# Privacy-Preserving Deep Learning with SPDZ

**Shreya Sharma**[1], **Chaoping Xing**[2], **Yang Liu**[3]
[1]Department of Electronics Engineering, Indian Institute of Technology (BHU) Varanasi, India
Email: shreyas.cd.ece17@iitbhu.ac.in
[2]School of Electronics, Information & Electrical Engineering, Shanghai Jiao Tong University, China
School of Physical & Mathematical Sciences, Nanyang Technological University, Singapore
Email: xingcp@ntu.edu.sg
[3] Webank, Shenzhen, China
Email: yangliu@webank.com

## Abstract

Neural Networks (NN) are powerful tools for supervised machine learning. However, extensive data collection from different sources for accurate training risks privacy. Most privacy-preserving solutions for secure Machine Learning either don't guarantee active security for a dishonest majority or do so for linear models only. In this work, we explore the practicality of Neural Network training and evaluation using SPDZ, a family of secret-sharing based MPC protocols that provide active security against a dishonest majority. We investigate different intricacies of Machine Learning suitable for the setting, benchmark the models in fields, and extrapolate various results of previous benchmarks to explore promising improvements in rings. A single inference by our implementation takes 0.11 seconds for FNN and 0.16 seconds for CNN in the online phase.

## Introduction

Machine learning as a service (MLaaS) is a cloud-based service paradigm that enables clients to train machine learning models and perform online inference on the already trained ones. While this has clear benefits, it puts the privacy of the clients at risk because the clients' input-data submitted to the cloud service may contain sensitive private information, such as, medical records, financial data, or location; which in-turn hinders the applicability of Deep Learning models. In order to protect inference input data, one solution can be to make the model available for public use. However, this can lead to leakage of the data on which the model was trained. Moreover, the trained model itself may require confidentiality because it constitutes a competitive or monetary advantage for the parties that performed the training. Thus arises the need for models that can train and predict in such a way that the server learns nothing about clients' input, and clients learn nothing about the model except the prediction results.

This work focuses on providing a solution for the above problem by enabling $M$ data owners to train a deep learning model on their joint data using $N$ servers. Secure Multiparty Computation (MPC) allows this to be achieved in the client-server setting by letting the data owners and the model owners secret-share their input and then a set of servers run the computation over these shares. More specifically, these M parties send secret shares of their input data to the N servers. An interactive protocol to train a neural network over the received data is run by the servers collectively to produce a trained model that can be used for inference. The trained model can be kept hidden from any single party and retained as secret shares between the servers or reconstructed to obtain the model in the clear. Predictions on any new input can be made using the trained model even if the model is retained as secret shares between the N servers. Both M and N can be arbitrarily chosen; for simplicity, we focus on the case where N=2. We would like to stress on how prevalent a two-party setting is on the Internet in the form of client-server models, and since our work supports a dishonest majority(number of corrupt parties $\geq$ N/2), it stands relevant to the case where one party trains the model while the other party performs inference on it.

However, theoretical MPC protocols over low-level circuits often do not scale efficiently to real-world data for complex tasks such as neural network training and lead to highly inefficient implementation. One reason for this fact is that Neural Network evaluation involves performing floating-point arithmetic, whereas secret-sharing based MPC protocols are better suited for modular integer-based arithmetic. Moreover, Neural Networks include non-linear activation functions which are typically evaluated by expensively converting to different sharing and the use of Garbled Circuits (Rotaru and Wood 2019) or by using a polynomial approximation of the function which leads to loss of accuracy, as in SecureML (Mohassel and Zhang 2017).

Implementation using MPC protocols with active security against a dishonest majority tend to be much more complicated. These protocols are typically field-based, i.e. they support arithmetic modulo $p$, where $p$ is a large prime. This restriction makes it much more difficult to convert between $\mathbb{Z}_p$ sharings and binary sharings, resulting in operations such as truncation (which is necessary for operating on floating-point numbers in arithmetic circuits), and comparison being expensive in fields. Thus, such protocols are rendered incompetent for this particular application.

### Our Contribution

- Our work aims to calibrate the practicality of Deep Learning with malicious security in both fields and rings. It is the first to give experimental results for both training and inference of NN in the field-based actively secure setting

involving a dishonest majority.

