FANNG-MPC: Framework for Artificial Neural Networks and Generic MPC

Najwa Aaraj*, Abdelrahaman Aly*, Tim Güneysu†, Chiara Marcolla*,
Johannes Mono†, Rogerio Paludo*, Iván Santos-González*, Mireia Scholz*,
Eduardo Soria-Vazquez*, Victor Sucasas*, Ajith Suresh*

* *Technology Innovation Institute, Abu Dhabi*
† *Ruhr-University Bochum, Bochum, Germany*

***Abstract*—In this work, we introduce FANNG-MPC, a versatile secure multi-party computation framework capable to offer active security for privacy-preserving machine learning as a service (MLaaS). Derived from the now deprecated SCALE-MAMBA, FANNG is a data-oriented fork, featuring novel set of libraries and instructions for realizing private neural networks, effectively reviving the popular framework. To the best of our knowledge, FANNG is the first MPC framework to offer actively secure MLaaS in the dishonest majority setting, specifically two parties.**

FANNG goes beyond SCALE-MAMBA by decoupling offline and online phases and materializing the dealer model in software, enabling a separate set of entities to produce offline material. The framework incorporates database support, a new instruction set for pre-processed material, including garbled circuits and convolutional and matrix multiplication triples. FANNG also implements novel private comparison protocols and an optimized library supporting Neural Network functionality. All our theoretical claims are substantiated by an extensive evaluation using an open-sourced implementation, including the private evaluation of popular neural networks like LeNet and VGG16.

***Index Terms*—Multi-Party Computation, Privacy-Preserving Machine Learning, Homomorphic Encryption, Neural Networks, MPC, PPML.**

1. Introduction

The increasing relevance of AI and its brisk market penetration have sparked unprecedented interest in privacy-enhancing technologies (PETs) applied to machine learning (ML). This interest arises from the necessity to protect users' data, driven either by user concerns or obligations to regulations such as the General Data Protection Regulation (GDPR) in Europe and the California Consumer Privacy Act (CCPA) in the United States. The privacy problem in ML is twofold. Firstly, training AI models requires vast amounts of data, which may be distributed among various entities and subject to diverse privacy regulations. Secondly, the massive size of ML models makes deployment on the user side impractical, leading AI towards the cloud service paradigm, where users must send their data to the cloud. This poses a considerable privacy risk and can hinder the wide adoption of AI in scenarios involving sensitive data.

The privacy bottleneck can be mitigated through various PETs. Indeed, solutions based on data anonymization, generative models, or differential privacy have garnered attention from the research community, contributing to the coining of the term Privacy-Preserving Machine Learning (PPML). However, within the realm of PPML, confidential computing has gained recent prominence. The concept involves performing data processing confidentially using secure Multi-Party Computation (MPC) and/or Fully Homomorphic Encryption (FHE), offering enhanced privacy guarantees. Several PPML frameworks have been proposed, incorporating MPC, FHE, or a combination of both. Unfortunately, the majority of these frameworks either employ generic constructions without ML-specific optimizations or utilize tailored protocols linked to specific trust settings, often confined to honest majority and semi-honest (i.e., passive) security, as depicted in Table 1.

In this work, we introduce a novel MPC-based framework named FANNG-MPC, derived from the popular SCALE-MAMBA [3] framework. While maintaining the diverse set of protocols inherent in SCALE-MAMBA, FANNG introduces novel constructions tailored for private ML inference that operate in the dishonest majority setting with active security.

1.1. SoTA on PPML frameworks

In this section, we provide an overview of several important PPML frameworks that are based on MPC and HE techniques. While we won't go into the nuance of their protocol constructions, we will briefly discuss their settings, key strengths and limitations, as well as the security model they adhere to. Given the extensive nature of the literature on this topic, we will focus on a selected few frameworks from each category, as listed in Table 1. For a more comprehensive understanding, we recommend referring to [12], [35], [50] for detailed information.

As illustrated in Table 1, the majority of the studies focus on scenarios involving a small number of parties, specifically 2, 3, and 4. Frameworks supporting 2 parties are mainly based on passive security, except for XONN which is entirely based on GCs and can achieve active

TABLE 1: Summary of PPML frameworks (only a representative subset) in the literature. Security: ○ - semi-honest, ◐ - malicious (abort), ◕ - malicious (fair), ● - malicious (GOD). Notations: HE - Homomorphic Encryption, GC - Garbled Circuits, SS - Secret Sharing, OT - Oblivious Transfer, FSS - Function Secret Sharing.

Setting	Framework	Security	Techniques
2-party (dishonest majority)	CryptoNets [24]	○	HE
	SecureML [47]	○	HE+GC+SS
	MiniONN [42]	○	HE+GC+SS
	DeepSecure [57]	○	GC
	Gazelle [29]	○	HE+GC+SS
	XONN [54]	◐	GC
	CryptFlow2 [53]	○	HE+OT+SS
	Delphi [45]	○	HE+GC+SS
	ABY2.0 [52]	○	OT+GC+SS
	Cheetah [28]	○	HE+OT+SS
2-party (+ helper)	Chameleon [55]	○	OT+GC+SS
	Crypten [36]	○	SS
	LLAMA [25]	○	FSS
3-party (honest majority)	ABY^3 [46]	◐	GC+SS
	ASTRA [15]	◕	SS
	SecureNN [61]	○	SS
	SWIFT [37]	●	SS
	Falcon [62]	◐	SS
4-party (honest majority)	Trident [16]	◕	GC+SS
	FLASH [11]	●	SS
	SWIFT [37]	●	SS
	Fantastic Four [19]	●	SS
	Tetrad [39]	●	GC+SS
n-party	Cerebro [64]	◐	HE+GC+SS
	MPClan [38]	◕	SS

security through cut-and-choose technique, though this is rather inefficient. In the following sections, we present a concise overview of research within each category.

Two-party (2PC): The field of PPML for two parties traces its origins to a seminal work by Lindell and Pinkas [41]. They proposed a secure algorithm for data mining, specifically for decision trees. Subsequent research following [41] focused on algorithms such as k-means clustering, linear regression, and logistic regression. However, these approaches suffer from high efficiency overheads and are primarily theoretical in nature.

Later advancements in techniques like Levelled Homomorphic Encryption (LHE) have paved the way for innovative solutions like CryptoNets [24]. CryptoNets introduced a non-interactive solution for private neural network predictions over encrypted data. They employed LHE-friendly approximations for activation functions and made the assumption that one party possesses the model and evaluates it on the private data of another party.

SecureML [47] utilized techniques such as GC and SS to create efficient protocols for PPML inference, including neural networks. GC was employed for evaluating non-linear functions like Sigmoid, ReLU, and SoftMax, while SS-based techniques were utilized for evaluating linear layers. SecureML also introduced MPC-friendly versions of functions such as Sigmoid through the use of a piecewise polynomial evaluation paradigm, and demonstrated practicality of MPC-based techniques for PPML tasks. Subsequently, several works such as MiniONN [42], DeepSecure [57], Gazelle [29], and XONN [54] focused on improving efficiency by leveraging advancements in underlying primitives.

Recent works such as CryptFlow2 [53], Delphi [45], ABY2.0 [52] and Cheetah [28] employ state-of-the-art optimizations in HE and OT Extension domains. They utilize a combination of techniques such as arithmetic and Boolean secret sharing and garbled circuits, using a mixed protocol approach to achieve efficient solutions for PPML inference. Most of these works incorporate a preprocessing phase to enhance online phase efficiency, while works like ABY2.0 optimizes the online phase further through function-dependent preprocessing. However, with the exception of XONN,[1] these works only offer security against semi-honest adversaries and are unable to handle malicious corruptions.

Two-party with helper (2PC$^+$): In many real-world scenarios involving a server and a client, a 2PC setting is commonly used. However, even in the semi-honest solutions within this setting, there is a significant computational and communication burden in the preprocessing phase to generate correlated randomness in a distributed manner. To address this efficiency issue, some studies have explored the use of an external dealers, sometimes in the form of a signle trusted entity to generate the correlated randomness during the preprocessing phase [59]. This approach has proven beneficial in improving the efficiency of complex tasks such as PPML inference, as demonstrated in works like Chameleon [55] and Crypten [36]. More recently, works like LLAMA [25] have also leveraged this setting in their 2PC protocols, which are designed using the Function Secret Sharing (FSS) paradigm and have demonstrated practicality.

Three-party (3PC): The 2PC setting has a couple of significant drawbacks. First, there is a substantial amount of computation and communication involved. Second, these protocols have a high overhead for ensuring security against malicious corruptions. This typically involves computationally expensive operations like cut-and-choose and message authentication codes (MAC). Furthermore, when operating in a dishonest majority setting like 2PC, the level of security achieved by these protocols is limited to malicious security with abort, as indicated in Table 1.

In response to these limitations, subsequent works like ABY3 [46] and ASTRA [15] focused on a 3-party honest majority, demonstrating improvements over the 2PC protocols. Later on, the 3PC protocol in SWIFT [37] utilized function-dependent preprocessing and distributed zero-knowledge proofs to enhance communication and achieve the strongest output guarantee, i.e. GOD. These protocols assume a more flexible trust setting where only one party can be maliciously corrupt ($t < n/2$). Thus, they relax the trust setting to achieve higher efficiency or delivery guarantees.

