer (OT) and ASS [9]. In contrast, we aim to implement efficient general purpose classifiers that apply to a wide range of classification tasks.

3 CRYPTOGRAPHIC BUILDING BLOCKS

We provide a brief overview of the basic STC techniques that build the basis of related works and our own approach.

Oblivious Transfer. OT is a protocol between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$ which allows $\mathcal{R}$ to choose exactly one of many secrets held by $\mathcal{S}$ without $\mathcal{S}$ learning $\mathcal{R}$'s choice and $\mathcal{R}$ learning $\mathcal{S}$'s other secrets. In 1-2-OT, $\mathcal{S}$ holds two secret bits s_0 and s_1 while $\mathcal{R}$ holds a choice bit r; $\mathcal{R}$ obtains s_r and learns nothing about s_{1-r} while $\mathcal{S}$ learns nothing about the choice r. 1-2-OT can be generalized to 1-n-OT$_l$, where $\mathcal{S}$ holds n l-bit secrets and $\mathcal{R}$ learns only $s_r, r \in \{1, ..., n\}$. A batch of m parallel OTs is denoted by 1-n-OT$_l^m$, where $\mathcal{R}$ learns one secret s_{ir_i} from each run $1 \leq i \leq m$. 1-n-OT$_l^m$ can be efficiently instantiated with t bits symmetric security using *OT Extension* from only t *real* 1-2-OT$_t$, the so-called *base OTs* [8, 42].

Garbled Circuits. Yao's GCs [80] were the first generic STC protocol, allowing two parties $\mathcal{A}$ and $\mathcal{B}$ with private inputs x and y to evaluate $\mathcal{F}(x, y)$ without either party learning the other's input. Yao's protocol runs in three rounds: First, the function $\mathcal{F}$ is represented as a Boolean circuit $\mathcal{F}_{\text{Bool}}$, e.g., using special compilers [27]. Party $\mathcal{B}$ *garbles* this circuit by encrypting and permuting the truth table entries of each logic gate. Second, $\mathcal{B}$ sends the garbled circuit $\tilde{\mathcal{F}}_{\text{Bool}}$ together with its own garbled input $\tilde{y}$ to $\mathcal{A}$, while $\mathcal{A}$ obtains her own garbled input $\tilde{x}$ from $\mathcal{B}$ via OT. This ensures that $\mathcal{B}$ learns nothing about $\mathcal{A}$'s input x and vice versa. Finally, $\mathcal{A}$ obliviously *evaluates* $\tilde{\mathcal{F}}_{\text{Bool}}(\tilde{x}, \tilde{y})$ by decrypting the GC gate by gate. Yao's approach thus requires only a constant number of communication rounds such that its overheads are mainly determined by the circuit size. Different size-efficient circuit building blocks have been proposed in [41, 47]. Equally important are efficient garbling and evaluation functions [14, 73, 82].

Additive Secret Sharing. ASS [15, 28] uses an *arithmetic* circuit representation, i.e., $\mathcal{F}$ is represented using addition and multiplication gates over the ring $\mathbb{Z}_{2^l}$ (equality modulo 2^l denoted by $\equiv$). To evaluate such a circuit $\mathcal{F}_{\text{arith.}}$, $\mathcal{A}$ and $\mathcal{B}$ first share their input among each other, e.g., $\mathcal{A}$ with input x draws a random $r \in_U \mathbb{Z}_{2^l}$ and sends $\langle x \rangle_{\mathcal{B}} \equiv x - r$ to $\mathcal{B}$ keeping $\langle x \rangle_{\mathcal{A}} \equiv r$ as her own share. Since $\langle x \rangle_{\mathcal{A}} + \langle x \rangle_{\mathcal{B}} \equiv x$, we call $\langle x \rangle = (\langle x \rangle_{\mathcal{A}}, \langle x \rangle_{\mathcal{B}}) \leftarrow \textsc{Share}(x)$ an additive sharing of x. $\mathcal{A}$ and $\mathcal{B}$ then compute $\mathcal{F}_{\text{arith.}}(\langle x \rangle, \langle y \rangle)$ using only these shares. While addition can be evaluated locally due to commutativity of addition in $\mathbb{Z}_{2^l}$, multiplication gates require an interactive protocol between $\mathcal{A}$ and $\mathcal{B}$, which can be sped up using precomputed Multiplication Tripless (MTs) [12, 28]. Eventually, $\mathcal{A}$ and $\mathcal{B}$ obtain shares $\langle r \rangle_{\mathcal{A}}, \langle r \rangle_{\mathcal{B}}$ which they exchange and add to obtain the final result $r \equiv \langle r \rangle_{\mathcal{A}} + \langle r \rangle_{\mathcal{B}}$, denoted by $r \leftarrow \texttt{Recombine}(\langle r \rangle)$. Processing and communication overheads of ASS-based STC are dominated by the generation of the required MTs, i.e., by the number of multiplications in $\mathcal{F}_{\text{arith.}}$. The round complexity is determined by the multiplicative depth of the arithmetic circuit. Efficient building blocks have been proposed in [22, 23].

Hybrid STC. GCs are based on Boolean logic and thus suit logical operations. ASS, in contrast, is based on modular arithmetic and is more efficient for arithmetic operations. Following this observation, hybrid STC has first been proposed in *Tasty* [40] and since then been significantly improved by *ABY* [28] and *Chameleon* [66]. The common foundation of these frameworks are efficient conversion protocols between Boolean and arithmetic representations.

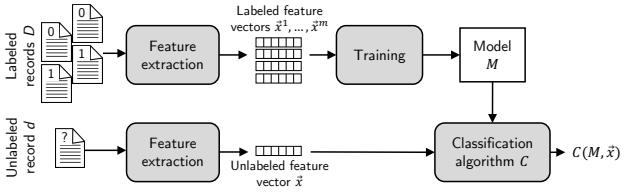

Figure 1: Overview of supervised classification.

4 SHIELD FRAMEWORK

Our approach to secure classification and pattern recognition is to abstract from specific applications and instead provide a framework of efficient and flexibly composable building blocks upon which a wide range of classifiers and use cases can be realized. To this end, we first briefly survey supervised classification and pattern recognition and distill core building blocks for which we propose protocols that are secure in the semi-honest model, efficient, accurate, and lend themselves to outsourcing. We then build and evaluate selected classifiers to showcase the applicability and flexibility of our SHIELD framework (Sec. 5 to 8). The entirety of our secure building blocks and secure classifiers complemented by our outsourcing protocols (Sec. 9) makes up our SHIELD framework.

4.1 Overview of Supervised Classification

Classification is the task of predicting a class label $c \in C = \{c_1, .., c_k\}$ for an *unlabeled* record d. In supervised machine learning (cf. Fig. 1), a statistical *model* M is trained on the *feature vectors* $\vec{x}^1, .., \vec{x}^m \in \mathbb{R}^n$ extracted from a labeled dataset D. Using the model M, the *classification algorithm* $C : \mathbb{R}^n \rightarrow C$ predicts a class $c = C(M, \vec{x}) \in C$ for d based on the feature vector $\vec{x}$ extracted from d. This traditional classification task can be generalized to sequence labeling where each element d_i of a sequence $(d_1, ..., d_T)$ should be assigned a class. Although we can reduce this problem to a set of T independent classifications, sequence labeling often involves (correlated) time-series data where classification accuracy can be increased by considering also nearby elements. Sequence labeling is a typical task in (temporal) pattern recognition with many real-world applications, e.g., part-of-speech tagging [59], localization [87], or sequence alignments in bioinformatics [37].

In this paper, we consider the problem of computing $C(M, \vec{x})$ securely to address scenarios where M and $\vec{x}$ are sensitive and held by two distrusting parties. There are, of course, different approaches to training models M, building features $\vec{x}$, and using them for classification in C. For this work, we select four classes of approaches: i) Hyperplane classifiers due to their ubiquity (e.g., in perceptrons, least squares, and Fisher's linear discriminant [17]), ii) ANNs due to their huge success in deep learning [49], iii) Naive Bayes as a popular baseline method [17, 20], and iv) HMMs as a representative and widespread approach to (temporal) pattern recognition [64]. Before we provide details and secure protocols for these classifiers (Sec. 5 to 8), we focus on their common building blocks.

4.2 Secure Building Blocks

We distill common building blocks of the selected classifiers then introduce secure, efficient, and accurate protocols for these based on the introduced cryptographic primitives (cf. Sec. 3). First, all classifiers require handling real-valued inputs and outputs, e.g., probabilities or weight vectors, and we thus provide secure building blocks for *computing over non-integers* (Sec. 4.2.1). A second ubiquitous building block is computing the *max* and *argmax* (Sec. 4.2.2), e.g., to select the most probable output class. *Scalar products* are a third basic building block that is heavily used in linear classifiers and ANNs, e.g., to compute convolutions (Sec. 4.2.3). A fourth important building block is the *evaluation of non-linear functions*, e.g., activation functions in ANNs or probability distributions in HMMs (Sec. 4.2.4). Finally, dynamic-programming algorithms, such as Viterbi, require *backtracking* to determine the optimal state sequence (Sec. 4.2.5). Tab. 7 in Appendix A summarizes all building blocks and a security discussion is given in Appendix B.1.

4.2.1 Representation of Real Numbers. Cryptographic primitives typically operate over discrete algebraic structures (cf. Sec. 3), raising the question how to handle non-integers. One approach is secure floating-point arithmetic [3, 27, 45], another is multiplying all non-integers v_i by a large constant K such that $Kv_i \in \mathbb{Z}$ [20]. Both approaches incur high overheads (e.g., multiplying by K blows values up to hundreds of bits in length) and often provide more accuracy than necessary.

In SHIELD, we represent non-integers with fixed-point precision as in [23, 86]. Formally, we transform $x \in \mathbb{R}$ to $x' \in \mathbb{N}$ by $\text{F2I}(x, l, s) = \lfloor 2^s x \rceil \bmod 2^l$ (float-to-integer). This encoding preserves signed integer arithmetic when decoded as $\text{I2F}(x', l, s) = (x' - 2^l)/2^s$ for $x' > 2^{l-1}$ and $\text{I2F}(x', l, s) = x'/2^s$ otherwise (integer-to-float). After transforming all inputs (i.e., models and features) using F2I, all intermediate values and results are kept in this representation. Note that the sum of two scaled values has the same scaling and the bitlength increases by at most one, while multiplication accumulates the scaling factor 2^s and bitlengths add up which may quickly overflow the available bitlength l. To prevent this, we use the secure Rescale protocol from [86] to scale down by factor 2^s before any subsequent addition or multiplication.

Since fixed-point precision introduces quantization errors we need to carefully evaluate whether our secure classifiers remain accurate. Indeed, we find that this approach is not sufficiently accurate for HMM computations which involve extremely small probabilities. In this context, Aliasgari et al. [3, 4] argue that full floating-point precision is required but report runtimes in the order of hours even for small HMMs. An alternative is to compute in *logspace* as proposed in [30]. Formally, we transform $p \in (0, 1] \subset \mathbb{R}$ to an integer in logspace $\hat{p}' \in \mathbb{N}$ by $\hat{p}' = \text{F2I}(\log(p), l, s)$ with the inverse $p = \exp(\text{I2F}(\hat{p}', l, s))$, denoted F2LI and LI2F. We represent $\log(0) =$ `Logzero` by a sufficiently small integer.

4.2.2 Secure Max and Argmax. Given an additively shared vector $\langle \vec{x} \rangle = (\langle x_1 \rangle, ..., \langle x_n \rangle)$, Argmax (Prot. 5, Appendix A) securely computes the maximum value (max) and its index (arg max). $\langle \vec{x} \rangle$ is first fed into n parallel garbled addition circuits to convert to garbled values $\tilde{x}_i$. On the garbled values, U and S efficiently select the max and arg max using pairwise comparisons as proposed in [47]. The garbled results $\tilde{x}^*, \tilde{i}^*$ are converted back to additive shares $\langle x^* \rangle$, $\langle i^* \rangle$ using the OT-based subtraction circuit proposed in [28]. Note that computing the arg max makes up one third of the overheads of Argmax and we can leave this part out if we only need the max.