- We present Fully Connected Neural Network (FNN) and Convolutional Neural Network (CNN) models that best suit the SPDZ framework, involving activation functions that avoid the switch to garbled circuits for division (as necessary in case of many functions such as softmax), and using initialization techniques such as Xavier Initialization in order to keep the range of values obtained in check for accurate results. Further, we investigate the impact of all such changes on accuracy of the model.
- We also provide a sound and promising extrapolation of our experimental results in fields to a ring-based actively-secure setting.

Our techniques are powerful enough to train CNNs that produce an accuracy of 99% and FNNs that produce an accuracy greater than 97% on the MNIST dataset.

### Related Work

The work on secure Neural Networks evaluation started with Homomorphic Encryption based techniques (Barni, Orlandi, and Piva 2006; Orlandi, Piva, and Barni 2007). Microsoft proposed CryptoNets (Gilad-Bachrach et al. 2016) based on Leveled Homomorphic Encryption (LHE), which assumes that training has been done on plain-text and the model is known to one party who evaluates it on encrypted inputs of the other party. Apart from the high computational overhead, it uses the polynomial approximation of non-linear activation functions, and as LHE restricts the degree of the polynomial used, the model was not accurate enough.

SecureML (Mohassel and Zhang 2017) focuses on private training and inference of various ML models like Linear regression, Logistic Regression and Neural Networks. However, this ring-based work does not guarantee active security, and for the evaluation of certain non-linear activation functions like softmax, it proposes an expensive switching between arithmetic secret shares and Garbled circuits. This work manages to achieve an accuracy of 93.7% through a Fully-Connected Neural Network with two hidden layers while incurring around a 1% accuracy loss as compared to the plain-text model.

MiniONN (Liu et al. 2017) is the first work to talk about building on top of existing Neural Network models to obtain security in a passively secure model. It equipped a combination of Garbled Circuits and Homomorphic encryption to do so in a 2-party setting. The use of Yao's Garbled Circuits for the computation renders the scalability of this work to more than two parties very difficult. Chameleon Framework (Riaz et al. 2018) for secure Neural Network evaluation relies on a third-party trusted dealer, but the existence of such a trusted third-party seems quite impractical. It presents results based on fixed-point arithmetic and suggests extension of their techniques to floating-point arithmetic for better coherence with the plain-text version of the model. Gazelle (Juvekar, Vaikuntanathan, and Chandrakasan 2018) presents a field-based hybrid combination of *packed additively homomorphic encryption (PAHE)* along with Garbled circuits. It achieves improvement by a factor of 20 and 30 with respect to MiniONN and Chameleon, assuming passive corruptions.

DeepSecure (Rouhani, Riazi, and Koushanfar 2018) is a work entirely based on Garbled Circuits. They implement a library that supports fixed-point arithmetic, introduces optimized Garbled Circuit protocols directed towards secure Neural Network evaluation. They show results for security against a passive adversary.

SecureNN (Wagh, Gupta, and Chandran 2018) is the state-of-the-art protocol for secure training and inference of Neural Networks. Its efficiency partly results from eliminating Oblivious Transfer from the preprocessing phase and involving a third party that acts as a source of randomness on top of participating in real computation of the protocol. Since the parties are likely to collude, the security provided for the case of honest majority restricts applicability to practical scenarios.

ABY3 (Mohassel and Rindal 2018) presents three-party based protocols that enable faster conversion between Arithmetic, Boolean and Yao shares. This paper is ring-based but shows experimental numbers for passive settings only. They also show how to convert their protocols into actively secure in case of honest majority.

One of the more recent works that talks about secure evaluation of NN is (Barak et al. 2019), which introduces a Machine Learning technique called *quantization* to privacy-preserving deep learning. It is the first work that provides results for actively secure NN inferencing with honest majority. This is achieved in fields using MP-SPDZ framework (MPS N1 Analytics). It also explores a ring-based approach for three-party MPC setting and manages to achieve passive security in case of honest majority.

Machine Learning using SPDZ in fields has been explored to some extent in (Zheng et al. 2019). But the work only focuses on linear models. (Chen, Pastro, and Raykova 2019) is another work on actively secure Machine Learning that shows some particular machine learning models trained using SPDZ can match the latency of (Mohassel and Zhang 2017).

Apart from the primarily MPC-based solutions presented so far, an orthogonal path considers NN evaluation with some limitations in Trusted Execution Environment like Intel SGX. For instance, Chiron (Hunt et al. 2018) which is vulnerable to covert and side-channel attacks and Slalom (Tramèr and Boneh 2019) which only allows secure inferences.