1. XONN's method can be generalized to the malicious setting, but its underlying cut-and-choose method is expensive for practical use.

In parallel, works like SecureNN [61] and Falcon [62] concentrated on enhancing the efficiency of underlying PPML primitives like Maxpool, normalization and division. They achieved this by utilizing MPC-friendly counterparts for these operations. Through these improvements and clever engineering, these works were able to support private training of deep neural networks like ResNet-18.

Four-party (4PC): Although 3PC could enhance the efficiency of 2PC counterparts, they faced challenges such as high computation caused by distributed zero-knowledge proofs and communication requirements due to cut-and-choose techniques for either generating the correlated randomness or performing verification of the computation. These overheads become impractical when considering the PPML training of deep ML models like ResNet-18. Consequently, Trident [16] and FLASH [11] aimed to address these issues by focusing on a super honest majority setting involving 4 parties ($t < n/3$). These approaches eliminated costly distributed zero-knowledge proofs and reduced computation to cheap symmetric key operations.

In later work, the authors of Fantastic Four [19] introduced an online-only robust protocol, building on the 4PC protocol in SWIFT. They also demonstrated techniques for *private robustness*, guaranteeing that the function output is correctly delivered to honest parties without revealing any other party's input. Recently, Tetrad [39] enhanced the communication of existing 4PC protocols and showcased protocols in the function-dependent and online-only paradigms, all with the same communication complexity.

n-party (MPC): While several works explore small parties ($n < 5$), there are only a few that specifically address the support for more than four parties in the domain of PPML. One such work is Cerebro [64], which introduced an end-to-end collaborative learning platform by developing a compiler based on SCALE MAMBA (SM) and EMP-Toolkit (EMP). Essentially, this platform converts ML-friendly APIs into either SM or EMP code, enabling the execution of various protocols supported by these frameworks. However, Cerebro lacks ML-specific optimization for SM and inherits the limitations of SM when compiling large programs.

In an orthogonal direction, MPClan [38] focused on PPML inference in n-party scenarios by utilizing function-dependent preprocessing. Nonetheless, their technique is limited to $n \leq 11$ parties due to the exponential growth of computation and storage with the number of parties.

1.2. Our Contributions

In this work, we introduce FANNG-MPC, a versatile framework designed for efficient protocols using secure multi-party computation (MPC) techniques. Referred to as FANNG in short, our framework is an independent fork derived from `SCALE-MAMBA` [3]. It maintains the diverse set of protocols included in `SCALE-MAMBA`, which operates over various fields ($\mathbb{F}_p$), including binary. Concerning the threat model, FANNG accommodates both honest and dishonest majority settings, ensuring active security.

While FANNG serves as a general-purpose MPC framework, its design is specifically tailored to enable private ML inference in a two-party scenario, involving a model owner and a client as illustrated in Figure 1. To the best of our knowledge, FANNG is the first framework exclusively supporting private ML inference in a dishonest majority setting with active security.

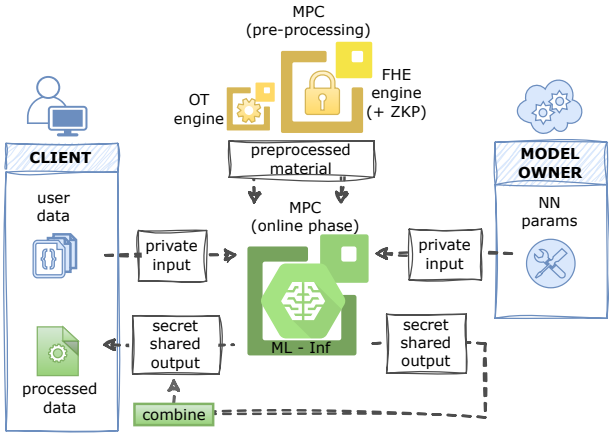

Figure 1: Private inference between a client in possession of private data and a cloud server owning the model in FANNG-MPC.

FANNG extends the functionality of `SCALE-MAMBA` by separating pre-processing from the online phase and introducing a dealer model. This model enables the generation of offline elements by a distinct set of parties, facilitated by an I/O API connecting the MPC engine to a file system or a database. This capability supports the storage and loading of gabled circuits for comparisons, random masks for probabilistic truncation, and matrix and convolutional triples. FANNG features an extended instruction set to accommodate the dealer model, along with a separate engine for matrix triple generation based on BGV, executable by external dealers. Matrix triples generated externally can be converted into a 2PC version using an MPC-2PC converter. Moreover, FANNG includes specialized implementations to enhance the performance of neural network (NN) operations. This encompasses convolutions, fully connected layers, and dedicated libraries for folding and normalization, ensuring optimal communication rounds.

Unlike a mere prototype, FANNG delves deeper into system-level details, identifies challenges, and addresses them using novel protocols and considerable engineering effort. This approach positions FANNG closer to a real-world implementation, facilitating more accurate performance results. For example, `SCALE-MAMBA` faces limitations in compiling large programs with the default optimizer for communication rounds (the `-O3` flag). Similarly, the MAMBA compiler within `SCALE-MAMBA` cannot process programs exceeding 2^{32} bytecodes, posing challenges for implementing large neural networks like VGG16 [58], as considered in this work. FANNG addresses these issues through the application of novel engineering techniques.

To summarize, we have the following contributions:

- Introduction of a novel MPC-based framework designed for private machine learning (ML) inference in a two-party setting with active security.
- Design and implementation of a dedicated Dealers Module, utilizing state-of-the-art FHE-based protocols to preprocess matrix triples, and the associated Converter Module.
- Support for a dedicated Preprocessing Unit capable of handling various types of input-independent data, including support for offline garbling, and incorporating efficient storage mechanisms for persistent sharing of values.
- Novel protocol for combining ReLU with truncations, resulting in improved instruction size and efficiency.
- Specialized ML libraries for designed for efficient linear transformations and an enriched set of instructions.
- Open-sourced implementation with detailed evaluations, including private inference of large NNs like VGG16 [58].

2. Framework Design

In this section, we discuss the design of our FANNG-MPC framework. Our framework prioritizes private machine learning (ML) inference within a 2-party context, emphasizing *active* security. Instead of merely designing a prototype, our focus extends to intricate system-level aspects such as instruction size, storage requirements, support for vectorization, and I/O handling (cf. §3.7). This approach allows us to emulate an MPC framework that closely resembles a real-world implementation. Our objective is to demonstrate the practicality of actively-secure MPC in a dishonest majority setting, a scenario still considered costly in the existing literature. Inherited from SCALE-MAMBA, FANNG operates over prime-order fields ($\mathbb{F}_p$) that encompass binary ($\mathbb{F}_2$), with plans for future work to include support for rings.

We chose SCALE-MAMBA as our baseline among existing MPC frameworks because it supports various protocols with *active security*, including full-threshold Secret Sharing (SS) [7], [32], [56], and honest majority setups via Shamir SS, replicated SS, and generic monotone span programs [33], [60]. We would like to stress that SCALE-MAMBA is the only framework in existence that allows us to combine those protocols with Boolean circuit evaluation via [HSS17] [27] whilst providing active security. However, SCALE-MAMBA, designed as a versatile solution for implementing circuits, lacks optimizations necessary for the practical evaluation of complex applications like privacy-preserving Machine Learning. For instance, it prioritizes flexibility over performance and lacks a decoupled *pre-processing* phase, which happens in parallel with the online execution. Moreover, SCALE-MAMBA is no longer maintained with the last update in June 2022[2]. Our primary objective is to enhance the functionality of this widely used MPC framework, ensuring compatibility with real-world setups and better suited for concrete machine learning applications, essentially revitalizing the framework.

2. https://homes.esat.kuleuven.be/~nsmart/SCALE/

Figure 2 depicts the comprehensive design of FANNG, comprising primarily of five components. A brief overview of these components, in the context of our ML inference application (cf. Figure 1), is provided next, with detailed descriptions provided in §3.

1. **Dealer Module (§3.1):** In a two-party scenario, outsourcing the pre-processing-phase computations (input-independent) to a trusted helper can significantly enhance the efficiency of the online phase [25], [36], [55]. However, the existence of such a helper contradicts the main axiom of actively secure setting with dishonest majority, i.e."the parties do not need to trust anyone", and thus may not align with real deployments. To circumvent this, we incorporated a dealer module which replaces the trusted helper with a group of untrusted dealers. These dealers engage in an MPC protocol to generate correlated randomness, reducing the trust requirement from a single trusted helper to one party selected from many. Both the client and model owner can make the dealer selection, thus mitigating the trust concern.