4.2.3 Secure Scalar Products. ScalarProd (Prot. 6, Appendix A) securely computes scalar products on additive shares in a straightforward manner using only ASS-based addition and multiplication (Sec. 3) and rescaling (Sec. 4.2.1): U and S engage in n multiplications and add resulting shares locally then invoke Rescale on the result to restore the correct scaling by factor 2^s as required in our fixed-point number representation. To improve efficiency, we batch all messages required for the n parallel multiplications, resulting in a total of only two rounds of communication.

4.2.4 Secure Approximation of Non-linear Functions. With the established STC techniques (Sec. 3), we can already efficiently compute many functions used in machine learning, e.g., the identity, binary step, rectified linear, or maxout activation functions for neurons in ANNs. However, a wide range of other important, especially non-linear, functions cannot be efficiently computed, e.g., Sigmoid, Gaussian, and SoftPlus activation functions, as well as Gaussian, Gamma, and LaPlace probability distributions. Related works often try to circumvent usage of these functions, e.g., Dowlin et al. [29] construct an ANN with the Sigmoid function on the output layer then note it is only important for training and is left out during classification. We argue that it is generally desirable to be able to evaluate such non-linear functions and probability distributions securely as they are important building blocks for classification and other machine learning algorithms [29, 69].

Generally, what we aim for is a building block that securely computes a possibly secret function f at a possibly secret point x and returns the result in secret-shared form. In the following, we present three such building blocks with different characteristics: i) a generic yet efficient approach for the evaluation of arbitrary secret functions at secret points, ii) a more efficient approach for the evaluation of arbitrary secret functions at evaluation points known by one party, iii) a highly efficient protocol for the evaluation of Gaussians with secret parameters at secret evaluation points.

Case 1: Secret arbitrary f and secret x. We approximate an arbitrary $f : \mathbb{R} \rightarrow \mathbb{R}$ at point $x \in \mathbb{R}$ by k polynomials $g_i(x) \in \mathbb{R}[X]$ of degree d for $x \in [r_i, r_{i+1})$ with $(-\infty, r_1) \cup ... \cup [l_k, \infty) = \mathbb{R}$ where the choice of intervals and polynomials minimizes an adequate error measure. PolyFuncApprox (Prot. 7, Appendix A) is a secure protocol for this task taking as input the shared evaluation point $\langle x \rangle$ and the shared approximation parameters $\langle \mathcal{P}_f \rangle = (\langle a_{11} \rangle, ..., \langle a_{kd} \rangle, \langle r_1 \rangle, ..., \langle r_k \rangle)$. It transforms the inputs to garbled values using an addition circuit, uses the circuit from [62] to select g_i such that $r_i \leq x < r_{i+1}$, and converts the coefficients $a_{id}, ..., a_{i0}$ of g_i to additive shares. To efficiently compute $\langle g_i(x) \rangle$ on $\langle x \rangle$ and $\langle g_i \rangle = (\langle a_{id} \rangle, ..., \langle a_{i0} \rangle)$ (denoted by $\langle g_i(x) \rangle \leftarrow$ EvalPoly$(\langle g_i \rangle, \langle x \rangle)$), we propose a tree-based scheme that requires $\lceil \log_2(d) + 1 \rceil$ rounds of multiplications by evaluating $g_i(x)$ up to $a_{2^i} x^{2^i}$ in round i. Applying rescaling after each round of multiplications results in a total of $2(\lceil \log_2(d) \rceil + 1)$ communication rounds. Finally, shares of the approximated result $\langle f'(x) \rangle$ = EvalPoly$(\langle g_i \rangle, \langle x \rangle)$ are returned to U and S.

Case 2: Secret discrete f and known x. We treat the scenario where the evaluation point x is known to one party and the target function f is discrete, e.g., in Naive Bayes with discrete features or HMMs with discrete emissions (note that a continuous function f could be easily discretized by subsampling). In related work [37, 61], this problem is solved using HE, i.e., U encrypts her choice x_i in m selectors of which only the i^{th} encrypts a one and all others encrypt zero. S can then obtain an encryption of $f(x)$ by multiplying the selectors pairwise against $f(x_1), ..., f(x_m)$ and summing the results using the HE scheme, i.e., $[\![f(x)]\!] = [\![0]\!] \odot [\![f(x_1)]\!] \oplus ... \oplus [\![1]\!] \odot [\![f(x_i)]\!] \oplus ... \oplus [\![0]\!] \odot [\![f(x_m)]\!]$ ($[\![\cdot]\!]$ denotes encryption, $\oplus$ addition and $\odot$ multiplication on ciphertexts). In comparison, our protocol OtFuncApprox (Prot. 8, Appendix A) is much more efficient since we substitute the expensive HE operations by OT which can be realized using highly efficient symmetric cryptography primitives and one-time-pad operations (cf. Sec. 3). At the start of OtFuncApprox, U and S hold shares of the m function values $\langle f(X) \rangle = (\langle x_1 \rangle, ..., \langle x_m \rangle)$ and U holds the evaluation point $x \in \{x_1, ..., x_n\}$ in clear. In the first step, S blinds each of its share with the *same* random value $r_S \in_R \mathbb{Z}_{2^l}$, i.e., computes $\langle f(x_i) \rangle_S + r_S$. Both parties then engage in 1-m-OT$_l$ on the m blinded shares $(\langle f(x_1) + r_S \rangle_S, ..., \langle f(x_1) + r_S \rangle_S)$ from which U learns $\langle f(x) + r_S \rangle_S$ and nothing else while S learns nothing about x_i. U computes her share $\langle f(x) \rangle_U \equiv \langle f(x_i + r_S \rangle_S + \langle f(x_i) \rangle_U$ while S simply sets $\langle f(x) \rangle_S \equiv -r_S$.

Case 3: Secret Gaussian f and secret x. We consider the case where f is the popular Gaussian distribution $\mathcal{N}_{\mu,\sigma}$ with secret parameters μ, σ and secret evaluation point x. Though we could use PolyFuncApprox, we design the special-purpose but more efficient Gaussian protocol (9, Appendix A) since the frequent use of Gaussians justifies the additional handwork. We first transform to log-space, i.e., $\log(\mathcal{N}(\mu, \sigma)(x)) = \log((2\pi\sigma^2)^{-1/2}) + (x-\mu)^2 / -2\sigma^2$. μ, x and $1/-2\sigma^2$ are given as normal additive shares and σ is shared in log-space. U and S then compute $(x - \mu)^2$, multiply by $1/-2\sigma$, and finally subtract $\log(\sigma)$ using only the additive shares. We drop the term $\log((2\pi)^{-1/2})$ since it is constant and thus irrelevant for classification. Apart from inexpensive local operations, Gaussian requires only two secure multiplications and rescaling operations which is more efficient than applying PolyFuncApprox. When high accuracy is required, Gaussian is also more efficient than running OtFuncApprox on a fine-grained subsample of $\mathcal{N}(\mu_i, \sigma_i)$ and also more general since x can be secret.

We note that tailoring protocols, e.g., to the special non-linearities typically involved in neural networks as proposed in [29, 55], yields potentially more efficient protocols. Our aim is, however, to present a widely applicable framework rather than optimize our approach towards a single classifier. Our first two protocols for approximation of non-linear functions are in line with this goal, i.e., we deliberately trade performance improvements of specific protocols against the wider applicability of our general protocols.

4.2.5 Backtracking. Backtracking is a common step in dynamic-programming algorithms (e.g., for determining the optimal state sequence in the HMM Viterbi algorithm) and we propose the Backtrack (Prot. 10, Appendix A) to compute this task securely. We assume that only U should learn the final result, the case where S or both should learn the result being straightforward. At the start, the state matrix $M \in \mathbb{N}^{N \times T}$ together with the final state s_T^* is additively shared among U and S. First, S sends $\langle s_T^* \rangle_S$ such that U is able to recombine s_T^*. Starting from $t = T$, U then iteratively obtains $\langle s_{t-1}^* \rangle_S = \langle M_{s_t^*, t} \rangle_S$ via 1-N-OT$_l$ from S and recombines s_{t-1}^* locally. After T sequential OTs, U thus learns $S^* = s_1^*, ..., s_T^*$.

Protocol 1 Secure Hyperplane protocol based on ASS and GC.

Input: U has feature vector $\vec{x} \in \mathbb{R}^n$
S has k models $M_1 = \vec{w}_1, \ldots, M_k = \vec{w}_k$ with $\vec{w}_j \in \mathbb{R}^n$
Output: Class $c^* = C_{Hyperplane}((M_1, \ldots, M_k), \vec{x})$

Initialization:

$U: \quad \langle x_i \rangle_U = \text{F2I}(x_i), \quad \langle w_{j,i} \rangle_U = 0 \qquad \forall i = 1 \ldots n, \forall j = 1 \ldots k$
$S: \quad \langle x_i \rangle_S = 0, \quad \langle w_{j,i} \rangle_S = \text{F2I}(w_{j,i}) \qquad \forall i = 1 \ldots n, \forall j = 1 \ldots k$

Compute distance to each hyperplanes:

$U \Leftrightarrow S: \quad \langle z_j \rangle \leftarrow \text{ScalarProd}(\langle \vec{x} \rangle, \langle \vec{w}_j \rangle) \qquad \forall j = 1 \ldots k$

Determine most probable class:

$U \Leftrightarrow S: \quad \langle c^* \rangle \leftarrow \text{Argmax}(\langle z_1 \rangle, \ldots, \langle z_k \rangle)$
$U \Leftrightarrow S: \quad c^* \leftarrow \text{I2F}(\text{Recombine}(\langle c^* \rangle))$

4.3 Implementation and Evaluation Setup

To thoroughly quantify performance and accuracy of SHIELD, we implement all classifiers, evaluate them on popular real-world datasets from the machine learning community and compare their performance against the fastest approaches in related work.

Implementation. We implement all secure primitives and classifiers in C++ relying on the OT extension library [33] and the ABY framework [32] for creating and evaluating GCs as well as ASS-based multiplication. Besides the OT extensions library and the ABY framework, which are multithreaded, the rest of our implementation realizes only obvious optimizations, e.g., batching of trivially parallel loops in the classifiers.

Experimental Setup. We perform experiments between two desktop machines (Ubuntu 14.04 LTS, Intel i7-4770S with 4 cores at 3.10 GHz, 16 GB RAM) connected over a 1 Gbit/s LAN. We use $l = 64$ bit for our fixed-point number representation (cf. Sec. 4.2.1) and set the symmetric security level t to 128 bit for long-term security. Our results are averaged over 30 independent runs.

5 HYPERPLANE CLASSIFIERS

Hyperplane classifiers [17] compute a linear combination of features in $\vec{x}$ with a trained weight vector $\vec{w}$, i.e., $C_{Hyper}(M, \vec{x}) = f(\sum_{i=1}^{n} w_i \cdot x_i) = f(\vec{w} \cdot \vec{x})$ where the function f maps the inner product to two classes. The classification model is thus given by $M = (\vec{w}, f)$. We can visualize hyperplane classifiers by interpreting $\vec{w}$ as the normal vector of a hyperplane $\vec{w} \cdot \vec{x} - b = 0$ that splits the n dimensional feature space into two parts. Hyperplane classifiers can be generalized to non-linearly separable data using the *kernel trick* [71] and to data with multiple classes through a one-versus-all approach, i.e., training k models where model M_j decides whether a given feature vector $\vec{x}$ belongs to class $c_j \in C$ [20] by $C_{Hyper}(M_1, \ldots, M_k, \vec{x}) = \arg\max_{c_j \in C} (\vec{w}_j \cdot \vec{x})$. With these definitions, we can model classifiers with linear predictor functions, such as SVMs, (multinomial) logistic regression, least squares, perceptrons, and Fisher's linear discriminant [17].