## Preliminaries

**SPDZ** is a family of MPC schemes which work on additive-sharing with information theoretic MACs, providing malicious security for dishonest majority. More specifically, each Party $P_i$ holds an additive share, $\alpha_i \in \mathbb{F}_p$ of the fixed MAC key $\alpha = \alpha_1 + ... + \alpha_n$. A value is said to be $\langle \cdot \rangle$ shared if each Party $P_i$ has a tuple $(a_i, \gamma(a)_i)$ where $a_i$ is an additive share of $a$ such that $a = a_1 + ...a_n$ and $\gamma(a)_i$ is an additive share of $\gamma(a)$ such that $\gamma(a) = \alpha a$ and $\gamma(a) = \gamma(a)_1 + .. + \gamma(a)_n$. The online-phase of the SPDZ protocol is given in Figure 1.

Figure 1: $\Pi^{\text{SPDZ}}_{\text{Online}}$ - SPDZ Online Protocol

The set $P$ is the complete set of parties.
**Initialise:** The parties call preprocessing functionality to obtain enough multiplication triples $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ and input mask values $(r_j, \langle r_j\rangle)$ according to the function being evaluated. If the functionality aborts, the parties output $\perp$ and abort.
**Input:** To input $x_j$, party $P_j \in P$ takes a mask value $(r_j, \langle r_j\rangle)$, then:

1. Broadcasts $\Delta \leftarrow x_j - r_j$
2. Parties compute $\langle x_j\rangle \leftarrow \langle r_j\rangle + \Delta$

**Add:** On input $(\langle x\rangle, \langle y\rangle)$, locally compute $\langle x+y\rangle \leftarrow \langle x\rangle + \langle y\rangle$
**Multiply:** On input $(\langle x\rangle, \langle y\rangle)$, the parties:

1. Take a multiplication triple $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$, compute $\langle \epsilon\rangle \leftarrow \langle x\rangle - \langle a\rangle$ and $\langle \rho\rangle \leftarrow \langle y\rangle - \langle b\rangle$ and partially open them to obtain $\epsilon$ and $\rho$.
   Partially opening a share involves each party sending its own share of the value to every other party and computing the sum of all the shares available to it, while the corresponding MAC value $\gamma(x_i)$ is kept secret.
2. Set $\langle z\rangle \leftarrow \epsilon\cdot\rho + \epsilon\cdot\langle a\rangle + \rho\cdot\langle b\rangle + \langle c\rangle$.

**Output:** To output a share $\langle x\rangle$:

1. Check all partially opened values since the last batched MAC-check, as follows:
   - The parties have ids $id_1$, ... $id_k$ corresponding to opened values $x_1$, .. $x_k$
   - Parties agree on a random vector $\mathbf{r} \leftarrow \mathcal{F}_{\text{Rand}}\left(\mathbb{F}_q^k\right)$
   - Party $P_i$ computes $c \leftarrow \sum_{j=1}^{k} r_j \cdot x_j$ and $\gamma(c)_i \leftarrow \sum_{j=1}^{k} r_j \cdot \gamma\left(x_j\right)_i$
   - Parties run batched MAC-check on $c$, where party $P_i$ inputs $c$ and $\gamma(c)_i$.
2. If the MAC-check fails, output $\perp$ and abort.
3. Open each party $P_i$'s input sent to every other party $P_j$, to compute $x \leftarrow \sum_{i\in P} x_i$. Run MAC-check with party $P_i$'s input $x$ and $\gamma(x_i)$, to verify $\langle y\rangle$. In case this check fails, output $\perp$ and abort; otherwise output $x$.

## Secure Neural Network Interface

A significant challenge while coming up with Secure NN Interface that is efficiently compatible with SPDZ is the choice of activation function. For instance, non-linear functions like softmax introduce a computational overhead of converting from Arithmetic circuit to Yao's Garbled Circuit for division. One way of avoiding this overhead is approximating the function, but this can result in further loss of accuracy. Another way can be the use of ReLU function, which involves a relatively cheaper comparison operation. Our plaintext results on accuracy seem to indicate that ReLU is the best choice for attaining greater accuracy with less hidden layers. Thus, our experiments primarily focus on models of FNN and CNN that use only ReLU in the non-linear layers.