   The dealer module currently generates matrix triples for fully connected and convolutional layers, addressing complex operations in ML model pre-processing. It will eventually expand its support to include other pre-processed materials like authenticated bits and oblivious transfer. The design is flexible enough to accommodate multiple dealer modules, reflecting our anticipation of dedicated services replacing these modules in the future. For example, we are actively working on FPGA hardware acceleration to support matrix triple generation. This adaptable design enables the integration of various dedicated services for efficient pre-processing within FANNG.
2. **Converter Unit (§3.2):** The dealer module is designed to be adaptable in various scenarios involving different numbers of parties. To enable the two-party execution of the machine learning inference between the model owner and the client, we need to convert the pre-processed material into a state that is compatible with this setup. The converter unit is specifically created to ease this transition, converting pre-processed materials from one type to another that is suitable for online evaluation involving a different set of parties. In this work, we focus on dealers in the dishonest majority setting and the subsequent conversions needed for online evaluation in a two-party scenario.
3. **Pre-processing Unit (§3.3):** This unit is responsible for generating input-independent data essential for the online evaluation of the MPC protocol. This approach has proven to significantly enhance the efficiency of the online phase, leading to practical runtimes [22], [26], [30], [32]. The unit currently covers Garbled Circuits (GCs), truncation masks, daBits, beaver triples, and authenticated singles. In the FANNG roadmap, we aim to extend dealer support for generating all these data types, except for beaver triples and authenticated singles.
4. **Storage Support (§3.4):** A modular component within FANNG, dedicated to persisting information shared among its various units. It revamps the legacy SCALE-MAMBA I/O by introducing a novel controller-based approach, enabling

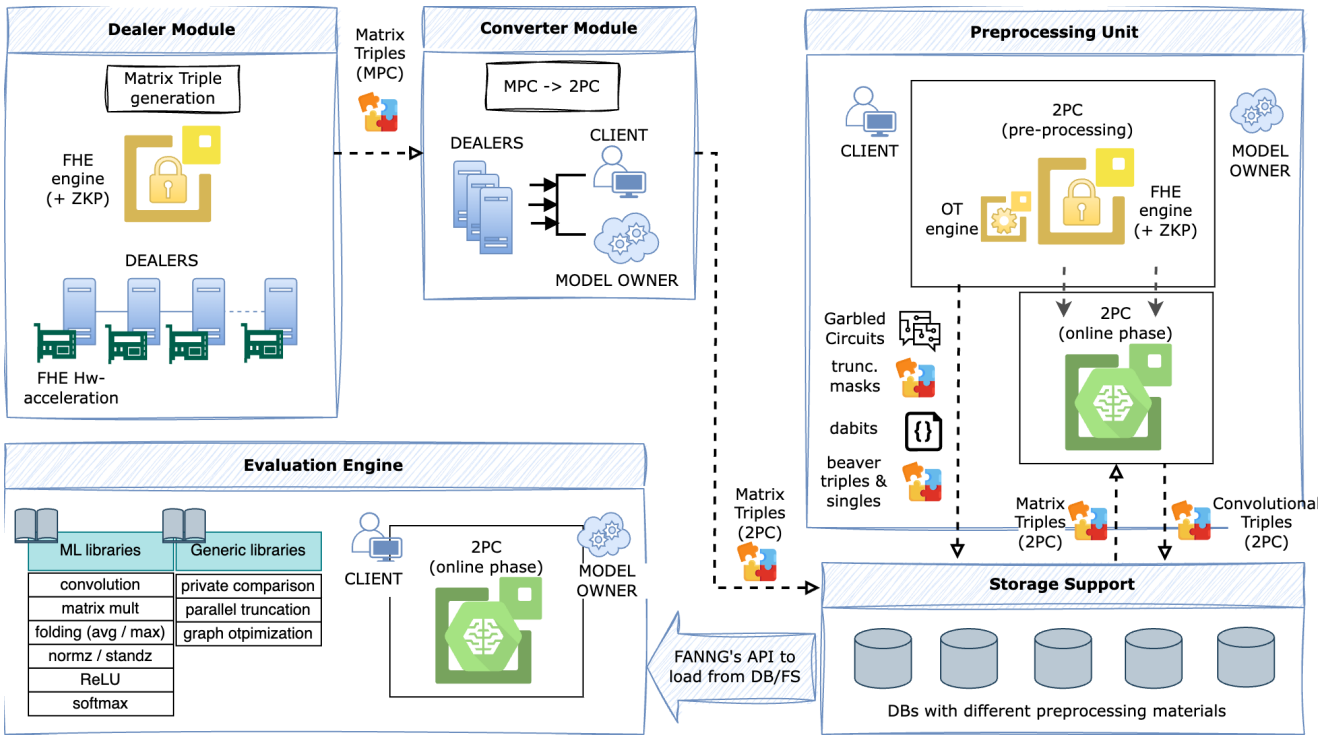

Figure 2: System design of FANNG-MPC framework.

the storage and retrieval of pre-processed material from files or databases, whilst maintaining backwards compatibility. Unlike its predecessor, it offers enhanced customization, allowing users to select the persistence mode via a configuration file. The current version of FANNG provides support for `MySQL 8.0` and the File System (FS), with added `PostgreSQL` support in the Converter Unit, highlighting its adaptability to various database engines.

5. **Evaluation Engine (§3.5):** This engine is responsible for performing the protocol evaluation, private ML inference in our case, serving as the interface between the client and model owner. It orchestrates all other components in FANNG to facilitate the entire evaluation pipeline. This involves signaling dealers in the dealer module for data processing, performing its own preprocessing, utilizing storage support for storing the resulting data, and subsequently retrieving the stored preprocessed data for protocol evaluation. Finally, it performs the evaluation using private inputs from both parties utilizing the preprocessed data.

3. Framework Details

In this section, we discuss the technical details of our framework's components. We explore the integration of new functionalities and optimizations built upon `SCALE-MAMBA`, facilitating private evaluations of large neural networks like ResNet and VGG16. While we have categorized these new functionalities under distinct components, achieving a clear-cut separation is challenging, as many functionalities necessitate the collaborative functioning of multiple components. We start by describing our experimental setup, which we used to obtain the benchmarks for our evaluations.

Experimental setup: For our experiments, we have provided two different hardware test beds:

1. Our *General Purpose MPC* testbed includes 5 servers connected via Gigabit connections with a ping latency of $0.15ms$. Each server has 512 GB of RAM and `Intel(R) Xeon(R) Silver 4208 @ 2.10GHz` processors. They all share a common `/home` directory and are installed with `/sbin/tc` to simulate latency.
2. Our *Machine Learning* testbed comprises a single server equipped with 2TB of RAM and an `Intel(R) Xeon(R) Gold 6250L CPU @ 3.90GHz` processor.

Regarding communications, we conduct all our experimentation considering three relevant setups:

1. **Local:** All parties are emulated on the same machine with no communication cost. Albeit unrealistic, it is typically used as a baseline.
2. **Ping:** Parties run on different machines at point-to-point connection speed of $\approx 0.3ms$, suitable for highly dedicated setups such as in Triples-as-a-Service (TaaS) [59] with Dealers sharing the same data center.
3. **WAN:** Parties run on different machines, considering achievable common ping times for cloud service providers, which is $\approx 20ms$.

From an MPC perspective, we explore three settings, each with active/malicious security:

1. **Full Threshold (2p):** Two-party setting in the dishonest majority setting, tolerating at most 1 corruption.
2. **Full Threshold (3p):** Three-party setting in the dishonest majority setting, tolerating at most 2 corruptions.
3. **Shamir (3p):** Three-party setting in the honest majority setting, tolerating at most 1 corruption.

3.1. Dealer Module

In FANNG, the dealer module generates random matrix triples and convolutional triples required for supporting matrix multiplication and convolutions in private ML inference, as illustrated in Figure 3.

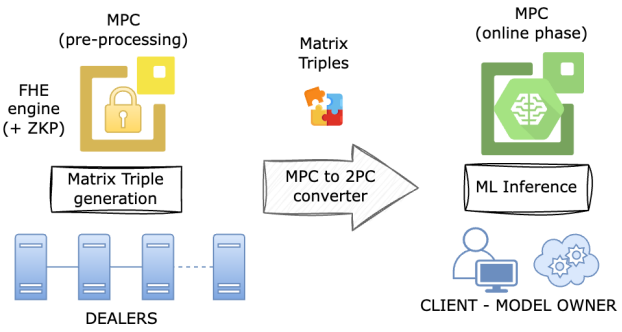

Figure 3: FANNG's dealer model for matrix triple generation.

In FANNG, we opted for Fully Homomorphic Encryption (FHE) [44] techniques to instantiate the dealers, inspired by SCALE−MAMBA's use of the FHE-based SPDZ protocol to generate traditional beaver triples in the offline phase. Specifically, we adopt a levelled version of the BGV scheme [9] to instantiate the FHE. Towards this, we implemented three state-of-the-art protocols, each serving a distinct purpose:

- Π_{Prep}: This protocol depends on leveled homomorphic encryption and serves as a crucial component for generating SPDZ-like matrix triples. We implemented the protocol by Chen et al. [17, Fig. 1], which avoids sacrificing a triple at the expense of an additional multiplicative depth.
- *FHE-based matrix multiplication*: To generate matrix triples instead of traditional beaver triples, Π_{Prep} relies on an algorithm for homomorphically multiplying encrypted matrices. Towards this, we implemented the optimized protocol proposed by Mono et al. [48], which allows the reuse of key-switching keys and eliminates constant multiplications.
- Π_{ZKPoK}: To achieve active security, Π_{Prep} uses zero-knowledge proof of knowledge (ZKPoK) techniques, which establish a global proof of knowledge for a set of ciphertexts. We implemented the approach proposed by Baum et al. [7, Fig. 1], where the authors designed an n-prover protocol, providing the capability to prove the validity of the sum of n ciphertexts instead of proving each one individually.