Hyperplane (Prot. 1) securely computes $C_{Hyper}(M_1, \ldots, M_k, \vec{x})$ where S holds the k models M_j and the U holds the feature vector $\vec{x}$. In the first step, the user U and service provider S initialize shares of the weight vectors $\vec{w}_j$ and feature vector $\vec{x}$: Each party uses F2I to initialize shares of its own inputs and sets shares of the other party's inputs to zero (we denote this as a *dummy sharing* since there is no interaction between U and S and no values are actually shared). U and S then compute one secure scalar product (using ScalarProd) for each pair $\vec{w}_j, \vec{x}$ in parallel in one round of communication to improve performance. U and S then invoke Argmax on the shared scalar products $\langle z_j \rangle$ to determine the target class $c^* \in \mathbb{N}$ which can be recombined by U, S, or both. Note that c^* is actually the index of the target class $c_{c^*} \in C$ and for simplicity we assume from now on w.l.o.g. $C = \{1, \ldots, k\}$. Security guarantees of Hyperplane are discussed in Appendix B.2.

	Security level t	WBCD 1 ms	WBCD 40 ms	Credit 1 ms	Credit 40 ms	HAR 1 ms	HAR 40 ms
Bost et al. [20]	80 bit	0.22	0.47	0.30	0.56	0.72	1.01
EzPC [24]	128 bit	0.10	0.30	0.10	0.30	-	-
Hyperplane (this work)	128 bit	0.02	0.35	0.02	0.39	0.03	0.73

Table 1: Comparison of runtimes [s] of secure hyperplane classifiers on different datasets.

5.1 Evaluation

We compare Hyperplane against Bost et al. [20] and *EzPC* [24] on the Wisconsin Breast Cancer Diagnostic (WBCD) dataset [78] with 32 features 2 classes, the Credit Approval (Credit) dataset [25] with 48 features and 2 classes, and the Human Activity Recognition (HAR) dataset [65] with 561 features and 6 classes.

Runtime. We measure runtimes in the offline and online phase for two different networks with Round Trip Time (RTT) of 1 ms (LAN) and 40 ms (WAN) (cf. Tab. 1). On average, Hyperplane is 17.02× faster than Bost et al.'s approach and 5× faster than *EzPC* in the LAN setting and and only slightly slower in the WAN setting. Notably, Hyperplane provides long-term security while Bost et al. provide only an equivalent of 80 bit symmetric security which is widely considered insecure [10]. Hyperplane is especially efficient in the critical online phase improving by 21.01× (LAN) and 1.54× (WAN) on Bost et al. affording very low latency in end-user scenarios where classifications are performed sporadically using idle times for precomputations.

Communication. On all three datasets, Hyperplane requires more communication than the related approaches, e.g., 256.55 kB vs. 54.55 kB (Bost et al.) and 36.00 kB (*EzPC*) on the Credit dataset. Hyperplane's communication overheads are almost completely due to the precomputation of MTs which could be reduced using the optimized *Du-Attalah* protocol [66] (published and proposed in parallel to this work). Furthermore, most of Hyperplane's communications falls into the offline phase while the online phase requires only 14.16 kB compared to 41.16 kB in Bost et al.'s approach.

Accuracy. We measure the numerical accuracy of Hyperplane by classifying 300 randomly selected test vectors and comparing against a reference implementation that operates on double precision plain texts. We observe a very low average absolute numerical error of 2.46×10^{-7} ($\sigma = 2.71 \times 10^{-7}$) and find that Hyperplane predicts exactly the same classes as the insecure reference implementation. From this, we conclude that the classification accuracy of Hyperplane is thus only limited by the quality of the classification model; tuning models is not the goal of this work.

Summary. Hyperplane is a simple, fast, and secure protocol for any classifier with a linear predictor function. In the next section, we generalize Hyperplane to full-fledged ANNs.

Protocol 2 The secure ANN classifier protocol.

Input: U has feature vector $\vec{x} \in \mathbb{R}^n$
S has ANN $M = (\vec{w}_1^1, \vec{w}_2^1 \ldots, \vec{w}_k{}^L, \varphi^1, \ldots, \varphi^L)$
Output: Class $c^* = C_{ANN}(M, \vec{x})$

Precomputation:

S : $\mathcal{P}_l = (a_{11}^l, \ldots, a_{kd}^l), (r_1^l, \ldots, r_k^l) \leftarrow \text{Approx}(\varphi^l, k, E)$ $\forall l = 1 \ldots L$

Initialize shares:

U : $\langle \vec{w}_{j,i}^l \rangle_U = 0, \langle x_i \rangle_U = \text{F2I}(x_i), \langle \mathcal{P}_l \rangle_U = 0$ $\forall l, i, j$

S : $\langle w_{j,i}^l \rangle_S = \text{F2I}(w_{j,i}^{(l)}), \langle x_i \rangle_S = 0$ $\forall l, i, j$

$\langle \mathcal{P}_l \rangle_S = \text{F2I}(a_{11}^l), \ldots, \text{F2I}(a_{kd}^l), \text{F2I}(r_1^l), \ldots, \text{F2I}(r_k^l)$ $\forall l = 1 \ldots L$

Feed-forward through all layers $l = 1 \ldots L$:

$U \Leftrightarrow S$: $\langle e_i^l \rangle \leftarrow \text{ScalarProd}(\langle \vec{y}^{\,l-1} \rangle, \langle \vec{w}_i^l \rangle)$ $\forall i = 1 \ldots m_l$

$U \Leftrightarrow S$: $\langle y_i^l \rangle \leftarrow \text{PolyFuncApprox}(\langle e_i^l \rangle, \langle \varphi^l \rangle)$ $\forall i = 1 \ldots m_l$

Determine most probable class:

$U \Leftrightarrow S$: $\langle c^* \rangle \leftarrow \text{Argmax}(\langle y_1^L \rangle, \ldots, \langle y_k^L \rangle)$

$U \Leftrightarrow S$: $c^* \leftarrow \text{I2F}(\text{Recombine}(\langle c^* \rangle))$

6 ARTIFICIAL NEURAL NETWORKS

ANNs [16, 26, 69] are composed of many individual *artificial neurons* organized in multiple *layers*. Any single neuron computes a weighted sum of its inputs, the *excitation level*, and fires when it exceeds a threshold. In *feed-forward networks*, a neuron on an intermediate layer l takes inputs only from neurons on the previous layers $l-1$ and passes its output on to neurons on the subsequent layer $l+1$. The classification result is then read from the output layer $l = L$ with one neuron per class. Feed-forward ANNs can be modeled as a function that is composed of the activation functions of the individual neurons [69, pp. 567-570]: The i^{th} neuron on layer $l \geq 1$ is modeled by $y_i^l = \varphi^l(\sum_{j=1}^{m_{l-1}} w_{j,i}^l \cdot y_j^{l-1}) = \varphi^l(\vec{w}_i^{\,l} \cdot \vec{y}^{\,l-1})$ where $\vec{w}_i^{\,l}$ are the *synaptic weights* between the i^{th} neuron on the l^{th} layer and all neurons on the layer $l-1$, $\vec{y}^{\,l-1}$ the outputs of those neurons, and $\varphi^l(\cdot)$ the activation function for all neurons on layer l. An ANN model is thus defined by $M = (\vec{w}_1^1, \vec{w}_2^1 \ldots, \vec{w}_k^L, \varphi^1, \ldots, \varphi^L)$ and the classification rule is given by $C_{ANN}(M, \vec{x}) = \arg\max_{j \in C} y_j^L$. In its simplest form, an ANN consists of a single neuron which corresponds almost exactly to the previously introduced hyperplane classifier. A single-layer perceptron, just as hyperplane classifiers, is limited to binary classification problems and linearly separable data [69, pp. 573-574]. Building ANNs with many neurons and multiple hidden layers, referred to as *deep learning*, overcomes this limitation and tackles much more complex classification problems.

Ann (Prot. 2) computes ANNs securely. S holds the ANN and first computes approximation parameters for the activation functions φ^l for use in PolyFuncApprox (cf. Sec. 4.2.4). As before for Hyperplane, U and S then dummy-share all model parameters and inputs. On these shares, U and S first securely compute the excitation level e_i^l (for each layer $l = 1 \ldots L$ and each of neuron $i = 1, \ldots, m_l$) using ScalarProd then evaluate the (secret) activation function φ^l on the shared evaluation point $\langle e_i^l \rangle$ using PolyFuncApprox, obtaining additive shares of the neuron's output, i.e., $\langle y_i^l \rangle$. Note that we can compute the output of all neurons on the same layer in parallel and batch communication to increase performance. Finally, U and S invoke Argmax on $\langle y_1^L \rangle$,...,$\langle y_k^L \rangle$ to determine the target class c^*. We discuss security of Ann in Appendix B.2.

	Sec. level t	Arbit. act.func.	Out-sourc.	MINST 1 ms	MINST 40 ms	MINST 100 ms
Cryptonets [29]	128 bit	✗	✗	297.65	297.73	297.85
DeepSecure [68] †	128 bit	✓	✓	9.67	-	-
SecureML [55] ‡	0 bit	✗	(✓)	4.88	-	18.37
Chameleon [66] ⋆†	128 bit	✓	(✓)	2.70	-	7.90
MiniONN [52]	128 bit	✓	(✓)	1.28	-	-
EzPC [24]	128 bit	✓	(✓)	0.60	1.60	-
Gazelle [43]	128 bit	✓	✗	0.03	-	-
Ann (this work)	128 bit	✓	✓	0.60	4.98	12.08

Table 2: Comparison of runtimes [s] of secure ANN classifiers on the MNIST dataset for different network scenarios. ⋆ Requires a trusted third party. † Similar network but uses ReLU activation function. ‡ Slightly more complex network with three fully-connected layers.

6.1 Evaluation

Since code for related works is not (yet) open-source, we compare directly against the results from the respective papers but note that they were obtained on comparable yet different machines. To maintain comparability as far as possible, we select only those results obtained on the same dataset, i.e., MNIST [50], and with the same convolutional neural network (CNN) architecture described in the initial work of Dowlin et al. [29].

Runtime. Tab. 2 summarizes the runtimes of Ann and previous works for classification of a single image. The *Cryptonets* approach [29] optimizes for throughput and allows batching up to 4096 images into one ciphertext at no additional costs. Being based on FHE, *Cryptonets* is a two-rounds protocol and thus scales nicely to networks with higher latencies. FHE, however, puts high load on the user which cannot be outsourced as we will discuss further in Sec. 9. *DeepSecure* is a purely GCs-based approach and shows how to securely outsource the client's protocol part to an untrusted third party. In comparison, Ann has a 16.1× lower classification latency. *SecureML* [55], *Chameleon* [24], and *MiniONN* [52] are hybrid approaches that build on different combinations of OT, GC, ASS, HE, and GMW. SecureML only considers linear activation functions, while *Chameleon* depends on a semi-trusted third party dealer and thus provides weaker security guarantees than the other approaches. None of these approaches explicitly presents outsourcing protocols, although the underlying STC techniques conceptually lend themselves to outsourcing. In comparison, Ann is 8.1×, 4.5×, and 2.1× faster and facilitates highly efficient secure outsourcing (cf. Sec. 9). *EzPC* [24] is a competitive approach that outperforms Ann in slower networks (while SHIELD, in turn, outperforms *EzPC* on hyperplane and Naive Bayes classifiers). The most recent approach, *Gazelle* [43], outperforms Ann by 20× but cannot be outsourced due to its use of FHE.

Communication. The *Cryptonets* approach requires 372.20 MB of communication to classify up to 4092 images but unfortunately requires the same high amount for classification of a single image. While no communication overheads are reported for *SecureML*, *DeepSecure* requires even more with 791.00 MB. *MiniONN*, and *EzPC* reduce communication to 70.00 MB and 47.60 MB, respectively, for

a single image. *Chameleon* optimizes especially for low communication and manages to reduce overheads to 8.60 MB. Using highly efficient packing and Single-Instruction-Multiple-Data (SIMD) strategies, *Gazelle* reduces communication to 0.50 MB. With 76.36 MB, Ann's overheads are one and two order of magnitude higher than *Chameleon's* and *Gazelle's*, but competitive w.r.t. the other approaches. As for Hyperplane, Ann's communication overheads are mostly due to MT precomputation and could be reduced by employing the improved *Du-Attalah* protocol [66] for precomputing MTs which is optimized for low communication overheads. Precomputing MTs also renders the online phase of Ann very efficient with only 2.20 MB of communication.