Although the use of ReLU has its benefits, not using

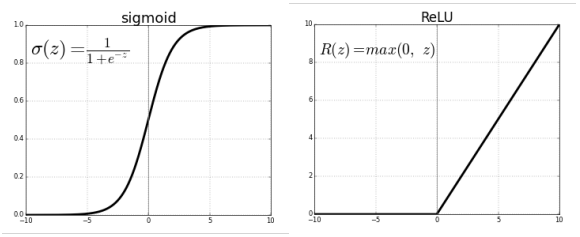


Figure 2: Range of Sigmoid and ReLU

smooth functions that restrict the value of the output, leads to overflow after just a few operations on the features. This is because ReLU has no upper bound, whereas functions like softmax/sigmoid restrict the values of the outputs in the range of $[0, 1]$, as is evident in Fig. 2. Introducing another set of comparisons on top of the ReLU functionality for limiting the values can result in comparison operation becoming the bottleneck of computation.

To overcome this problem, we take help of **initialization techniques** popular in Machine Learning. An appropriate parameter initialization of Neural Networks can lead to optimizations being reached in the least time, otherwise converging to a minima using gradient descent can become tedious. We use two different initialization techniques for our weights and biases respectively. For our weights, we use *Xavier Initialization* i.e. dividing the randomly initialized values by square root of the number of neurons in the previous layer. For our biases, we use *Zero Initialization* i.e. setting all values to zero. These helped us control the range of gradient and prevent them from vanishing or exploding too quickly. Furthermore, this is achieved without introducing an extra set of comparison operations for each value in every layer.

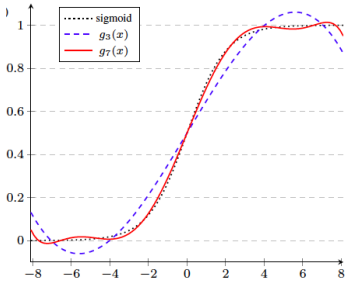


Figure 3: Sigmoid function and Least Square Approximation (Kim et al. 2018)

In case of our FNN model, we have compared the accuracy for ReLU, polynomial approximation and linear approximation of sigmoid. In the polynomial case, least square approximations of sigmoid have been used in place of the sigmoid function, as it is bounded in the interval $[0, 1]$ for a wider range and thus helps prevents overflow.

$$g_7(x) = b_0 + b_1\times(x/8) + b_3\times(x/8)^3 + b_5\times(x/8)^5 + b_7\times(x/8)^7 \quad (1)$$

Eq. 1 represents the degree-7 polynomial approximation of

sigmoid, where $b_0, b_1, b_3, b_5, b_7$ = *(0.5, 1.73496, – 4.19407, 5.43402, – 2.50739).*

For the linear approximation, we used the function given by eq. 2, where $c = 0.28$.

$$f(x) = 0.5 \times (1 + c \times x) \tag{2}$$

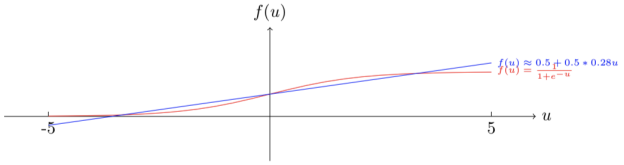


Figure 4: Sigmoid Function and Linear Approximation of Sigmoid

## Description of the Model

**Fully-connected Neural Network** This model uses a multi-layer perceptron and we evaluate it with 3 different function as the activation function, namely ReLU, Least-square approximation of Sigmoid and, Linear approximation of Sigmoid.

| | |
|---|---|
| *Fully-Connected:* | input image $28 \times 28$, connects the incoming 784 nodes to the outgoing 128 nodes i.e. $R^{100\times1} \leftarrow R^{100\times784} \cdot R^{784\times1}$ |
| *Activation:* | returns the post-nonlinearity values of the input. |
| *Fully-Connected:* | connects the incoming 128 nodes to the outgoing 10 nodes i.e. $R^{10\times1} \leftarrow R^{10\times100} \cdot R^{100\times1}$ |
| *Activation:* | returns the post-nonlinearity values of the input |

**Convolutional Neural Network** The layers in the model are similar to the one introduced by Chameleon (Riaz et al. 2018). On top of the layers, we involve normalisation of input and weight initialization techniques to keep the output values in check.