FHE possesses an inherent characteristic where the error employed in encryption, essential for upholding FHE security properties, increases with each homomorphic operation. Notably, this growth becomes exponential when homomorphic multiplications are executed (see [49] for a detailed analysis of error growth across operations in BGV). As a result, only a finite number of homomorphic operations can be carried out before the error level starts impacting the decryption operation. Hence, it is essential to evaluate error growth within the circuit and choose the FHE scheme parameters accordingly.

This section provides a detailed analysis of parameter estimation, focusing on homomorphic error, particularly within the most complex circuit across all our protocols—matrix multiplication depicted in [48, Fig. 6]. Although Mono et al. [48] offer parameter estimation for BGV schemes, we observe that their protocol overlooks ZKPoK (Π_{ZKPoK}). Consequently, their analysis of matrix triple, starting with a fresh ciphertext, leads to an underestimation of the correct parameters and hence proves insufficient for our scenario. We begin our analysis with a brief recap of all the necessary fundamental concepts.

3.1.1. Mathematical background. Let $\mathcal{R}$ denote the polynomial ring $\mathcal{R} = \mathbb{Z}[x]/\langle x^{\mathsf{n}} + 1\rangle$ and $\mathcal{R}_p = \mathbb{Z}_p[x]/\langle x^{\mathsf{n}} + 1\rangle$, where n is a power of two and p is an integer. Let t and q indicate the plaintext and the ciphertext modulus respectively, where $t \equiv 1 \mod 2\mathsf{n}$. Moreover, $q = q_{L-1}$ denotes a chain of primes, i.e., $q_\ell = \prod_{j=0}^{\ell} p_j$ where $p_j \equiv 1 \mod 2\mathsf{n}$ and $\ell \leq L-1$ [23].

For $x \in \mathbb{R}$, $[x]_p \in [-p/2, p/2)$ denotes the centered representative of $x \bmod p$. For a random polynomial $a \in \mathcal{R}$ and probabilistic distribution χ, we use $a \leftarrow \chi$ to denote the sampling of coefficients independently from χ. We use χ_e to represent the RLWE error distribution and χ_s for the secret key distribution. We also define two parameters, $\mathsf{ZK}_{\mathsf{sec}}$ and $\mathsf{DD}_{\mathsf{sec}}$, related to the simulation-based security of the ZKPoK protocol [7]. $\mathsf{ZK}_{\mathsf{sec}}$ measures the statistical distance between the coefficients of the ring elements (represented as polynomials) in an honest ZKPoK transcript and one generated through simulation. Likewise, $\mathsf{DD}_{\mathsf{sec}}$ denotes the statistical distance between the distributions of honest and simulated transcripts in the distributed decryption protocol.

Canonical embedding and norms: For a random polynomial $a \in \mathcal{R}$, the *infinity norm* of a is defined as $\|a\|_\infty = \max\{|a_i| : 0 \leq i \leq n-1\}$.

The *canonical embedding* of a is the vector obtained by evaluating a at all primitive $2\mathsf{n}$-th roots of unity. The *canonical embedding norm* of a is defined as

$$\|a\|^{\mathsf{can}} = \max_{j \in \mathbb{Z}_m^*} |a(\zeta^j)|,$$

where ζ is a fixed complex primitive $2\mathsf{n}$-th root of unity.

Consider a random polynomial $a \in \mathcal{R}$ with coefficients independently sampled from one of the following zero-mean distributions:

- $\mathcal{DG}(\sigma^2)$, the discrete Gaussian distribution with standard deviation σ.

- $\mathcal{B}_k$, the centered binomial distribution of width k.
- $\mathcal{U}_q$, the uniform distribution over $\mathbb{Z}_q$.

Then, the random variable $a(\zeta)$ is well approximated by centred Gaussian distribution with variance $\mathsf{n} \cdot V_a$, where V_a is the variance of each coefficient in a [21]. Moreover, we can bound the canonical norm of a as $\|a\|^{\mathsf{can}} \leq D\sqrt{\mathsf{n} \cdot V_a}$ with probability $\geq 1 - \mathsf{n} \cdot e^{-D^2}$ [8]. Thus, we set $D = 8$, so that the probability of failure is limited to 2^{-76}.

To compute $\|a\|^{\mathsf{can}}$, we have to study the variance V_a of each coefficient of a. Specifically,

$$V_a = \begin{cases} \sigma^2 & \text{if } a \leftarrow \mathcal{DG}(\sigma^2) \\ k/2 & \text{if } a \leftarrow \mathcal{B}_k \\ q^2/12 & \text{if } a \leftarrow \mathcal{U}_q \end{cases} \quad (1)$$

In the following, χ_e denotes the discrete Gaussian distribution $\mathcal{DG}(\sigma^2)$, and, as in [1], we approximate $\mathcal{DG}(\sigma^2)$ with a centered binomial distribution $\mathcal{B}_k$, with $k = 21$. Thus, the variance of each element of the error vector is $V_e = 21/2 = 10.5$ and its standard deviation is 3.24. Moreover, we set $\chi_s = \mathcal{B}_1$, and thus $V_s = 0.5$.

3.1.2. 4-Leveled BGV scheme. Let q be an integer product of 4 primes, i.e., $q = p_0 \cdot p_1 \cdot p_2 \cdot p_3$. We recall that $q_\ell = \prod_{j=0}^{\ell} p_j$, for any $\ell \leq 3$. A BGV ciphertext is a vector of polynomials $\mathbf{c} \in \mathcal{R}_q^2$, with the following three basic algorithms:

- $\mathrm{KeyGen}(\lambda)$: Key generation algorithm samples $s \leftarrow \mathcal{B}_1$, $a \leftarrow \mathcal{U}_q$ and $e_{sk} \leftarrow \mathcal{DG}(\sigma^2)$, outputs the secret key sk and the public key pk, where

$$\mathsf{sk} = s \text{ and } \mathsf{pk} = (b, a) = [(-a \cdot s + te_{sk}, a)]_{q_\ell}.$$

- $\mathrm{Enc}_{\mathsf{pk}}(m)$: Encryption algorithm takes as input a plaintext $m \in \mathcal{R}_t$ and the public key pk. It samples $u \leftarrow \mathcal{B}_1$, $e_0, e_1 \leftarrow \mathcal{DG}(\sigma^2)$ and outputs the $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$, where

$$\mathbf{c} = (c_0, c_1) = [(b \cdot u + te_0 + m, a \cdot u + te_1)]_q \in \mathcal{R}_q^2,$$

is a ciphertext, ℓ denotes the level and ν the *critical quantity* of $\mathbf{c}$ (see below).

- $\mathrm{Dec}_{\mathsf{sk}}(\mathfrak{c})$: Decryption algorithm takes as input the secret key sk and the ciphertext $\mathbf{c} = (c_0, c_1)$ and outputs

$$m = [[c_0 + c_1 \cdot s]_{q_\ell}]_t .$$

Ciphertext noise: Let $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ be the *extended ciphertext*. The *critical quantity* ν of $\mathbf{c}$ (for the associated level ℓ) is defined as the polynomial $\nu = [c_0 + c_1 \cdot s]_{q_\ell}$, and it determines whether $\mathbf{c}$ can be correctly decrypted [18]. Specifically, if the error does not wrap around the modulus q_ℓ, namely $\|\nu\|^{\mathsf{can}} < q_\ell/2$, the decryption algorithm works. Otherwise, the plaintext cannot be recovered due to excessive noise growth.

To understand the error growth and analyze the critical quantity for any homomorphic operation in the BGV scheme, we refer the readers to [49].

Homomorphic operations: Let $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$ and $\mathfrak{c}' = (\mathbf{c}', \ell, \nu')$ be two extended ciphertexts at the same level ℓ and $\alpha \in \mathcal{R}_t$ be a constant polynomial. Then,

- $\mathrm{Add}(\mathfrak{c}, \mathfrak{c}')$: Addition algorithm outputs

$$([(c_0 + c_0', c_1 + c_1')]_{q_\ell}, \ell, \nu + \nu').$$

- $\mathrm{Mul}(\mathfrak{c}, \mathfrak{c}')$: Multiplication algorithm outputs

$$([(c_0 \cdot c_0' \ , \ c_0 \cdot c_1' + c_1 \cdot c_0' \ , \ c_1 \cdot c_1')]_{q_\ell}, \ell, \nu \cdot \nu').$$

Note that the output of the multiplication is a vector $\mathbf{d} = (d_0, d_1, d_2) \in \mathcal{R}_{q_\ell}^3$. To convert the ciphertext $\mathbf{d}$ back to a ciphertext $\bar{\mathbf{c}} = (\bar{c}_0, \bar{c}_1) \in \mathcal{R}_{q_\ell}^2$ we use a relinearization procedure called *key-switching*.