Accuracy. We measure the numerical accuracy of Ann by classifying 300 randomly selected test vectors and compare against a reference implementation on plaintexts. We measure an average absolute numerical error of 8.60×10^{-2} ($\sigma = 7.42 \times 10^{-2}$) which is four orders of magnitude higher than for Hyperplane, but still low enough such that Ann predicts exactly the same classes as the reference implementation. The increase is due to the higher multiplicative depth of ANNs, i.e., the numerical errors grow with each layer. Networks with many more layers may thus require a deterministic rounding strategy, e.g., as proposed in [35].

Summary. Ann is a secure protocol for feed-forward neural networks optimized for classification latency and support for arbitrary (non-linear) activation functions. In the next two section, we now focus on approaches that are based on probability theory.

7 NAIVE BAYES

A Naive Bayes classifier is a conditional probability model that assigns probabilities $P(C = c_j | X = \vec{x})$ for all classes $c_j \in C$ to all possible feature vectors $\vec{x}$ [69, Chap. 14]. Classifications are computed by selecting the most probable class, i.e., $C_{Bayes}(M, \vec{x}) = \arg\max_{c_j \in C} p(c_j|\vec{x})$. Since it is often infeasible to learn the *posteriors* $p(c_j|\vec{x})$ directly from the data [7], e.g., for very large or high-dimensional feature spaces, the Bayes theorem is usually applied to compute the posteriors from the *likelihoods* $p(\vec{x}|c_j)$, the *priors* $p(c_j)$, and the *evidence* $p(\vec{x})$ which can be better learned from the training data D, *naively* assuming features $x_i \in \vec{x}$ (modeled by random variable X_i) to be conditionally independent from each other feature, i.e., $P(X_i, X_j|C) = P(X_i|C) \cdot P(X_j|C)$. The classification model M is then given by the distribution of the likelihoods, priors, and evidence, i.e., $M = (P(X_1|C), ..., P(X_n|C), P(C), P(X))$. This classifier is called *naive* because the central assumption of conditional independence actually does not hold for most real-world datasets. Perhaps surprisingly, Naive Bayes classifiers have been shown to still provide good results in real-world applications [67, 83].

NaiveBayes (Prot. 3) is a secure Naive Bayes classifier in logspace representation. Since secure multiplications are expensive and introduce numerical errors, we transform computations of the posteriors into logspace, i.e., $\log(p(c_j|\vec{x}) = \sum_{i=1}^{n} \log(p(x_i|c_j) + \log(p(c_j)) - \log(p(x_i))$. The representation is advantageous since it contains only additions which can be computed much more efficiently and accurately over additive shares. Note that even on plaintexts, Naive Bayes is often computed in logspace to increase the numerical stability. At the start of NaiveBayes, U holds the feature vector $\vec{x}$ and S has the Naive Bayes classification model consisting of the probability mass functions $P(X|C), P(X), P(C)$. S first transforms the priors $p(c_j)$ and likelihoods $p(x_i|c_j)$ to scores with fixed-point precisions using f2li and both parties initialize shares of the priors $\langle \hat{p}(c_j) \rangle$ using dummy sharing. In the next step, U and S use OtFuncApprox to compute shares $\langle p(x_i|c_j) \rangle$ of the likelihoods. Computing shares of the posteriors $\langle \hat{p}(c_j|\vec{x}) \rangle$ is then a simple matter of summing the shares of the likelihoods and the shared priors. Finally, U and S determine shares of the class that maximizes the posterior scores using Argmax. Note that we drop the evidence $p(x_i)$ since it is constant for a fixed feature vector $\vec{x}$ and thus only linearly scales the scores $\hat{p}(c_j|\vec{x})$ which does not change the arg max c^*. A security discussion of NaiveBayes is given in Appendix B.2.

Protocol 3 Secure NaiveBayes protocol based on ASS and GC.

Input: U has feature vector $\vec{x}$
S has Naive Bayes classification model $M = (P(X|C), P(X), P(C))$
Output: Class $c^* = C_{NaiveBayes}(M, \vec{x})$

Initialize shares:

U : $\langle \hat{p}(c_j) \rangle_U = \texttt{Logzero}$ $\langle \hat{p}(x_i|c_j) \rangle_U = \texttt{Logzero}$ $\forall i, j$

S : $\langle \hat{p}(c_j) \rangle_S = \text{f2li}(p(c_j))$ $\langle \hat{p}(x_i|c_j) \rangle_S = \text{f2li}(P(X_i = x_i|c_j))$ $\forall i, j$

Compute posteriors:

$U \Leftrightarrow S$: $\langle \hat{p}(x_i|c_j) \rangle \leftarrow \text{OtFuncApprox}(\langle P(X_i|c_j) \rangle, x_i)$ $\forall i, j$

U, S : $\langle \hat{p}(c_j|\vec{x}) \rangle = \sum_{i=1}^{n} \langle \hat{p}(x_i|c_j) \rangle + \langle \hat{p}(c_j) \rangle$ $\forall j$

Determine most probable class:

$U \Leftrightarrow S$: $\langle c^* \rangle \leftarrow \text{Argmax}(\langle \hat{p}(c_j|\vec{x}) \rangle)$

$U \Leftrightarrow S$: $c^* \leftarrow \text{Recombine}(\langle c^* \rangle)$

	Security level t	WBC 1 ms	WBC 40 ms	Nursery 1 ms	Nursery 40 ms	Audiology 1 ms	Audiology 40 ms
Bost et al. [20]	80 bit	0.38	0.75	0.81	1.90	3.35	5.79
EzPC [24]	128 bit	-	-	0.10	0.40	1.50	2.90
NaiveBayes (this work)	128 bit	0.02	0.27	0.02	0.31	0.03	0.58

Table 3: Comparison of runtimes [s] of secure Naive Bayes classifiers on different datasets.

7.1 Evaluation

We compare NaiveBayes against Bost et al. [20] and *EzPC* [24] on the original Wisconsin Breast Cancer (WBC) dataset [77] (9 features, 2 classes), the Nursery dataset [74] (9 features, 5 classes), and the Audiology dataset [63] (70 features, 24 classes).

Runtime. We measure offline and online runtimes in LAN and WAN settings (cf. Tab. 3). As for Hyperplane and Ann, NaiveBayes performs best in the fast LAN scenario where it outperforms related works by 63.47× (Bost et al.) and 27.5× (*EzPC*) on average while still achieving a notable improvement of 6.24× and 3.16× in the WAN scenario. We observe that a large fraction of the overheads in Bost et al.'s approach is due to onetime overheads which would amortize over larger batches of classifications. Not considering these onetime overheads, our NaiveBayes still achieves a 37.24× and 4.11× higher throughput than Bost et al. for the LAN and WAN scenario, respectively. NaiveBayes's improvements are due to the efficient OtFuncApprox primitive for sampling probability mass distributions and the hybrid protocol design based on ASS and GCs in contrast to the costly HE primitives used in Bost et al.'s approach.

Communication. NaiveBayes requires significantly less communication than the other two approaches, e.g., only 0.74 MB on the Audiology dataset compared to 1.91 MB for Bost et al. and 37.00 MB for *EzPC*. Note that communication for Bost et al. approximately triples when using 3072 bit keys, i.e., an equivalent to the 128 bit symmetric security of NaiveBayes and *EzPC*.

Accuracy. We measure an average absolute numerical error of 6.37×10^{-8} ($\sigma = 6.70 \times 10^{-8}$) over 300 random test cases, i.e., results are practically indistinguishable from a reference implementation operating with double precision on plaintexts. The reason for the high accuracy of NaiveBayes is the logspace transformation which replaces multiplications by additions which are numerically more accurate and stable in our number representation.

Summary. NaiveBayes clearly outperforms prior works due to the use of efficient ASS-techniques in log-space. Next, we apply these techniques to more complex problems on HMMs.

8 HIDDEN MARKOV MODELS

An HMM is defined by the tuple $\lambda = (S, A, V, B, \pi)$. The set $S = \{s_1, ..., s_N\}$ are the possible *internal states* of the HMM with $A \in \mathbb{R}^{N \times N}$ the *state transition matrix*, i.e., $a_{ji} = p(s_i|s_j)$ is the probability that the HMM moves from state s_j into state s_i. The states of the HMM are hidden and cannot be observed directly but only inferred from the *emissions* the HMM outputs depending on its current state. The alphabet of emissions is defined by $V = \{v_1, ..., v_M\}$ with $B \in \mathbb{R}^{N \times M}$ the *emission probability matrix*, i.e., $b_i(v_j) := b_{ij} = p(v_j|s_i)$ is the probability that the HMM emits v_j in state s_i. Finally, the *initial state distribution* $\pi \in \mathbb{R}^N$ defines the probabilities $\pi_i = p(s_i)$ that the HMM's initial state is s_i. The output of the HMM is a sequence of emission symbols $O = o_1...o_T \in V^{1\times T}$ referred to as an *observation sequence* (each o_i could be viewed as a separate feature vector $\vec{x}^{\,i}$ in the simple classification setting).

Two main problems are associated with HMMs. *Filtering* asks for the probability $P(O|\lambda)$ that an HMM λ generated an observation sequence O. Filtering is solved using the *Forward* algorithm, which i) initializes $\alpha_1(i) = \pi_i \cdot b_i(o_1), \forall i$, ii) recursively computes the forward variables $\alpha_t(i) = \sum_{j=1}^{N} \alpha_{t-1}(j) \cdot a_{ji} \cdot b_i(o_t), \forall t, i$, and iii) outputs $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$. *Decoding*, searches for the most probable sequence of hidden states S^* of the HMM λ for emitting the observation sequence O and its probability $p(O, S^*|\lambda)$. This problem is solved by the *Viterbi* algorithm, which i) initializes $\alpha_1(i) = \pi_i \cdot b_i(o_1), \forall i$ (as before), ii) recursively computes the forward variables $\alpha_t(i) = \max_{j=1}^{N} \alpha_{t-1}(j) \cdot a_{ji} \cdot b_i(o_t), \forall t, i$ and the backtracking matrix $v_t(i) = \arg\max_{j=1}^{N} \alpha_{t-1}(j) \cdot a_{ji}, \forall t, i$, and iii) finally outputs the optimal state sequence $s^*_{t-1} = v_t(s^*_t), \forall t$ and its probability $P(O, S^*|\lambda), s^*_t = \arg\max_{i=1}^{N} \alpha_T(i)$.

In both algorithms, the probabilities $\alpha_t(i)$ decrease with each iteration which quickly causes underflows and numerical instability [30, 64]. Rabiner [64] proposes to normalize $\alpha_t(i)$ after each iteration while Durbin et al. [30] propose to compute in logarithmic space. We refer to probabilities in logspace as *scores* with $\log(p(O|\lambda))$ and $\log(p(O, S^*|\lambda))$ the Forward and Viterbi score, respectively.