| | |
|---|---|
| *Convolution*: | input image $28 \times 28$, filter size $5 \times 5$, stride $(2, 2)$, number of output channels 5 i.e. $R^{5\times196} \leftarrow R^{5\times25} \cdot R^{25\times196}$ |
| *Activation*: | ReLU function applied to each input |
| *Fully-Connected*: | connects the incoming 980 nodes to the outgoing 100 nodes i.e. $R^{100\times1} \leftarrow R^{100\times980} \cdot R^{980\times1}$ |
| *Activation*: | ReLU function applied to each input |
| *Fully-Connected*: | connects the incoming 100 nodes to the outgoing 10 nodes i.e. $R^{10\times1} \leftarrow R^{10\times100} \cdot R^{100\times1}$ |

# Experimental Results

We have based our entire implementation on a single framework i.e. MP-SPDZ (MPS N1 Analytics). For the field-based experiments, we work on 128-bit prime field where statistical security parameter $\sigma$=57. The framework provides a C++ online-phase implementation of SPDZ based on SPDZ-2 (Damgård et al. 2013), which is the most recent online protocol for SPDZ. The offline phase implementation is based on Overdrive (Keller, Pastro, and Rotaru 2018) in Low Gear which is the most efficient SPDZ preprocessing protocol for two parties. For most efficient results in SPDZ2k, the preprocessing should be based on TinyOT (Nielsen et al. 2012), (Wang, Ranellucci, and Katz 2017) as compared to the native SPDZ2k triples. Since such an implementation of SPDZ2k is not available in MP-SPDZ and the native implementation offers roughly the same experimental numbers as obtained modulo $p$ for the online phase, we resort to extrapolations based on the various experimental results, which we further explain in the last section.

**Setup** We run all experiments in two-party setting, with each party equipped with an Intel i9-7960X CPU clocked at 2.80 GHz with 128 GB RAM, and running the Arch Linux operating system. The parties are connected over a 10Gbps link and present in the same region.

## Results in Fields

The following results are highlighted for different experiments conducted on the MNIST dataset for a 128-bit prime field.

**Activation Function** In order to compare the performance of various activation functions as a part of our FNN model, we present the accuracy attained when all non-linear layers used the same activation function. The tests were conducted for 15 epochs, with batch size 128 and learning rate 0.5.

- The accuracy attained in plain-text with sigmoid function was 95%. Both linear and polynomial approximation incur significant loss of accuracy. (Table 1)
- The use of initialization techniques not only helped us avoid overflow, but also helped us attain better accuracy with just one hidden layer as compared to all the other works that used two hidden layers with more neurons. For instance, the FNN model of SecureML (Mohassel and Zhang 2017) with two-hidden layers with 128 neurons each, managed to attain 93% accuracy while incurring a loss of 1% as compared to the plain-text model.
- As for the precision, a 13-bit precision in the floating-point operations is enough to attain results that match the accuracy associated with the plain-text version.

| Function | Accuracy |
|---|---|
| Linearly Approximated Sigmoid | 87.50% |
| Polynomial Approximated Sigmoid | 93.60% |
| ReLU | 97.73% |

Table 1: Accuracy attained by different activation functions in Non-linear layer.

**Training and Inference** We have specified the training time for one iteration and inference time for a single image in Table 2. The online phase was run using a single thread. The number of triples and random bits needed have been mentioned in multiple of thousand.

- The Low-Gear based preprocessing was run using 16 threads, such that communication per triple turned out to be 14.922 kbits.

| Task | NN | Offline Phase | | | Online Phase | |
|---|---|---|---|---|---|---|
| | | # triples (k) | # random-bits (k) | Time (s) | Communication (MB) | Time (s) |
| Training | FNN | 35968 | 20504 | 354 | 1144.32 | 24.32 |
| | CNN | 48641 | 52121 | 479 | 1497.60 | 29.18 |
| Inference | FNN | 187 | 122 | 1.84 | 5.98 | 0.11 |
| | CNN | 260 | 263 | 2.56 | 8.16 | 0.16 |

Table 2: Results for Fields for a single iteration in training where batch size is 128 and a single inference

- The major optimization equipped for implementation is merging the communication of multiple operations into a single round. This technique is well suited for Deep Learning in particular, since NNs extensively involve performing the same operation on the entire layer which can be very slow if a sequential approach is used.
- To attain such parallelism for truncation, we have performed the entire training and testing on integers, where each value was scaled up by $2^{\mathsf{precision}}$ during input, this allowed us to truncate entire rows of matrices obtained after each matrix multiplication in one round.
- For training, to attain an accuracy greater than 97% FNN will require 2000 iterations, with batch size 128 and learning rate 0.5. Similarly, CNN will require 7000 iterations with batch size 128 and learning rate $2^{-5}$.