- $\mathrm{MulConst}(\alpha, \mathfrak{c})$: Plaintext-ciphertext multiplication algorithm outputs

$$(\alpha \cdot \mathbf{c}, \ell, \nu_{\mathsf{const}}),$$

with $\|\nu_{\mathsf{const}}\|^{\mathsf{can}} \leq \mathsf{B}_{\mathsf{const}} \|\nu\|^{\mathsf{can}}$ and $\mathsf{B}_{\mathsf{const}} \approx t\sqrt{\mathsf{n}/12}$ [49].

Modulus switching: The *modulus switching* procedure allows sacrificing one (or more) of the primes p_i that compose the ciphertext moduli q to obtain a noise reduction. As mentioned before, in our case, we switch from a ciphertext modulus q_ℓ to $q_{\ell-1}$. Let $\mathfrak{c} = (\mathbf{c}, \ell, \nu)$, then

- $\mathrm{ModSwitch}(\mathfrak{c})$: Modulus switching algorithm over $\mathfrak{c}$ sets $\boldsymbol{\delta} = t \cdot [-\mathbf{c} \cdot t^{-1}]_{p_\ell}$ and outputs $\left([\frac{1}{p_\ell}(\mathbf{c} + \boldsymbol{\delta})]_{q_{\ell-1}}, \ell - 1, \nu_{\mathsf{ms}}\right)$, where $\|\nu_{\mathsf{ms}}\|^{\mathsf{can}} \leq \frac{1}{p_\ell}\|\nu\|^{\mathsf{can}} + \mathsf{B}_{\mathsf{scale}}$ [49], with

$$\mathsf{B}_{\mathsf{scale}} = Dt\sqrt{\frac{\mathsf{n}}{12}(1 + \mathsf{n}V_s)}. \quad (2)$$

Key switching: The *key-switching* technique is used for either reducing the degree of a ciphertext polynomial, usually the output of a multiplication, or changing the key after a rotation. There are different variants of the key-switching procedure. In this work, we used the Hybrid-RNS which combines the RNS adaptations of BV [10] and GHS [23] variant. The BV variant decomposes d_2 with respect to a base $\mathfrak{b}$ to reduce the error growth and the GHS variant switches to a bigger ciphertext modulus $Q_\ell = q_\ell \cdot P$. Then, key switching takes place in $\mathcal{R}_{Q_\ell}$ and, by modulus switching back down to q_ℓ, the error is reduced again. As a trade-off, we have to make sure that our RLWE instances are secure with respect to Q_ℓ.

However, in the Hybrid-RNS variant, instead of decomposing with respect to each single RNS prime, we group the primes into ω chunks. Specifically, the modulus $q = p_0 \cdots p_{L-1}$ is split in a smaller numbers $\tilde{q}_j = \prod_{i=1}^{k} r_i^{(j)}$ of k elements by gathering the $r_i^{(j)}$ in ω chunks:

$$q_\ell = p_0 \cdots p_\ell = r_1^{(1)} \cdots r_k^{(1)} \cdot \ldots \cdot r_1^{(\omega)} \cdots r_k^{(\omega)} = \tilde{q}_1 \cdot \ldots \cdot \tilde{q}_\omega$$

Hence, we do not apply the decomposition to the base $\mathfrak{b}$ but to the base $\tilde{q}_j$. Thus, we define $\mathcal{D}$ and $\mathcal{P}$ as

$$\mathcal{D}(\alpha) = \left(\left[\alpha \left(\frac{q_\ell}{\tilde{q}_1} \right)^{-1} \right]_{\tilde{q}_1}, \ldots, \left[\alpha \left(\frac{q_\ell}{\tilde{q}_\omega} \right)^{-1} \right]_{\tilde{q}_\omega} \right)$$

$$\mathcal{P}(\beta) = \left(\left[\beta \frac{q_\ell}{\tilde{q}_1} \right]_{q_\ell}, \ldots, \left[\beta \frac{q_\ell}{\tilde{q}_\omega} \right]_{q_\ell} \right).$$

Thus, if $\mathbf{d} = (d_0, d_1, d_2) \in \mathcal{R}^3_{q_\ell}$ is the output of multiplication, we have to extend d_2 from each $\tilde{q}_j$ to Q_ℓ. So, the key switching algorithms are defined as

- KeySwitchGen(s): Sample $\mathbf{a} \leftarrow \mathcal{U}^{\omega}_{Q_L}$, $\mathbf{e} \leftarrow \chi^{\omega}_{e}$. Output key switching key

$$\mathbf{ks} = (\mathsf{ks}_0, \mathsf{ks}_1) \equiv (-\mathbf{a} \cdot s + t \cdot \mathbf{e} + P \cdot \mathcal{P}(s^2), \mathbf{a}) \mod Q_L$$

- KeySwitch($\mathbf{ks}, \mathfrak{d}$): Compute:

$$\mathbf{c}' \equiv (P \cdot d_0 + \langle \mathcal{D}(d_2), \mathsf{ks}_0 \rangle, P \cdot d_1 + \langle \mathcal{D}(d_2), \mathsf{ks}_1 \rangle) \bmod Q_\ell.$$

Set $\boldsymbol{\delta} = t \cdot [-\mathbf{c}' \cdot t^{-1}]_P$, modulus switch back and output $([\frac{1}{P}(\mathbf{c}' + \boldsymbol{\delta})]_{q_\ell}, \ell, \nu_{\mathsf{ks}})$.

The division is done considering either $P \approx \sqrt[\omega]{q}$ if the $r_i^{(j)}$ has the same size or $P \approx \tilde{q}_\iota$ supposing that $\tilde{q}_\iota$ is the biggest among the $\tilde{q}_j$. So $P = \prod_{j=1}^{k} P_j$. Note that, before scaling down with the modulus switching, the noise is $\nu' = \nu \cdot P + \sqrt{\omega(\ell+1)} \cdot \max(\tilde{q}_j) \cdot \mathsf{B}_{\mathsf{ks}}$ [49], where

$$\mathsf{B}_{\mathsf{ks}} = D \cdot t \cdot \mathsf{n} \sqrt{V_e/12}. \tag{3}$$

Thus, the Hybrid-RNS key switching noise after the modulus switching is bounded by $\frac{q_\ell}{Q_\ell}\|\nu'\|^{\mathsf{can}} + \mathsf{B}_{\mathsf{scale}}$, that is,

$$\|\nu_{\mathsf{ks}}\|^{\mathsf{can}} \leq \|\nu\|^{\mathsf{can}} + \sqrt{\omega(\ell+1)}\frac{\max(\tilde{q}_j)}{P}\mathsf{B}_{\mathsf{ks}} + \sqrt{k} \cdot \mathsf{B}_{\mathsf{scale}}.$$

Distributed Decryption: The SPDZ offline phase utilizes a form of distributed decryption, which is also supported in the BGV scheme. A secret key $s \in \mathcal{R}_q$ can be additively distributed among N parties by assigning each party a value s_i, such that $s = s_1 + \ldots + s_N$. As explained in [7], the BGV parameter grows during distributed decryption. Indeed, in the usual BGV case, for the bottom modulus $p_0 = q_0$, we do not apply either the key switching or the modulus switching afterwards. Thus, to ensure correct decryption, we require that $\|\nu\|^{\mathsf{can}} < q_0/2$. Instead, in the case of distributed decryption, we have

$$q_0 > 2 \cdot (1 + N \cdot 2^{\mathsf{DD}_{\mathsf{sec}}}) \cdot \|\nu\|^{\mathsf{can}}. \tag{4}$$

See [7] for more details.

Dishonest Encryption: In the BGV scheme, as shown in [49], the noise after a fresh encryption is bounded by

$$\|[c_0 + c_1 \cdot s]_{q_\ell}\|^{\mathsf{can}} \leq D\sqrt{n \cdot V_{m + te_{sk}u + e_1 s + e_0}}.$$

However, using the ZKPoK protocol, we are only able to guarantee that

$$\|2\sum_{i=1}^{N} m_i\|_\infty \leq N \cdot 2^{(\mathsf{ZK}_{\mathsf{sec}}+2)} \cdot t/2, \text{ and}$$

$$\|2\sum_{i=1}^{N} e_{j,i}\|_\infty \leq N \cdot 2^{(\mathsf{ZK}_{\mathsf{sec}}+2)} \cdot \rho_j$$

where $\rho_{sk} = 1$ and $\rho_0 = \rho_1 = 21$ [7]. Thus,

$$\|c_0 - c_1 \cdot s\|^{\mathsf{can}} \leq \sum_{i=1}^{N} \|2 \cdot m_i\|^{\mathsf{can}} + t \cdot (\|2 \cdot e_{sk,i}\|^{\mathsf{can}}\|u\|^{\mathsf{can}} + \|2 \cdot e_{1,i}\|^{\mathsf{can}} \cdot \|s\|^{\mathsf{can}} + \|2 \cdot e_{0,i}\|^{\mathsf{can}}).$$

Since $\|a\|^{\mathsf{can}} \leq \mathsf{n} \cdot \|a\|_\infty$, we have

$$\begin{aligned}\|c_0 - c_1 \cdot s\|^{\mathsf{can}} \leq\ & t \cdot \mathsf{n} \cdot N \cdot 2^{(\mathsf{ZK}_{\mathsf{sec}}+1)} \\ &+ D \cdot t \cdot \mathsf{n} \cdot N \cdot 2^{\mathsf{ZK}_{\mathsf{sec}}+2} \cdot \rho_{sk}\sqrt{\mathsf{n}V_u} \\ &+ D \cdot t \cdot \mathsf{n} \cdot N \cdot 2^{\mathsf{ZK}_{\mathsf{sec}}+2} \cdot \rho_1\sqrt{\mathsf{n}V_s} \\ &+ t \cdot \mathsf{n} \cdot N \cdot 2^{\mathsf{ZK}_{\mathsf{sec}}+2} \cdot \rho_0.\end{aligned}$$

Because $V_u = V_s = 1/2$, we set

$$\mathsf{B}^{\mathsf{dishonest}}_{\mathsf{clean}} \approx t \cdot \mathsf{n} \cdot N \cdot 2^{(\mathsf{ZK}_{\mathsf{sec}}+2)}(21 + 11 \cdot D\sqrt{2n}). \tag{5}$$

3.1.3. Noise analysis of matrix triple generation. To compute a valid and secure set of BGV parameters for the multiplication of matrices A and B, we analyze each stage of the Π_{Prep} protocol, focusing on the operations outlined by Chen et al. [17, Fig. 1]. Note that we instantiate matrix multiplication in [17] using the protocol from [48]. For consistency, we adopt the same notation as Chen et al. [17].