Viterbi (Prot. 4) computes the Viterbi algorithm in logspace securely using EvalProb, MaxArgmax, and Backtrack as building blocks. At the start, U holds the observation sequence $O = o_1, ..., o_T$ and S holds the HMM λ. In the initialization phase, U and S compute shares of the emission scores $\hat{b}_i(o_1)$ via EvalProb and add the dummy shares of the initial state scores $\hat{\pi}_i$ locally. The goal of the following recursion phase is to compute the forward variables $\alpha_t(i)$ in logspace, i.e., the probability of the optimal partial state sequence given only the partial observation sequence $o_1o_2...o_t$ up to time step t. Additionally, we need to keep track of this optimal state sequence in the variables $v_t(i)$. These steps are given in logspace by $\hat{\alpha}_t(i) = \max_{s_j \in S} \left(\hat{\alpha}_t(j) + \hat{a}_{ji}\right) + \hat{b}_i(o_1)$ and $v_t(i) = \arg\max_{s_j \in S} \left(\hat{\alpha}_t(j) + \hat{a}_{ji}\right)$ and can be efficiently combined in MaxArgmax such that U and S only need to locally add $\langle \hat{b}_i(o_t) \rangle$ to obtain the desired additive sharing $\langle \hat{\alpha}_t(i) \rangle$ of the forward scores $\hat{\alpha}_t(i)$. Since we also obtain additive shares of the maximum argument from MaxArgmax, we can directly set the entry $v_t(i)$ in the backtracking matrix v. Finally, U and S first invoke MaxArgmax on $\hat{\alpha}_T(1), ..., \hat{\alpha}_T(N)$ to compute the Viterbi score $\hat{P}(O, S^*|\lambda)$ and the optimal end state s^*_T, then invoke Backtrack on s^*_T and $\langle v \rangle$ to let U reconstruct the optimal state sequence S^* that led to s^*_T. We discuss the security of Viterbi in Appendix B.2.

Protocol 4 Secure Viterbi protocol based on ASS, GC, and OT.

Input: U has $O \in V^{1\times T}$, S has $\lambda = (S, V, A, B, \pi)$
Output: Viterbi score $\hat{P}(O, S^*|\lambda)$ and Viterbi path $S^* \in S^{1\times T}$

Initialization:

$U \Leftrightarrow S$: $\langle \hat{b}_i(o_t) \rangle \leftarrow$ OtFuncApprox(o_t, B_i) $\quad \forall t, i$
U: $\langle \hat{\pi}_i \rangle_U =$ Logzero $\quad \langle \hat{a}_{ji} \rangle_U =$ Logzero $\quad \forall j, i$
S: $\langle \hat{\pi}_i \rangle_S =$ f2li(π_i) $\quad \langle \hat{a}_{ji} \rangle_S =$ f2li(a_{ji}) $\quad \forall j, i$
U, S: $\langle \hat{\alpha}_1(i) \rangle = \langle \hat{b}_i(o_1) \rangle + \langle \hat{\pi}_i \rangle$

Recursion: For $2 \le t \le T$, $1 \le i \le N$

U, S: $\langle \vec{\hat{\alpha}}'_t(i) \rangle = \left(\langle \hat{\alpha}_{t-1}(1) + \hat{a}_{1i} \rangle, ..., \langle \hat{\alpha}_{t-1}(N) + \hat{a}_{Ni} \rangle\right)$
$U \Leftrightarrow S$: $\langle \hat{\alpha}'_t(i) \rangle, \langle v_t(i) \rangle \leftarrow$ MaxArgmax$\left(\langle \vec{\hat{\alpha}}'_t(i) \rangle\right)$
U, S: $\langle \hat{\alpha}_t(i) \rangle = \langle \hat{\alpha}'_t(i) \rangle + \langle \hat{b}_i(o_t) \rangle$

Termination:

$U \Leftrightarrow S$: $\langle \hat{P}(O, S^*|\lambda) \rangle, \langle s_T \rangle \leftarrow$ MaxArgmax$\left(\langle \hat{\alpha}_T(1) \rangle, ..., \langle \hat{\alpha}_T(N) \rangle\right)$
$U \Leftarrow S$: $\hat{P}(O|\lambda) \leftarrow$ Recombine$\left(\langle \hat{P}(O|\lambda) \rangle\right)$
$U \Leftrightarrow S$: $S^* \leftarrow$ Backtrack$(\langle v \rangle, \langle s^*_T \rangle)$

8.1 Evaluation

We evaluate Viterbi in three use cases, i) secure bioinformatics services, ii) secure speech recognition, and iii) secure localization.

Secure Bioinformatics Services. We consider the secure bioinformatics service described in [37, 86] where we match a given protein sequence against the *Pfam* [1] database of HMMs that model protein families (e.g., relating to certain phenotypes and diseases). Note that Pfam contains *profile HMMs* that feature a special architecture and sparsely connected state space which significantly speeds up Forward and Viterbi computation. Since Forward and Viterbi compute identical matches in this use case, we also consider the Forward algorithm in our comparison. As summarized in Tab. 4, Viterbi outperforms Franz et al.'s Forward by 14.11× and Viterbi by 48.29× and requires 2.25× and 1.28× less communication even despite providing higher security–using only 80 bit of security reduces Viterbi's overheads by another 37 %. *Priward* [86] is a more efficient secure Forward algorithm that is based on ASS and GC

	Sec. level t	SH3_1 L=48	Ras L=162	BID L=192	IDO L=408	3HBOH L=689
Franz et al. [35] (Fwd.)	80 bit	22.0	298.0	449.0	-	-
Franz et al. [35] (Vit.)	80 bit	94.0	933.0	1357.0	-	-
Priward [86] (Fwd.)	128 bit	13.4	137.4	187.8	857.5	2310.6
Viterbi (this work)	128 bit	1.8	20.8	28.4	142.2	375.6

Table 4: Comparison of runtimes [s] of secure Bioinformatics use case on the Pfam database. L denotes the length of a profile HMM with a total of $N = 3L + 4$ states.

	Security	N	T	Runtime	Comm.
Pathak et al. [59]	80 bit	5	96	$\geq$ 785.0 s	-
Aliasgari et al. [4] (2PC)	80 bit	6	96	$>$ 400 min	-
Aliasgari et al. [4] (MPC)	2/3 honest	6	96	$\approx$ 23.0 s	-
Viterbi (this work)	128 bit	10	100	2.4 s	68.12 MB

Table 5: Comparison of secure Viterbi protocols on fully connected HMMs with N states and T observations.

similar as Viterbi. Still, Viterbi is 5.67× faster and requires 4.74× less communication by replacing *Priward's* complex logsum primitive by our highly efficient Argmax primitive. Finally, we measure a low relative numerical error of 2.7×10^{-2} % averaged over all models and sequences.

Secure Speech Recognition. We consider the secure speech recognition use case proposed in [4, 59] where HMMs encode short words and the observation sequence an utterance. In contrast to the previous use case, HMMs are now fully connected. We compare runtimes in Tab. 5. Pathak et al. [59] securely compute Forward and Aliasgari et al. [4] compute Viterbi in the two- and multi-party setting on very small HMMs and only with short-term security. On an HMM with more states (note that the complexity of the Forward and Viterbi algorithms is quadratic in N), considering more observations (complexity is linear in T) and at a much higher security level, Viterbi still outperforms these works by 327× (Pathak et al.), 10 000× (Aliasgari et al.'s two-party setting), and 9.58× (Aliasgari et al.'s multi-party setting). Aliasgari et al. justify the huge overheads of their two-party protocol arguing that the HMM itself must be hidden even from the service provider and stored only in encrypted form using expensive threshold-HE. Notably, this is a special case of our more general outsourcing problem scenario (cf. Fig. 2, Sec. 2.1) and is thus also be covered by Viterbi as detailed in section 9. Since previous works [4, 59] do not evaluate communication overheads, we cannot provide a comparison. Finally, we measure a very low relative numerical error of 9.73×10^{-6} % averaged over models and sequences with different length, i.e., $N = 10, \ldots, 100$ and $T = 10, \ldots, 100$. The three order of magnitudes lower error is due to the significantly smaller model sizes and sequence lengths compared to the previous bioinformatics use case.

Secure Localization. In [87], users are securely tracked by matching signal measurements against an indoor signal propagation and human movement model using a secure Viterbi algorithm. The authors aim to provide fresh location updates every 10 s and scale the underlying HMM accordingly to $N = 160$ states with at most $N' = 5$ predecessors. In contrast, Viterbi (we substitute OtFuncApprox by Gaussian to compute emission probabilities according to [87]) can compute updates at the same frequency on much larger HMMs of $N = 900$ states with $N' = 90$ predecessors which greatly increases the localization accuracy as the indoor state space can be segmented into finer parts (as we have more states N available) and the human mobility model can be much more refined (since we may consider more predecessors N').

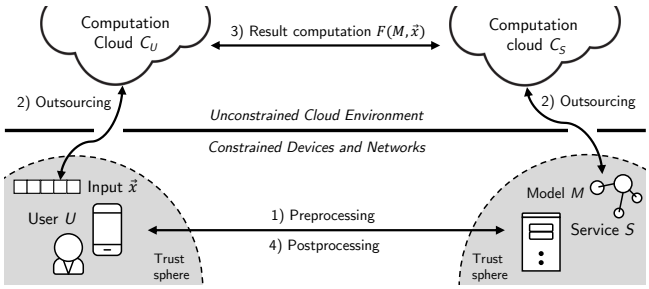

Figure 2: U wishes to classify data $\vec{x}$ by S's model M but cannot execute STC protocols due to resource constraints. SHIELD allows securely outsourcing these computations.

Summary. Viterbi is a highly efficient secure protocol for HMM-based pattern recognition with applications in different domains. It is feasible on very large models but may overtax mobile users. With outsourcing, we propose a solution in the next section.

9 OUTSOURCING

Arguably, secure classification could be too costly to be executed in constrained environments. As a solution, we show how U and S can securely outsource computations to untrusted clouds according to Fig. 2: 1) User U and service S engage in a short preprocessing, 2) individually outsource encrypted data to the cloud peers, 3) wait for the cloud peers to obliviously compute the encrypted result, and 4) decrypt and postprocess the result.Besides unburdening U and S from most overheads, outsourcing affords disruption tolerance, i.e., U and S need to be online and available only at the start and end of the computation. To present a real alternative for mobile users, an outsourcing protocol must fulfill the following requirements: First, the preprocessing, outsourcing, and postprocessing overheads for U or S must be minimized. Second, the overheads for cloud peers must remain feasible. Third, outsourcing must remain secure against C_U and C_S which we assume to be semi-honest and non-colluding since cloud providers typically have strong incentives to guard their reputation [2, 44, 60].

Outsourcing Building Blocks. Most of our building blocks, i.e., Rescale, ScalarProd, MaxArgmax, PolyFuncApprox, and Gaussian, take only additive shares as inputs, learn nothing from intermediate values (cf. Appendix B), and output additive shares of the result. This renders outsourcing easy and efficient: Without any preprocessing, U and S just send their individual shares to C_U and C_S which execute the protocol exactly as presented and return the shares of the result back to U and S for postprocessing.

OtFuncApprox and Backtrack require cleartext knowledge of some inputs which prevents outsourcing for U. We could still efficiently outsource both primitives using generic GCs but this incurs significantly higher overheads on the clouds. Alternatively, we can move the very efficient original OtFuncApprox primitive to the preprocessing phase, e.g., as we already did for Viterbi with the goal of batching the calls to OtFuncApprox into a single round.

In summary, all but the OtFuncApprox and Backtrack primitives can be outsourced by providing the inputs as shares to the

computation parties, who then compute shares of the result according to the original protocols. We emphasize two important points: First, sharing and recombining are very cheap operations that require no preprocessing – computing these primitives is feasible even on very constrained devices. Second, there is no need to involve U or S in between successive executions of outsourceable primitives – the cloud peers just keep hold of the shared outputs and input them to the subsequent primitive, and so on. This argument is the basis for outsourcing SHIELD's classifiers.

Outsourcing Classifiers. Outsourcing Hyperplane and Ann is straightforward, since all underlying primitives can be outsourced. E.g., for outsourcing Hyperplane, U creates shares $\langle x_i \rangle_{C_U}$ and $\langle x_i \rangle_{C_S}$ of $\text{F2I}(x_i)$ and sends them to C_U and C_S while S does the same with the weights $w_{j,i}$. C_U and C_S compute Hyperplane as described in Prot. 1 and return $\langle c^* \rangle_{C_U}$ and $\langle c^* \rangle_{C_S}$ to U and S.