Although the numbers presented so far seem promising when seen with respect to other work in the malicious setting, but they fail to compete with the run-time of privacy-preserving machine learning in the passive setting. The FNN evaluation using (Mohassel and Zhang 2017) in rings for a passive setting is multifolds faster than these results for fields in active setting. Therefore, we further present a highly promising extrapolation of our results for a ring-based setting in the next section.

### Results for Rings

*SPDZ2k* (Cramer et al. 2018) is a particular member of the SPDZ family that supports arithmetic over rings of the form $2^k$. The protocol supports arithmetic modulo $2^k$, which is correspondent to the native 32-bit/64-bit operations performed in CPUs, thus makes way for highly efficient implementations. The main idea behind the protocol is that it accepts shares modulo $2^k$ but performs computation over a larger ring modulo $2^{k+s}$ where the statistical security parameter is $\sigma = s - log(s)$ and correctness is guaranteed modulo $2^k$. Due to the presence of many divisors in the ring, MACs cannot protect integrity of an element $x \in Z_{2^{k+s}}$, but can do so for its $k$ lower order bits i.e. $x$ mod $2^k$.

The online-phase of a ring-based implementation benefits from two advantages i.e. faster local computation and reduced communication. The former arises from the elimination of modular reductions. Figure 5 gives a clear picture of how impactful this elimination can be. It shows the time for conducting 1000,000 sequential multiplications in GMP modulo a 128-bit prime for fields and a multiplication on custom data type using two 64 bit unsigned integers, modulo $2^{128}$ for rings in C++ and highlights around a 5 times improvement in the ring-based case. The reduced

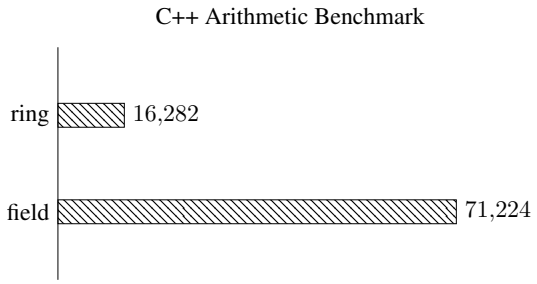


Figure 5: Time in microseconds for 1,000,000 multiplication in C++, using $mpz\_t$ (modulo 128 bit prime) and custom datatype modulo $2^{128}$

data communication roots from the use of bit-triples, that enable computation by sending only $k$ least significant bits of an element, rather than the entire element in case of SPDZ. The results of leveraging these two advantages have already been experimentally shown by (Damgård et al. 2019) for a protocols of primitives like comparison, bit-decomposition etc. The online-phase in rings turned out to give a five fold improvement in computation and an eighty-five fold reduction in the online communication for the Java implementation using FRESCO (Fre Alexandra Institute) Framework. Since benchmarks on Java arithmetic showed similar folds of improvement in their work as highlighted in Fig. 5, and the improvement folds of comparison etc. were even better than multiplication, it can safely be assumed that a C++ implementation of the online-phase in rings would give multifold improvement for the case of NN. Such an improvement can enable actively secure NN models to match the latency of SecureML and thus render them even more appropriate for practical use. As for the offline phase, the TinyOT based preprocessing in rings does lag behind the LowGear version in fields, but the gap can be diminished via SHE-based techniques, which can further enable SPDZ based Deep Learning to compete with the passive versions so far.

## Future Work

Apart from obtaining experimental results for NN based on rings, a direction for further research can be building a SPDZ library specific to Machine Learning. Since Machine Learning in itself is a broad field, yet many of its applications involve extensive use of basic computation like matrix multiplication, batched operations etc., a library catering to these in particular can provide a platform for further implementations and help unify the research in Privacy Preserving Machine Learning.