Top Modulus p_3: In Π_{Prep} protocol, the error of $\mathbf{c}_{A_{jk}}$ (and $\mathbf{c}_{B_{jk}}$) is bounded by $\mathsf{B}^{\mathsf{dishonest}}_{\mathsf{clean}}$ (Equation 5). Before performing the matrix multiplication of matrices $\mathbf{c}_{A_{jk}}$ and $\mathbf{c}_{B_{jk}}$, we reduce the error down to a threshold B, using the modulus switching procedure. Thus, the error is scaled by p_3 and the additional *modulus-switching error* $\mathsf{B}_{\mathsf{scale}}$ (Equation 2) is added. Hence, we have

$$\frac{\mathsf{B}^{\mathsf{dishonest}}_{\mathsf{clean}}}{p_3} + \mathsf{B}_{\mathsf{scale}} < \mathsf{B} \Longrightarrow p_3 > \frac{\mathsf{B}^{\mathsf{dishonest}}_{\mathsf{clean}}}{\mathsf{B} - \mathsf{B}_{\mathsf{scale}}}.$$

Middle Modulus p_2: During the FHE-based matrix multiplication, parties compute

$$\mathbf{c}_A \circledast \mathbf{c}_B = \sum_{\kappa=0}^{d-1} (\phi^\kappa(\mathbf{c}_A) \boxtimes \psi^\kappa(\mathbf{c}_B)),$$

where $\boxtimes$ denotes homomorphic multiplication of two ciphertexts. As detailed in [48, Fig. 6], the computation of $\phi^\kappa(\mathbf{c}_A)$ involves addition of two *special* ciphertexts. These special ciphertexts are constructed by first rotating $\mathbf{c}_A$ by κ positions, followed by multiplying the result of any rotation by a scalar, where $1 \leq \kappa \leq d$. Also, rotation operation involves key-switching procedure as well. On the other hand, $\psi^\kappa(\mathbf{c}_B)$ is obtained via simple rotation by κ positions.

While the rotations themselves do not influence the noise directly, switching the key back to the original adds key switching noise v_{ks}, which depends on the key switching method [49]. In our case, we have

$$v_{\mathsf{ks}} = \frac{\sqrt{3\omega} \cdot \max(\tilde{q}_j)}{P} \cdot \mathsf{B}_{\mathsf{ks}} + \sqrt{k} \cdot \mathsf{B}_{\mathsf{scale}},$$

where $\mathsf{B}_{\mathsf{scale}}$ and B_{ks} are as in Equations 2 and 3. Thus, if B is the starting noise of $\mathbf{c}_A$ and $\mathbf{c}_B$, then after τ rotations, the bounded error of each ciphertext grows from B to $\mathsf{B} + \tau \cdot v_{\mathsf{ks}}$.

The next step for $\phi^\kappa(\mathbf{c}_A)$ involves a ciphertext-scalar multiplication, increasing the error to $\mathsf{B}_{\mathsf{const}} \cdot (\mathsf{B} + \tau \cdot v_{\mathsf{ks}})$. Similarly, the subsequent addition with a similar ciphertext doubles the error. A modulus switching is performed next

to reduce the noise magnitude down to B. Hence, the noise at this stage (from $\phi^{\kappa}(\mathbf{c}_A)$) is at most

$$\frac{2\mathsf{B}_{\mathsf{const}} \cdot (\mathsf{B} + \tau \cdot (\frac{\sqrt{3\omega} \cdot \max(\tilde{q}_j)}{P} \cdot \mathsf{B}_{\mathsf{ks}} + \sqrt{k} \cdot \mathsf{B}_{\mathsf{scale}}))}{p_2} + \mathsf{B}_{\mathsf{scale}}.$$

Since we want p_2 to be as small as possible, we use a larger B and P such that,

$$p_2 > 2\mathsf{B}_{\mathsf{const}} \cdot (\mathsf{B} + \tau \cdot \sqrt{k} \cdot \mathsf{B}_{\mathsf{scale}})/(\mathsf{B} - \mathsf{B}_{\mathsf{scale}}).$$

Thus, we set $\mathsf{B} \approx \alpha \cdot \mathsf{B}_{\mathsf{scale}}$, for $\alpha \geq 2$, and we have $p_2 \approx 2\mathsf{B}_{\mathsf{const}} \cdot (\alpha + \tau \cdot \sqrt{k})/(\alpha - 1)$.

Middle Modulus p_1: Note that the noise magnitude grows from B to $d \cdot \mathsf{B}^2$ while computing $\sum_{\kappa=0}^{d-1}(\phi^{\kappa}(\mathbf{c}_A) \boxtimes \psi^{\kappa}(\mathbf{c}_B))$. Since the product of two ciphertexts results in a 3-dimensional vector, key-switching is necessary. Moreover, we want the noise to be reduced to B, and thus require the modulus switching procedure as well. In FANNG, we employ hybrid key switching that allows for merging key switching with the modulus switching, enabling a direct switch to a smaller modulus, i.e., from $Q_1 = P \cdot q_1$ to q_0, decreasing the noise by $q_0/Q_1 = 1/(P \cdot p_1)$. Also, the error before scaling down using the modulus switching is

$$\nu' = d \cdot \mathsf{B}^2 \cdot P + \sqrt{2\omega} \cdot \max(\tilde{q}_j) \cdot \mathsf{B}_{\mathsf{ks}}.$$

Thus, the error after the modulus switching procedure is bounded by $\frac{1}{P \cdot p_1} \cdot \nu' + \sqrt{k+1} \cdot \mathsf{B}_{\mathsf{scale}}$ [49, Sec. 3.2]. Since we want to reduce the noise back to B, we set

$$\frac{d \cdot \mathsf{B}^2}{p_1} + \frac{\sqrt{2\omega} \cdot \max(\tilde{q}_j)}{P \cdot p_1} \cdot \mathsf{B}_{\mathsf{ks}} + \sqrt{k+1} \cdot \mathsf{B}_{\mathsf{scale}} < \mathsf{B}. \quad (6)$$

For large values of P, Equation 6 becomes $\frac{d \cdot \mathsf{B}^2}{p_1} + \sqrt{k+1} \cdot \mathsf{B}_{\mathsf{scale}} < \mathsf{B}$, which (in the variable B) must have a positive discriminant. Namely $p_1 > 4d \cdot \sqrt{k+1} \cdot \mathsf{B}_{\mathsf{scale}}$, and thus, we can set $P \approx 10 \cdot \sqrt{3\omega} \cdot \max(\tilde{q}_j) \cdot (\mathsf{B}_{\mathsf{ks}}/\mathsf{B}_{\mathsf{scale}})$.

Bottom Modulus p_0: Note that modulus reduction is not required at level zero (cf. [48, Fig. 6]). However, since we execute $\mathsf{AddMacs}(\mathbf{c})$, the error grows from B to B^2. To ensure a correct distributed decryption, we require that the noise bounded by B^2 is smaller than $2 \cdot (1 + N \cdot 2^{\mathsf{DD}_{\mathsf{sec}}}) \cdot \mathsf{B}^2$ (cf. Equation (4)). Thus, $p_0 > 2 \cdot (1 + N \cdot 2^{\mathsf{DD}_{\mathsf{sec}}}) \cdot (\alpha \cdot \mathsf{B}_{\mathsf{scale}})^2$.