Outsourcing NaiveBayes requires to precompute all required shares $\langle \hat{p}(x_i|c_j) \rangle$ in the preprocessing as the employed OtFuncApprox protocol cannot be efficiently outsourced. U and S then add the derived shares locally and directly provide shares of the posteriors $\langle \hat{p}(c_j|\vec{x}) \rangle$ to the computation peers who then only compute Argmax and provide back the shared result $\langle c^* \rangle$. While this outsourcing schemes is clearly less efficient in unburdening U and S than the previous two, OtFuncApprox causes only very low overheads which are feasible even on mobile devices. In contrast, NaiveBayes with an underlying Gaussian distribution (Prot. 9) can be fully outsourced: U shares x_i and S shares μ_i, σ_i and $1/(-2\sigma_i)$ to C_U and C_S which compute NaiveBayes and Gaussian on these shares and provide back shares of the result.

Outsourcing Viterbi also requires to precompute all invocations of OtFuncApprox to compute shares of the emission scores $\langle \hat{b}_i(o_t) \rangle$. In the outsourcing phase, U then distributes $\langle \hat{b}_i(o_t) \rangle_U$ to C_U, while S provides $\langle b_i(o_t) \rangle_S$ to C_S. S further provides shares $\langle \hat{\pi}_i \rangle$ of the prior state distribution and shares $\langle \hat{a}_{ji} \rangle$ of the transition scores to C_U and C_S. Given these shares, C_U and C_S can then compute Viterbi as specified in Prot. 4 (only leaving out the invocation of OtFuncApprox). The backtracking phase cannot be outsourced (as it requires U to know each s_t^* in clear) and must be executed between U and S in the postprocessing phase.

9.1 Evaluation of Outsourcing

We evaluate the overheads for user U on an LG Nexus 5 smart phone (Android 4, 2.26 GHz CPU, 16 GB RAM) and service S on a desktop machine (Ubuntu 14.04 LTS, Intel i7-4770S with 4 cores at 3.10 GHz, 16 GB RAM). The smart phone is connected through a 300 Mbit/s WiFi network and the server is on a 1 Gbit/s LAN. We further assume the largest problem instance considered in our previous evaluation. We summarize the runtime and communication overheads for U in Tab. 6 (results for S in Tab. 8 in Appendix C).

For Hyperplane, Ann, and NaiveBayes outsourcing is highly efficient and clearly feasible on constrained mobile devices. For Viterbi, runtimes for preprocessing and outsourcing of are clearly feasible on mobile devices while the communication overheads of the preprocessing phase might overtax slower networks or strain the user's data plan. However, we considered the largest HMM and observation sequence from our evaluation – overheads for smaller models range only in the order of kB to a few MB. For all classifiers, we observe that the outsourcing overheads for the user are always one or two orders of magnitude smaller than for the service provider (cf. Appendix C). This is desired since we expect that users need to outsource more frequently than service providers who usually host their backends in the cloud already.

	Preprocessing		Outsourcing	
	Processing	Communication	Processing	Communication
Hyperplane	-	-	0.27 ms	4.49 kB
Ann	-	-	0.40 ms	6.68 kB
NaiveBayes	2.60 ms	0.13 MB	1.24 µs	0.10 kB
Forward	1.02 s	113.99 MB	49.17 ms	3.80 MB
Viterbi	1.02 s	113.99 MB	49.17 ms	3.80 MB

Table 6: Runtime and communication for U for outsourcing the largest considered problem instances.

To put these numbers into relation, we shortly revisit the comparison against the best performing secure ANN from related work, i.e., *Gazelle* [43]: In the standard setting, *Gazelle* outperforms Ann by 20× but cannot be outsourced (cf. Sec. 6.1). In an outsourcing setting, however, Ann requires 75× less computation and 5000× less communication on the client. Here, we thus trade a higher load on the unconstrained cloud peers against a much lower load on the constrained client, which clearly demonstrates the benefits of designing secure classification protocols for outsourcing.

10 CONCLUSION

We introduced SHIELD, an efficient framework for secure classification upon which we built four different classes of classifiers. Even though being primarily designed for generality and wide applicability, our thorough evaluation shows that SHIELD has competitive performance even compared to approaches specialized to a single classifier such as ANNs. We noted that despite the significant improvements made by SHIELD and related work, processing and, especially, communication overheads of secure classification may still overtax constrained devices and networks. As a solution, we designed SHIELD from the ground up to enable secure and efficient outsourcing to untrusted clouds. The evaluation shows that our proposed outsourcing protocols for SHIELD's choice of classifiers are feasible even for very constrained devices. Exciting future work includes applying our results to different classifiers and use cases as well as to the problem of secure training of the machine learning models that we assumed given in this work.

ACKNOWLEDGMENTS

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under funding reference number 16KIS0443. The responsibility for the content of this publication lies with the authors, who would also like to thank the German Research Foundation DFG for the kind support within the Cluster of Excellence "Integrative Production Technology for High-Wage Countries".

REFERENCES

[1] 2015. Pfam Database, version 29.0. http://pfam.xfam.org/.
[2] Aydin Abadi, Sotirios Terzis, and Changyu Dong. 2015. O-PSI: delegated private set intersection on outsourced datasets. In *IFIP International Information Security Conference*. Springer, 3–17.
[3] Mehrdad Aliasgari and Marina Blanton. 2013. Secure Computation of Hidden Markov Models. In *SECRYPT*.

[4] Mehrdad Aliasgari, Marina Blanton, and Fattaneh Bayatbabolghani. 2016. Secure computation of hidden Markov models and secure floating-point arithmetic in the malicious model. *International Journal of Information Security* (2016), 1–25.

[5] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. 2013. Secure Computation on Floating Point Numbers. In *NDSS*.

[6] Amazon. 2018. Machine learning on AWS. https://aws.amazon.com/machine-learning/.

[7] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, and Constantine D. Spyropoulos. 2000. An Experimental Comparison of Naive Bayesian and Keyword-based Anti-spam Filtering with Personal e-Mail Messages. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '00)*. ACM, New York, NY, USA, 160–167. https://doi.org/10.1145/345508.345569

[8] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 535–548.

[9] Shai Avidan and Moshe Butman. 2006. Blind Vision. In *Proceedings of the 9 th European Conference on Computer Vision*.

[10] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. 2007. Recommendation for Key Management - Part 1: General (Revised). In *NIST Special Publication 800-57*. NIST.

[11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security* 6, 2 (2011), 452–468.

[12] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*. Springer, 420–432.

[13] Donald Beaver. 1995. Precomputing oblivious transfer. In *Annual International Cryptology Conference*. Springer, 97–109.

[14] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. 2013. Efficient garbling from a fixed-key blockcipher. In *IEEE SP*. IEEE, 478–492.

[15] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/62212.62213

[16] Christopher M Bishop. 1995. *Neural networks for pattern recognition*. Oxford university press.

[17] Christopher M Bishop. 2006. Pattern recognition. *Machine Learning* 128 (2006), 1–58.

[18] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2016. Practical Secure Aggregation for Federated Learning on User-Held Data. *arXiv preprint arXiv:1611.04482* (2016).

[19] Joppe W Bos, Kristin Lauter, and Michael Naehrig. 2014. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics* 50 (2014), 234–243.

[20] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data.. In *NDSS*.

[21] Ran Canetti. 2000. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY* 13, 1 (2000), 143–202.

[22] Octavian Catrina and Sebastiaan De Hoogh. 2010. Improved Primitives for Secure Multiparty Integer Computation. In *SCN'10*. Springer.

[23] Octavian Catrina and Amitabh Saxena. 2010. Secure Computation with Fixed-point Numbers. In *FC'10*. Springer.

[24] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. 2017. *EzPC: Programmable, Efficient, and Scalable Secure Two-Party Computation*. Technical Report. IACR Cryptology ePrint Archive 2017/1109.

[25] Confidential Source. [n. d.]. Credit Approval Data Set . http://archive.ics.uci.edu/ml/datasets/credit+approval.

[26] ACC Coolen. 1998. A beginner's guide to the mathematics of neural networks. In *Concepts for Neural Networks*. Springer, 13–70.

[27] Daniel Demmler, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. 2015. Automated Synthesis of Optimized Circuits for Secure Computation. In *CCS'15*. ACM.

[28] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*.

[29] Nathan Dowlin, Ran Gilad-Bachrach, 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 ICML*, Vol. 48. 201–210.

[30] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. 1998. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

[31] Khaled El Emam. 2011. Methods for the de-identification of electronic health records for genomic research. *Genome medicine* 3, 4 (2011), 25.

[32] EncryptoGropu. 2015. ABY – A Framework for Efficient Mixed-protocol Secure Two-party Computation. https://github.com/encryptogroup/ABY.

[33] EncryptoGroup. 2018. C++ OT extension implementation. https://github.com/encryptogroup/OTExtension.

[34] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. 2009. Privacy-Preserving Face Recognition. In *PETS*, Ian Goldberg and MikhailJ. Atallah (Eds.). LNCS, Vol. 5672. Springer, 235–253.

[35] Martin Franz. 2011. *Secure Computations on Non-integer Values*. Ph.D. Dissertation. Technische Universität Darmstadt.

[36] Martin Franz, Björn Deiseroth, Kay Hamacher, Somesh Jha, Stefan Katzenbeisser, and Heike Schröder. 2010. Secure computations on non-integer values. In *WIFS'10*. IEEE.

[37] Martin Franz, Björn Deiseroth, Kay Hamacher, Somesh Jha, Stefan Katzenbeisser, and Heike Schröder. 2011. Towards Secure Bioinformatics Services (Short Paper). In *FC'11*. Springer.

[38] Google. 2018. Cloud Machine Learning Engine. https://cloud.google.com/ml-engine/.

[39] Thore Graepel, Kristin Lauter, and Michael Naehrig. 2012. ML confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*. Springer, 1–21.

[40] Wilko Henecka, Ahmad-Reza Sadeghi, Thomas Schneider, Immo Wehrenberg, et al. 2010. TASTY: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 451–462.

[41] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster Secure Two-party Computation Using Garbled Circuits. In *USENIX Security*. USENIX.

[42] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO 2003*, Dan Boneh (Ed.). LNCS, Vol. 2729. Springer, 145–161.

[43] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. Gazelle: A Low Latency Framework for Secure Neural Network Inference. *CoRR* abs/1801.05507 (2018). arXiv:1801.05507 http://arxiv.org/abs/1801.05507

[44] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. 2014. Scaling Private Set Intersection to Billion-Element Sets. In *International Conference on Financial Cryptography and Data Security*. Springer, 195–215.

[45] Liina Kamm and Jan Willemson. 2015. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security* 14, 6 (2015), 531–548.

[46] Ágnes Kiss and Thomas Schneider. 2016. Valiant's universal circuit is practical. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer Berlin Heidelberg, 699–728.

[47] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. 2009. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *CANS 2009*, JuanA. Garay, Atsuko Miyaji, and Akira Otsuka (Eds.). LNCS, Vol. 5888. Springer, 1–20.

[48] Vladimir Kolesnikov and Thomas Schneider. 2008. A practical universal circuit construction and secure evaluation of private functions. In *International Conference on Financial Cryptography and Data Security*. Springer, 83–97.

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[50] Yann LeCun, Corinna Cortes, and Christopher Burges. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[51] Yehuda Lindell and Benny Pinkas. 2009. Secure Multiparty Computation for Privacy-Preserving Data Mining. *Journal of Privacy and Confidentiality* 1, 1 (2009).

[52] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 619–631. https://doi.org/10.1145/3133956.3134056

[53] Viktor Mayer-Schönberger and Kenneth Cukier. 2013. *Big Data: A Revolution That Will Transform How We Live, Work and Think*. John Murray Publishers, UK.

[54] Microsoft. 2018. Microsoft Machine Learning Services. https://azure.microsoft.com/en-us/services/machine-learning-services/.

[55] P. Mohassel and Y. Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. 19–38. https://doi.org/10.1109/SP.2017.12

[56] Moni Naor and Benny Pinkas. 2005. Computationally Secure Oblivious Transfer. *Journal of Cryptology* 18, 1 (2005), 1–35.