## References


Barak, A.; Escudero, D.; Dalskov, A.; and Keller, M. 2019. Secure Evaluation of Quantized Neural Networks. *IACR Cryptology ePrint Archive* 2019:131. https://eprint.iacr.org/2019/131.

Barni, M.; Orlandi, C.; and Piva, A. 2006. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th Workshop on Multimedia and Security*, MM&Sec '06, 146–151. New York, NY, USA: ACM.

Chen, V.; Pastro, V.; and Raykova, M. 2019. Secure computation for machine learning with SPDZ. *CoRR* abs/1901.00329.

Cramer, R.; Damgård, I.; Escudero, D.; Scholl, P.; and Xing, C. 2018. SPDZ2k: efficient MPC mod 2k for dishonest majority. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, 769–798.

Damgård, I.; Keller, M.; Larraia, E.; Pastro, V.; Scholl, P.; and Smart, N. P. 2013. Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits. In Crampton, J.; Jajodia, S.; and Mayes, K., eds., *Computer Security – ESORICS 2013*, 1–18. Berlin, Heidelberg: Springer Berlin Heidelberg.

Damgård, I.; Escudero, D.; Frederiksen, T.; Keller, M.; Scholl, P.; and Volgushev, N. 2019. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. *2019 2019 IEEE Symposium on Security and Privacy (SP)* 1325–1343.

Alexandra Institute. FRESCO - a framework for efficient secure computation. https://github.com/aicis/fresco.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K. E.; Naehrig, M.; and Wernsing, J. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, 201–210. JMLR.org.

Hunt, T.; Song, C.; Shokri, R.; Shmatikov, V.; and Witchel, E. 2018. Chiron: Privacy-preserving machine learning as a service.

Juvekar, C.; Vaikuntanathan, V.; and Chandrakasan, A. 2018. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, 1651–1669. Baltimore, MD: USENIX Association.

Keller, M.; Pastro, V.; and Rotaru, D. 2018. Overdrive: Making SPDZ great again. In Nielsen, J. B., and Rijmen, V., eds., *Advances in Cryptology – EUROCRYPT 2018*, 158–189. Cham: Springer International Publishing.

Kim, M.; Song, Y.; Wang, S.; Xia, Y.; and Jiang, X. 2018. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR Med Inform* 6(2):e19.

Liu, J.; Juuti, M.; Lu, Y.; and Asokan, N. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 619–631. New York, NY, USA: ACM.

Mohassel, P., and Rindal, P. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, 35–52. New York, NY, USA: ACM.

Mohassel, P., and Zhang, Y. 2017. Secureml: A system for scalable privacy-preserving machine learning. *2017 IEEE Symposium on Security and Privacy (SP)* 19–38.

N1 Analytics. MP-SPDZ - versatile framework for multi-party computation. https://github.com/n1analytics/MP-SPDZ.

Nielsen, J. B.; Nordholt, P. S.; Orlandi, C.; and Burra, S. S. 2012. A new approach to practical active-secure two-party computation. In Safavi-Naini, R., and Canetti, R., eds., *Advances in Cryptology – CRYPTO 2012*, 681–700. Berlin, Heidelberg: Springer Berlin Heidelberg.

Orlandi, C.; Piva, A.; and Barni, M. 2007. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security* 2007(1):037343.

Riaz, M. S.; Weinert, C.; Tkachenko, O.; Songhori, E. M.; Schneider, T.; and Koushanfar, F. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, 707–721. New York, NY, USA: ACM.

Rotaru, D., and Wood, T. 2019. Marbled circuits: Mixing arithmetic and boolean circuits with active security. Cryptology ePrint Archive, Report 2019/207. https://eprint.iacr.org/2019/207.

Rouhani, B. D.; Riazi, M. S.; and Koushanfar, F. 2018. DeepSecure: Scalable Provably-Secure Deep Learning. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 1–6.

Tramèr, F., and Boneh, D. 2019. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Wagh, S.; Gupta, D.; and Chandran, N. 2018. SecureNN: efficient and private neural network training. *IACR Cryptology ePrint Archive* 2018:442.

Wang, X.; Ranellucci, S.; and Katz, J. 2017. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 39–56. New York, NY, USA: ACM.

Zheng, W.; Popa, R. A.; Gonzalez, J. E.; and Stoica, I. 2019. Helen: Maliciously secure coopetitive learning for linear models. *2019 IEEE Symposium on Security and Privacy (SP)*.