All the parameters together: Since we require q to be as small as possible, we set $\alpha = 2$. Moreover, we have only 2 parties in our case ($N = 2$) and $\tau \approx d$. Thus, we have:

$$p_0 \approx 2^{\mathsf{DD}_{\mathsf{sec}}+4} \cdot \mathsf{B}_{\mathsf{scale}}^2, \qquad p_1 \approx 4 \cdot d \cdot \sqrt{k} \cdot \mathsf{B}_{\mathsf{scale}},$$
$$p_2 \approx 2 \cdot d \cdot \sqrt{k} \cdot \mathsf{B}_{\mathsf{const}}, \qquad p_3 \approx \mathsf{B}_{\mathsf{clean}}^{\mathsf{dishonest}}/\mathsf{B}_{\mathsf{scale}}.$$

Finally, setting our parameters as $t = 2^{128}$, $d = 64$, $w = 3$, $k = 5$ and $\mathsf{DD}_{\mathsf{sec}} = \mathsf{ZK}_{\mathsf{sec}} = 80$, we have $\log q \approx 760$ and security level $\lambda = 128$.

3.2. Converter Unit

This section describes FANNG's mechanism for transferring the data produced by a set of $\mathsf{N}_{\mathcal{D}}$ dealers, denoted by the set $\mathcal{D}$ (typically with $\mathsf{N}_{\mathcal{D}} = |\mathcal{D}| > 2$) to the two parties running the private inference: i) the client C; and ii) the model owner M. Given that multiple clients engage in private inference on the ML model owned by M, and the identity of M is known beforehand, in contrast to the clients, we leverage this aspect in our design. When necessary to differentiate between various clients in a set $\mathcal{C}$ with size $\mathsf{N}_{\mathcal{C}}$, we denote them as $\{\mathsf{C}_j\}_{j \in [\mathsf{N}_{\mathcal{C}}]}$. Also, κ denotes the computational security parameter.

General re-sharing strategy: Consider a value $x \in \mathbb{F}_p$ additively secret-shared among the dealers $\{\mathsf{D}_i\}_{i \in [\mathsf{N}_{\mathcal{D}}]}$, i.e. $x = \sum_{i \in [\mathsf{N}_{\mathcal{D}}]} x_i$ with D_i holding x_i. They aim to redistribute this value to a fresh sharing among the parties $\{\mathsf{M}, \mathsf{C}_j\}$. The naive method for achieving this involves each dealer D_i generating a 2-out-of-2 sharing of x_i and sending one share to each of M and C_j. This approach incurs a cost of $\mathsf{N}_{\mathcal{D}} \cdot 2 \cdot p$ bits of communication, where p is the bit length of x. When $p > \kappa$, we can optimize the cost using a pseudo-random function $F : \{0,1\}^{\kappa} \to \{0,1\}^{p}$, as follows:

1) For $i \in [\mathsf{N}_{\mathcal{D}}]$, dealer D_i does as follows:
   a) Sample random keys $k^{(i)}_{\mathsf{M},\mathsf{C}_j} \leftarrow \{0,1\}^{\kappa}$, for $j \in [\mathsf{N}_{\mathcal{C}}]$, where $\mathsf{N}_{\mathcal{C}}$ is the bound on the number of clients that the model owner M expects.
   b) Send $x^{(i)}_{\mathsf{M},\mathsf{C}_j} = x_i - F(k^{(i)}_{\mathsf{M},\mathsf{C}_j})$ to M, who defines its share of x as $x_{\mathsf{M},\mathsf{C}_j} = \sum_{i \in [\mathsf{N}_{\mathcal{D}}]} x^{(i)}_{\mathsf{M},\mathsf{C}_j}$.
   c) When the specific identity of the client C_j becomes known, send the key $k^{(i)}_{\mathsf{M},\mathsf{C}_j}$ to C_j.
2) C_j defines its share of x as $x_{\mathsf{C}_j,\mathsf{M}} = \sum_{i \in [\mathsf{N}_{\mathcal{D}}]} F(k^{(i)}_{\mathsf{M},\mathsf{C}_j})$.

Note that the output shares satisfy the equation $x = x_{\mathsf{C}_j,\mathsf{M}} + x_{\mathsf{M},\mathsf{C}_j}$. The approach above is described for only one value for ease of presentation, but the same key can be used for all the values re-shared to the same client. In terms of communication, this approach incurs a total of $\mathsf{N}_{\mathcal{D}} \cdot (p + \kappa)$ bits. Notably, a significant portion of the communication ($\mathsf{N}_{\mathcal{D}} \cdot p$) occurs before the specific identity of C_j is known. This improves the online latency of the protocol in scenarios where the size of the combined messages to be communicated (say m bits) significantly outweighs κ, allowing M and C_j to commence the online phase faster. This is advantageous compared to waiting for C_j to receive $\mathsf{N}_{\mathcal{D}}$ messages, each of length m bits, which would take several rounds if the resulting communication exceeds the network bandwidth.

Re-sharing SPDZ values: Currently in FANNG, dealers operate in a dishonest majority setting, using SPDZs Topgear [7] for computation. In this context, every value x is associated with a message authentication code (MAC), denoted as $\mathsf{MAC}(x)$. Additionally, there exists a global MAC key, denoted as Δ, which the dealers collectively agree upon to generate all associated MAC values for a given computation. To simplify, the process of re-sharing a value

from dealers to $\{\mathsf{M}, \mathsf{C}_j\}$ in FANNG involves re-sharing a triple $(x, \mathsf{MAC}(x), \Delta)$. To achieve this, dealers employ the PRF-based approach outlined earlier for each of the three values. This approach is illustrated in Figure 4 below.

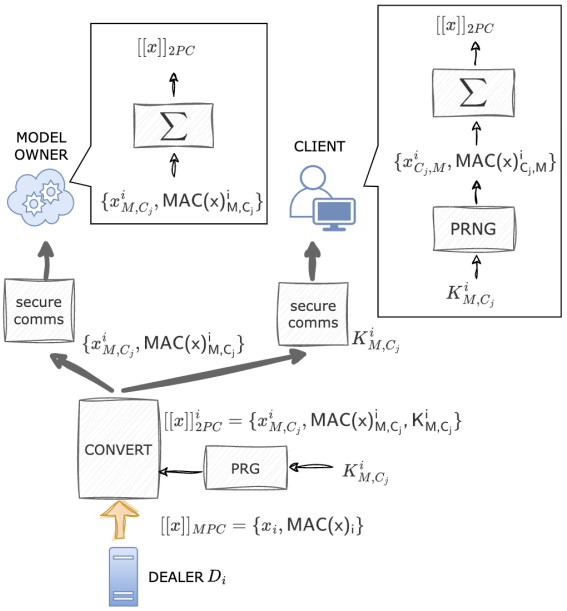

Figure 4: MPC-2PC Converter for SPDZ values in FANNG.

We remark that this approach allows a corrupted dealer in $\mathcal{D}$ to introduce an *additive error* in the re-shared value by sending inconsistent values. Nevertheless, since we use additive secret-sharing, this kind of error remains possible from a corrupted party during reconstruction in any case. However, our incorporation of information-theoretic MACs during the execution of the online phase effectively prevents such errors.

MAC key strategy: Note that the dealers will use different MAC keys based on the trust relations among different clients. Specifically, if a client C_1 lacks trust in another client C_2 to potentially collude with the model owner M, C_1 cannot use the same MAC keys as C_2. Hence, the dealers must take this into account when generating the pre-processing material. This is due to the fact that $\{\mathsf{M}, \mathsf{C}_2\}$ possess a 2-out-of-2 sharing of the MAC key, which will get compromised upon collusion. In the extreme case where no client trusts any other client to be non-colluding with M, the dealers should use a different MAC key for each $\{\mathsf{M}, \mathsf{C}_j\}$ pair.

3.3. Pre-processing Unit

This unit is responsible for performing all input-independent pre-processing tasks that are not carried out by the dealers. We dedicate this section mainly to provide details regarding moving the garbling operation in SCALE-MAMBA to the pre-processing stage, which significantly enhanced the online performance of the framework. We begin with the details of offline garbling.

3.3.1. Offline Garbling. For the distributed computation of garbling circuits (GCs) among n parties, SCALE-MAMBA incorporates the garbling schemes proposed by Hazay et al. [27] and Wang et al. [63]. Furthermore, it utilizes techniques from Zaphod [5] to facilitate interaction between Arithmetic (Full Threshold) and Boolean Circuits (Distributed GCs) through field conversion. Notably, the entire garbling procedure in SCALE-MAMBA occurs online, signifying that circuit garbling takes place just before the evaluation of the garbled circuit, in parallel with the online execution. This way, SCALE-MAMBA guarantees reactivity, enabling parties to decide on the protocol's progression based on intermediate and public values.

The computational model based on the disassociation/independence of the offline and online phases is difficult to materialize in general-purpose frameworks like SCALE-MAMBA and MP-SPDZ, especially if there is no previous knowledge of the function. Moreover, there are several technical aspects that limit the adaptation of offline garbling in SCALE-MAMBA, especially when considering active security:

1. From a cryptographic perspective, the inputs from the online phase require masking with authenticated bits (aBits) generated via a chain of different Oblivious Transfer (OT) protocols that start from a set of *choicebits* selected by each party (cf. [31] for details). The circuits themselves depend on similar processes to generate the keys that are embedded in each circuit. In that sense, both need to come from the same set of choicebits.
2. From an engineering point of view, current versions of SCALE-MAMBA are not built to handle circuits offline. More specifically, there is no trivial way to parameterize the type and quantity of circuits needed online, as we can do with the triples (via the `-max` flag). Additionally, for architectural reasons, there are no mechanisms on SCALE-MAMBA that would allow us to trivially interact with them.