[57] Manas Pathak, Shantanu Rane, Wei Sun, and Bhiksha Raj. 2011. Privacy preserving probabilistic inference with Hidden Markov Models. In *ICASSP'11*. IEEE.

[58] Manas A Pathak and Bhiksha Raj. 2013. Privacy-Preserving Speaker Verification and Identification Using Gaussian Mixture Models. *IEEE Transactions on Audio, Speech, and Language Processing* 21, 2 (2013), 397–406.

[59] Manas A Pathak, Bhiksha Raj, SD Rane, and Paris Smaragdis. 2013. Privacy-Preserving Speech Processing: Cryptographic and String-Matching Frameworks Show Promise. *IEEE Signal Processing Magazine* 30, 2 (2013), 62–74.

[60] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection using Permutation-based Hashing. In *24th USENIX Security Symposium (USENIX Security 15)*. 515–530.

[61] Huseyin Polat, Wenliang Du, Sahin Renckes, and Yusuf Oysal. 2010. Private predictions on hidden Markov models. *Artificial Intelligence Review* 34, 1 (2010), 53–72.
[62] José Portêlo, Bhiksha Raj, and Isabel Trancoso. 2015. Logsum Using Garbled Circuits. *PLoS ONE* 10, 3 (2015), 1–16.
[63] Bruce Porter and Ross Quinlan. 1992. Audiology (Standardized) Data Set. http://archive.ics.uci.edu/ml/datasets/audiology+(standardized).
[64] Lawrence R Rabiner. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. IEEE* (1989).
[65] Jorge L. Reyes-Ortiz, Davide Anguita, Alessandro Ghio, Luca Oneto, and Xavier Parra. 2012. Human Activity Recognition Using Smartphones Data Set. http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones.
[66] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *13. ACM Asia Conference on Information, Computer and Communications Security (ASIACCS'18)*. ACM. To appear. Preliminary version: http://ia.cr/2017/1164.
[67] Irina Rish. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Vol. 3. IBM New York, 41–46.
[68] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable Provably-secure Deep Learning. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*. ACM, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3195970.3196023
[69] Stuart J. Russell and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
[70] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. 2009. Efficient Privacy-Preserving Face Recognition. In *ICISC*.
[71] John Shawe-Taylor and Nello Cristianini. 2004. *Kernel methods for pattern analysis*. Cambridge university press.
[72] Paris Smaragdis and Madhusudana Shashanka. 2007. A framework for secure speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on* 15, 4 (2007), 1404–1413.
[73] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. 2015. TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits. In *IEEE SP*. IEEE.
[74] V. Rajkovic et al. 1997. Nursery Data Set. https://archive.ics.uci.edu/ml/datasets/Nursery.
[75] Jaideep Vaidya and Chris Clifton. 2004. Privacy preserving naive bayes classifier for vertically partitioned data. In *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 522–526.
[76] Jaideep Vaidya, Murat Kantarcıoğlu, and Chris Clifton. 2008. Privacy-preserving naive bayes classification. *The VLDB Journal—The International Journal on Very Large Data Bases* 17, 4 (2008), 879–898.
[77] WIlliam H. Wolberg. 1992. Breast Cancer Wisconsin (Original) Data Set. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original).
[78] William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian. 1995. Breast Cancer Wisconsin (Diagnostic) Data Set. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).
[79] Rebecca Wright and Zhiqiang Yang. 2004. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 713–718.
[80] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 162–167.
[81] Hwanjo Yu, Xiaoqian Jiang, and Jaideep Vaidya. 2006. Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. In *Proceedings of the 2006 ACM symposium on Applied computing*. ACM, 603–610.
[82] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two Halves Make a Whole. In *EUROCRYPT*. Springer.
[83] Harry Zhang. 2004. The Optimality of Naive Bayes.. In *FLAIRS Conference*, Valerie Barr and Zdravko Markov (Eds.). AAAI Press. http://www.cs.unb.ca/profs/hzhang/publications/FLAIRS04ZhangH.pdf
[84] Jan Henrik Ziegeldorf, Oscar García Morchon, and Klaus Wehrle. 2014. Privacy in the Internet of Things: threats and challenges. *Security and Communication Networks* 7, 12 (2014), 2728–2742.
[85] Jan Henrik Ziegeldorf, Jan Metzke, Martin Henze, and Klaus Wehrle. 2015. Choose wisely: a comparison of secure two-party computation frameworks. In *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 198–205.
[86] Jan Henrik Ziegeldorf, Jan Metzke, Jan Rüth, Martin Henze, and Klaus Wehrle. 2017. Privacy-Preserving HMM Forward Computation. In *Proceedings of the Seventh ACM Conference on Data and Application Security and Privacy (CODASPY '17)*. ACM, New York, NY, USA, 83–94. https://doi.org/10.1145/3029806.3029816
[87] Jan Henrik Ziegeldorf, Nicolai Viol, Martin Henze, and Klaus Wehrle. 2014. POSTER: Privacy-preserving Indoor Localization. *WiSec'14* (2014).

A DETAILED PROTOCOLS FOR SECURE BUILDING BLOCKS

In this section, we provide the detailed protocols for our building blocks as presented in Sec. 4.2 and summarized in Tab. 7.

A.1 Max and Argmax

Prot. 5 (presented in Sec. 4.2.2) provides the details of the MaxArgmax protocol which is used in all our classifiers, i.e., Hyperplane, Ann, Ann, and Viterbi. $\mathcal{GC}_C(\langle x\rangle, \tilde{y}, z)$ denotes the secure evaluation of a Boolean circuit C on secret-shared input x (U and S each input their individual share), garbled input y (an input already held in garbled form by the evaluator of the circuit), and clear text input z (held by U or S in clear) using Yao's generic GC protocol. We only note the three steps, i) converting to GC using the garbled addition circuit C_{Add}, ii) computing the garbled argmax circuit C_{Argmax} and iii) converting to additive shares using garbled subtraction circuit C_{Sub}, separately for the sake of clarity – they are implemented as one monolithic circuit for greater efficiency.

Protocol 5 Secure Argmax protocol based on GC and ASS.

Input: Additive sharing of vector $\langle\vec{x}\rangle = (\langle x_1\rangle, \ldots, \langle x_n\rangle)$
Output: Additive sharing $\langle x^*\rangle$, $\langle i^*\rangle$ of
$x^* = \max_{i=1\ldots n} x_i$ and $i^* = \arg\max_{i=1\ldots n} x_i$

$U \Leftrightarrow S:\quad \tilde{x}_1, \ldots, \tilde{x}_n \leftarrow \mathcal{GC}_{C_{Add}}(\langle x_1\rangle, .., \langle x_n\rangle)$
$U \Leftrightarrow S:\quad \tilde{x}, \tilde{i}^* \leftarrow \mathcal{GC}_{C_{Argmax}}(\tilde{x}_1, \ldots, \tilde{x}_n)$
$U \Leftrightarrow S:\quad \langle x^*\rangle, \langle i^*\rangle \leftarrow \mathcal{GC}_{C_{Sub}}(\tilde{x}^*, \tilde{i}^*)$

A.2 Scalar Products

Prot. 6 (presented in Sec. 4.2.3) provides the details of the ScalarProd protocol for securely computing scalar products which is heavily used in Hyperplane and Ann to compute weighted sums. $\odot$ denotes multiplication on additive shares which is implemented using precomputed Multiplication Tripless (MTs) [12, 28]. The secure Rescale protocol on additive shares has been adopted from [86].

Protocol 6 Secure ScalarProd protocol based on ASS.

Input: Additive shares $\langle\vec{x}\rangle$ and $\langle\vec{w}\rangle$ of two equal sized vectors $\vec{x}$ and $\vec{w}$
Output: Additive shares $\langle z\rangle$ of the inner product $z = \vec{x}\cdot\vec{w}$

$U \Leftrightarrow S:\quad \langle z_i\rangle = \langle x_i\rangle \odot \langle w_i\rangle \qquad \forall i = 1\ldots n$
$U, S:\quad \langle z\rangle = \sum_{i=1}^{n} \langle z_i\rangle$
$U \Leftrightarrow S:\quad \langle z\rangle \leftarrow \text{Rescale}(\langle z\rangle)$

A.3 Polynomial Approximation of Arbitrary Functions

Prot. 7 (presented in Sec. 4.2.4) provides the details of the PolyFuncApprox protocol. As for MaxArgmax, we only note conversion steps between GC and ASS separately for the sake of clarity while C_{Add}, $C_{Selection}$, and C_{Sub} are implemented as one monolithic circuit in practice. The selected approximation polynomial

Protocol	Description
$x' \leftarrow$ F2I(x, l, s)	Conversion from reals to integers with s bit precision and maximum bitlength l (inverse: I2F)
$\hat{p}' \leftarrow$ F2LI(p, l, s)	Conversion from probabilities to integers in logspace with s bit precision s and bitlength l (inverse: LI2F)
$\langle x^* \rangle, \langle i^* \rangle \leftarrow$ ARGMAX$(\langle \vec{x} \rangle)$	Max and argmax on secret-shared input vector with secret output
$\langle z \rangle \leftarrow$ SCALARPROD$(\langle \vec{x} \rangle, \langle \vec{w} \rangle)$	Scalar product on two equal-sized secret-shared vectors $\vec{x}$, $\vec{w}$ with secret outputs
$\langle f'(x_i) \rangle \leftarrow$ POLYFUNCAPPROX$(\langle \mathcal{P}_f \rangle, \langle x \rangle)$	Polynomial approximation of possibly secret function f at secret point with secret output
$\langle f(x_i) \rangle \leftarrow$ OTFUNCAPPROX$(\langle f(X) \rangle, x)$	OT-based approximation of possibly secret function f at point x known by U with secret output
$\langle \mathcal{N}_{\mu,\sigma}(x) \rangle \leftarrow$ GAUSSIAN$(\langle x \rangle, \langle \mu \rangle, \langle \sigma \rangle)$	Secure evaluation of Gaussian $\mathcal{N}$ with secret parameters μ, σ at secret point x with secret output.
$S^* \leftarrow$ BACKTRACK$(\langle M \rangle, s_T^*)$	Backtracking through secret state matrix M from state s_T^* known by U or S with output to U or S

Table 7: Summary of the secure building blocks of SHIELD. The protocols are provided in full detail in Appendix A.

$g_i(x)$ is evaluated using the EVALPOLY subprotocol. EVALPOLY requires $\lceil \log_2(d) \rceil + 1$ rounds of parallel multiplications and rescaling: In round j, the terms $\langle x^{2^j} \rangle, \ldots, \langle x^{2^{j-1}+1} \rangle$ and $\langle a_{2^{j-1}} x^{2^{j-1}} \rangle, \ldots, \langle a_{2^{j-2}+1} x^{2^{j-2}+1} \rangle$. Finally, all shares are added up locally.

Protocol 7 Secure POLYFUNCAPPROX protocol for the evaluation of arbitrary functions based on GC and ASS.

Input: Shared evaluation point $\langle x \rangle$ and approximation parameters $\langle \mathcal{P} \rangle = (\langle a_{10} \rangle, \ldots, \langle a_{kd} \rangle, \langle r_1 \rangle, \ldots, \langle r_k \rangle)$
Output: Approximated result $\langle g_i(x) \rangle$ with $r_i \leq x < r_i$

Parameter selection:

$U \Leftrightarrow S: \quad \tilde{x} \leftarrow \mathcal{GC}_{C_{Add}}(\langle x \rangle)$
$U \Leftrightarrow S: \quad \tilde{\mathcal{P}} \leftarrow \mathcal{GC}_{C_{Add}}(\langle \mathcal{P} \rangle)$
$U \Leftrightarrow S: \quad \tilde{a}_{id}, \ldots, \tilde{a}_{i0} \leftarrow \mathcal{GC}_{C_{Selection}}(\tilde{x}, \tilde{\mathcal{P}})$
$U \Leftrightarrow S: \quad \langle g_i \rangle = (\langle a_{id} \rangle, \ldots, \langle a_{i0} \rangle) \leftarrow \mathcal{GC}_{C_{Sub}}(\tilde{a}_{id}, \ldots, \tilde{a}_{i0})$
$U \Leftrightarrow S: \quad \langle g_i(x) \rangle \leftarrow$ EVALPOLY$(\langle g_i \rangle, \langle x \rangle)$

Subprotocol EVALPOLY: $\forall i = 1, \ldots, \lceil log_2(d) \rceil + 1$

$U \Leftrightarrow S: \quad j = 1 : \langle x^2 \rangle \leftarrow \langle x \rangle \odot \langle x \rangle, \langle a_1 x \rangle \leftarrow \langle a_1 \rangle \odot \langle x \rangle$
$j = 2 : \langle x^4 \rangle \leftarrow \langle x^2 \rangle \odot \langle x^2 \rangle, \langle x^3 \rangle \leftarrow \langle x^2 \rangle \odot \langle x \rangle,$
$\langle a_2 x^2 \rangle \leftarrow \langle a_2 \rangle \odot \langle x^2 \rangle$
$j : \langle x^{2^j} \rangle, \ldots, \langle x^{2^{j-1}+1} \rangle, \langle a_{i2^{j-1}} x^{2^{j-1}} \rangle, \ldots, \langle a_{i2^{j-2}+1} x^{2^{j-2}+1} \rangle$
$j = \lceil \log_2(d) \rceil + 1 : \langle a_{id} x^d \rangle, \langle a_{id-1} x^{d-1} \rangle, \ldots$
$U, S: \quad \langle g_i(x) \rangle = \sum_{j=0}^{d} \langle a_{ij} x^j \rangle$

A.4 OT-based Evaluation of Discrete Functions

Prot. 8 (presented in Sec. 4.2.4) provides the details of the OTFUNCAPPROX protocol. We adopted this protocol from [86] and improved it such that the values $f(x_1), \ldots, f(x_m)$ can also be secret-shared as opposed to the protocol from [86] where they need to be known in clear by the service S.

A.5 Evaluating Gaussians

Prot. 9 (presented in Sec. 4.2.4) provides the details of the GAUSSIAN protocol for securely evaluating secret Gaussians $\mathcal{N}_{\mu,\sigma}$ at secret evaluation points x. Since we can parallelize multiplications and rescaling, the protocol requires only two rounds of communication.

Protocol 8 Secure OTFUNCAPPROX protocol for evaluating a probability mass function based on ASS and OT adapted from [86].

Input: Additive sharing $\langle f(X) \rangle = \langle f(x_1) \rangle, \ldots, \langle f(x_m) \rangle$, only U has x_i
Output: Additive sharing $\langle f(x_i) \rangle$

$S: \quad \langle f'(X) \rangle = \langle f(x_1) + r_S \rangle, \ldots, \langle fp(x_m) + r_S \rangle$ with $r_S \in_R \mathbb{Z}_{2^l}$
$U \Leftrightarrow S: \quad \langle f(x_i) \rangle_U \leftarrow 1\text{-m-OT}_l^1(x_i, \langle f'(X) \rangle)$
$S: \quad \langle f(x_i) \rangle_S = -r_S$

Protocol 9 Secure GAUSSIAN protocol for evaluating a Gaussian distribution based on ASS.

Input: Additive shares $\langle x \rangle, \langle \mu \rangle, \langle \log(\sigma) \rangle, \langle 1/-2\sigma^2 \rangle$
Output: Additive shares $\langle \hat{p}(x)' \rangle = \langle \log(\mathcal{N}_{\mu,\sigma}(x)) \rangle$

$U \Leftrightarrow S: \quad \langle \hat{p}(x)' \rangle = \langle x \rangle \oplus \langle -\mu_i \rangle$
$U \Leftrightarrow S: \quad \langle \hat{p}(x)' \rangle =$ RESCALE$(\langle \hat{p}(x)' \rangle \odot \langle \hat{p}(x)' \rangle)$
$U \Leftrightarrow S: \quad \langle \hat{p}(x)' \rangle =$ RESCALE$(\langle \hat{p}(x)' \rangle \odot \langle 1/-2\sigma^2 \rangle)$
$U, S: \quad \langle \hat{p}(x)' \rangle = \langle -\hat{\sigma} \rangle) \oplus \langle \hat{p}(x)' \rangle$

A.6 Backtracking

Prot. 10 (presented in Sec. 4.2.5) provides the details of the BACKTRACK protocol for secure backtracking through a secret-shared dynamic-programming matrix. It is used in VITERBI to compute the optimal state sequence for a given observation sequence. Adapting the protocol such that S instead of U learns the state sequence is straightforward by switching the roles of the two parties.

Protocol 10 Secure BACKTRACK protocol for backtracking through a DP matrix based on OT and ASS.

Input: Additive shares of DP matrix $\langle M \rangle$ and final state $\langle s_T^* \rangle$
Output: U obtains optimal state sequence $S^* = s_1^*, \ldots, s_T^*$

$U \Leftrightarrow S: \quad s_T^* \leftarrow$ RECOMBINE$(\langle s_t^* \rangle)$

Backtracking: For $T \geq t \geq 2$

$U \Leftrightarrow S: \quad \langle s_{t-1}^* \rangle \leftarrow 1\text{-}N\text{-OT}_l(s_t^*, (\langle M_{1t} \rangle_S, \ldots, \langle M_{Nt} \rangle_S))$
$U: \quad s_{t-1}^* \leftarrow$ RECOMBINE$(\langle s_{t-1}^* \rangle)$

B SECURITY DISCUSSION

We show that our classifiers are secure in the semi-honest adversary model. For the security proofs of the basic STC techniques underlying our approaches we refer to [8, 56] for OT, [13, 28] for ASS, and [14, 80] for GC.

We begin by showing that neither party learns anything in our proposed building blocks protocols, i.e., neither from the inputs, nor the outputs, nor any intermediate values. We then invoke Canetti's modular sequential composition theorem [21] to argue that our classifier designs built on top of these primitives are secure.

B.1 Security of the Building Blocks

We discuss security individually for each of the building blocks proposed in Sec. 4.2 and presented in detail in Appendix A.

B.1.1 Security of MaxArgmax. All steps of MaxArgmax are realized in one monolithic GC – we emphasize that we differentiate the three steps in our protocol description only for reasons of clarity but implement them in one single GC which yields better performance. Consisting of only one GC, security for these steps follows directly from the security of GCs. The inputs and outputs are all additively shared over both parties. Since individual shares are perfectly random, they reveal no information to either party.

B.1.2 Security of ScalarProd. Security of ScalarProd (Prot. 6, Sec. 4.2.3) follows from the security of addition and multiplication over additive shares [28].

B.1.3 Security of PolyFuncApprox. For PolyFuncApprox (Prot. 7) we show that neither party learns anything about the inputs $\langle x \rangle$ and $\langle \mathcal{P} \rangle$ and the output $\langle g_i(x) \rangle$. We first note that all inputs are given as additive shares and a single share is perfectly random and does not reveal any information to its holder. The first protocol steps, involving i) input conversion, ii) the selection of approximation parameters, and iii) the conversion of outputs, are realized in one monolithic GC. As for MaxArgmax before, we only differentiate these three steps in our protocol description for reasons of clarity but implement them in one single GC which yields better performance. Consisting of only one GC, security for these steps follows directly from the security of GCs. The output of these steps $\langle g_i \rangle$ is additively shared over both parties which reveals no information to either party holding only a single share of each output since additive sharing implements perfect blinding over $\mathbb{Z}_{2^l}$. It is also important to note that the structure of the circuit is independent of all parameters except for the public parameter k (the number of approximation intervals in this context), therefore leaking no sensitive information. In the penultimate step, we securely evaluate the shared polynomial $\langle g_i \rangle$ on the shared evaluation point $\langle x \rangle$. Since this step involves only multiplication and rescaling, security follows from the security of ASS and the security of Rescale as discussed in [86]. All outputs are again additively shared and reveal no information to either party. The last step involves an addition operation over additive shares which is executed locally and has no security implications in the semi-honest model. Finally, the output $\langle g_i(x) \rangle$ is obtained by the two parties in shared form, where a single share is indistinguishable from a random value and reveals no information. In summary, security of PolyFuncApprox depends on the security of Yao's GCs and the Rescale protocol. As Rescale offers only statistical security against a semi-honest S, PolyFuncApprox as well offers only statistical security.

B.1.4 Security of OtFuncApprox. Security of the OtFuncApprox protocol (Prot. 8, Sec. 4.2.4) follows directly from the security of OT and ASS.

B.1.5 Security of Gaussian. Gaussian (Prot. 9) only composes secure additions, multiplications, and the Rescale protocol from [86]. We thus conclude that it is secure in the semi-honest model. It is perfectly secure against U and offers statistical security against S due to Rescale. By switching the roles of U and S in Rescale, we can flip these guarantees if desired.

B.1.6 Security of Backtrack. Backtrack (Prot. 10) inherits all security guarantees directly from the utilized OT protocol which ensures that U is only able to learn the optimal state sequence S^* and nothing else about the DP matrix M while S learns nothing at all. Note that we can easily switch the roles if S should learn S^* instead of U.

B.2 Security of the Classifier Designs

The security argument is the same for all our classifier designs, i.e., Hyperplane, Ann, NaiveBayes, and Viterbi. We argue that U learns nothing about the involved classification model M (private input of S), and, vice versa, S learns nothing about the feature vector $\vec{x}$ (U's private input), except of course for what is implied in the final result that is learned in clear by one or both parties. This proposition holds since any interaction between U and S in all our secure classifiers happens only through one or multiple of the building blocks discussed above. As we have showed in the previous section, all of these primitives are secure in the semi-honest model and return their output in the form of random additive shares such that their output does not reveal anything to either party. In other words, their use reveals no information about the inputs or any intermediate values. This allows us to compose them and the composition is then secure according to the modular sequential composition theorem for semi-honest protocols [21]. All other steps in the secure classification protocols are local operations that have no security implications in the semi-honest model. Finally, one or both parties learn the output by recombining the shared result which is of course as intended.

It is important to note that the utilized STC techniques protect the inputs (i.e., the models and feature vectors) but not the structure of the evaluated classification function. In particular, this implies that S learns the length of the feature vector $\vec{x}$ while U learns the dimension of the models, e.g., the total number of possible classes, the number of layers and neurons in an ANN, or the number of states and possible emissions in an HMM. We emphasize that this is fully within the security model defined in Sec. 2.1. If desired this can be prevented in all our designs by padding inputs with dummy features or observations and models with dummy weights, neurons, states, and so forth but this inevitably increases processing and communication overheads. Another approach are universal circuits that also hide the function that is being evaluated [46, 48] also referred to as Private Function Evaluation (PFE). PFE causes orders of magnitude higher overheads than the secure evaluation of public functions. We argue that the costs of PFE are not justified in our application context since our classification algorithms are publicly known and do not require protection.

	Preprocessing		Outsourcing	
	Processing	Communication	Processing	Communication
Hyperplane	-	-	0.27 ms	4.49 kB
Ann	-	-	0.40 ms	6.68 kB
NaiveBayes	2.57 ms	0.13 MB	1.24 µs	0.10 kB
Viterbi	1.02 s	113.99 MB	786.47 ms	18.99 MB

Table 8: Runtime and communication for S for outsourcing the largest considered problem instances.

C EVALUATION OF OUTSOURCING FOR THE SERVICE PROVIDER

Tab. 8 summarizes the costs S for outsourcing our different classifiers to an untrusted computation cloud. The results complement our evaluation of the user's side as discussed in Sec. 9.1.

For Hyperplane, Ann, and NaiveBayes outsourcing is clearly highly efficient and feasible even if S is not running a powerful server-grade machine. For Viterbi, runtimes for preprocessing and outsourcing of are clearly feasible while the communication overhead, especially in the preprocessing phase, might prove challenging when S is connected via networks with constrained bandwidth. However, we considered the largest HMM and observation sequence from our evaluation – overheads for smaller models range only in the order of kB to a few MB.