To decouple the garbling procedure from the online phase, FANNG introduces a set of new instructions capable of Garbling/Storage, Loading and Executing Gabled Circuits for the dishonest majority setting:

- OGC `(circuit_id, amount)`: Garbles a given `amount` of the specified circuit and stores the result to a database/file system.
- LOADGC `(circuit_id, amount)`: Loads specific `amount` of persisted `circuit_id` instances to memory.
- EGC `(circuit_id)`: Executes the next specific circuit on the pile, using the same interaction method as with GC, the instruction for garbling in SCALE-MAMBA.

To garble offline, the parties need to know the type and amount of GCs they require beforehand. They can produce, store and load these circuits with the instructions above. FANNG also includes architectural components to support Database/File System connectivity for uploading a specified number of circuits of a given type. It is worth highlighting that GCs are processed offline using a specific set of *choicebits*. We then verify that the choicebits held by each client/online party for the circuits match with the ones

being used to generate authenticated bits for the masks. In the case of ML inference, GCs are generally used for private comparisons required in activation and pooling layers, but FANNG can handle any type of circuit.

Garbling Offline Data Flow: Though FANNG treats the garbling functionality of SCALE-MAMBA as a black box, moving the garbling process to offline requires manipulating the process necessary to create the keys embedded in the circuits. For instance, if a protocol requires aBits, daBits, and GCs altogether (for instance, in Aly et al. [4]), then the keys corresponding to all these materials should be generated from the same set of choicebits, as implemented in SCALE-MAMBA. Furthermore, in SCALE-MAMBA, a fresh set of choicebits is selected at every run. FANNG makes it possible to parameterize the choicebits, as illustrated in Figure 5. This slight change in the flow above (replacing the fresh selection of choicebits), implies the persistence of the choicebits used to garble the circuits and the way they are consumed by the framework.

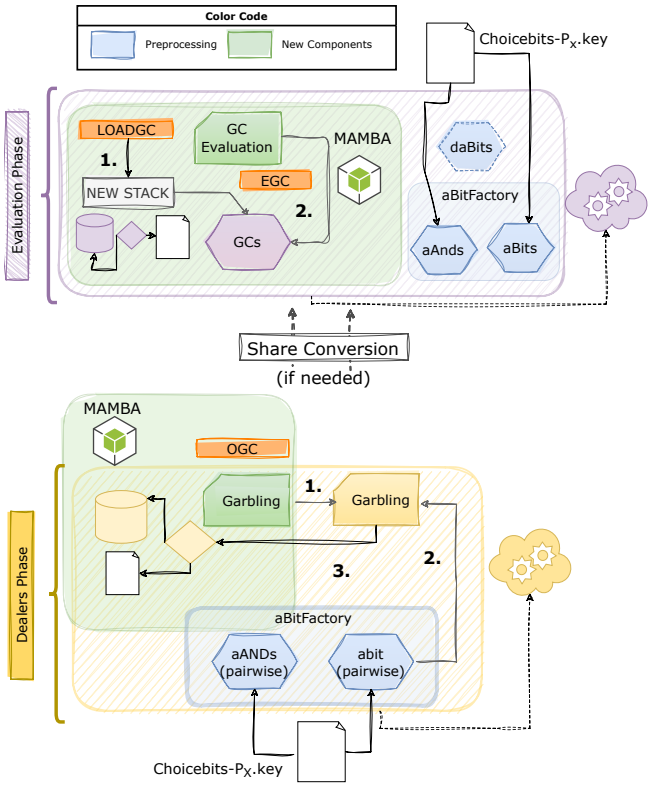

Figure 5: Offline Garbling in FANNG.

Note that FANNG currently supports offline garbling only by computing parties, not dealers. We provide further insights to offload this task to the dealers in §A. Additionally, for larger circuits, circuit sizes can be notably high (e.g. ≈ 10MB per party for float multiplication). As a result, the execution times of instructions like LOADGC heavily depend on the File System reading speed or Database response time. Therefore, we decided to design our DB support in a modular fashion, decoupled from the old I/O support in SCALE-MAMBA. New features introduced by FANNG for databases and file systems are specifically integrated into our modular data connectivity support.

Experimentation and Discussion: We conducted experiments in this section using our *General Purpose MPC* testbed. We evaluated performance in a two-party scenario using the basic less-than-or-equal (LTZEQ) circuit [43], with a size of ≈ 800KB, running in batches of up to 1000 circuits. As shown in Table 2, we obtain improvements for online runtime in the range 2.4×-4.2× .

TABLE 2: Offline Garbling Timings (seconds) per 1000 operations.

Stage	Offline Garbling			Online Garbling		
	Local	Ping	Wan	Local	Ping	WAN
offline						
Garbling:	2.91	4.7	86	==	==	==
Storage:	17.86	17.86	17.86	==	==	==
Loading:	14.88	14.88	14.88	==	==	==
online						
Execution	0.97	2.1	62.6	3.89	6.83	148.66

In reference to Table 2, we also explore FANNG's upcoming support for a dealer's model. In this scenario, dealers operate in a LAN or faster data centres, such as point-to-point connections, in contrast to the online environment like cloud service providers. Consequently, the more probable deployment scenario involves an offline phase with ping distance cost (roughly $3-5\times$ the cost of local computations) and an online phase over WAN, assuming an average ping time between cloud providers. In this setup, online garbling takes 148.66 seconds, while our offline garbling requires 37.45 seconds in the offline phase and 62 seconds in the online phase. This results in a $2.4\times$ improvement in online runtime, with a reduction in an overall runtime.

Finally, regarding Loading and Storage times, while they can be amortized, they ultimately depend on the underlying user's DB engine. We use a `MySQL 8.0` Vanilla Dockerized installation. The timings in both cases are linked to the speed at which we can process two classic SQL commands, namely INSERT and SELECT. Advanced use cases may involve modern DB setups, such as in-memory DBs and other Big Data processing engines.

3.4. Storage Support

I/O is crucial for any modern application requiring persistence. Furthermore, real-life implementations demand flexibility in handling secret-shared/public inputs. While SCALE-MAMBA initially provided I/O capabilities, they were limited to console access, necessitating manual code intervention for file system support [3]. FANNG addresses this requirement in a modular and adaptable way, via a novel Controller-Based IO. Furthermore, we introduced backward-compatible extensions to the existing I/O, incorporating *under the hood* flexible and maintainable components. This ensures current SCALE-MAMBA users can continue relying on the framework's I/O, while providing future users with the option to leverage our novel storage support.

Legacy I/O: SCALE-MAMBA I/O consists of two sections: i) instructions for private and public I/O invoked via MAMBA code, and ii) interfaces implementing I/O functionality directly in SCALE, with these interfaces being programmatically exchanged in the code. FANNG retains the instructions but broadens the set of interfaces, facilitating connections to file systems (FS) or databases (DBs) for storage (e.g., pre-processed materials generated by the framework) and loading during the online phase.

Controller-Based I/O: FANNG adds a parametrizable I/O controller, customizable through a configuration file. Users can configure their instances to utilize any available I/O mediums via the controller. Currently, the framework supports two options: general FS and MySQL. This eliminates the need for users to recompile the framework for I/O changes. All database connections, including those used by the revamped Legacy I/O, rely on the controller. This centralized configuration enables users to manage the database settings in a single text file, regardless of the chosen I/O model. The Controller-Based I/O is employed across all new functionalities dedicated to producing or retrieving preprocessed material.

Currently, FANNG supports only MySQL 8.0. However, it is designed with flexibility, allowing seamless extension of both legacy and new I/O functionalities to other DB engines like SQLite and PostgreSQL. We have have included some configuration examples in Appendix B.

3.5. Evaluation Engine

This section outlines FANNG's contributions to an efficient online phase of protocols for privately evaluating NNs. FANNG introduces a novel way of integrating private comparison protocol with a state-of-the-art probabilistic truncation protocol to reduce communication rounds. Furthermore, FANNG introduces several libraries for implementing ML blocks, facilitating convolutional, fully connected and folding layers, among others.

3.5.1. Protocols for private comparisons. Activation functions are fundamental in NN models, with the Rectified Linear Unit (ReLU) being one of the most popular. Essentially, ReLU can be implemented through comparison and multiplication. FANNG integrates practical privacy-preserving comparisons, leveraging contributions from the current SoTA [4]. This involves applying mixed circuits to constructions introduced by Catrina and De Hoogh [13], and Rabbit [43]. Our library includes an interface (rabbit_sint), returning the following for any $\langle x \rangle$, the secret-shares of $x \in \mathbb{Z}_{2^k}$ over prime-order field $\mathbb{F}_p$:

$$\mathsf{LTZ}(\langle x \rangle) : \mathbb{Z}_p \to \{0,1\} = \begin{cases} 1 & \text{if } x < 0, \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

The interface invokes constructions implemented as described in [4], performing boolean operations via Zaphod [5]. It can be parameterized with the following evaluation options:

- rabbit_slack: Integrates Zaphod [5] with the logic from Rabbit [43], providing statistical security for accelerated computing.
- rabbit_list: Implements the original Rabbit [43] with a rejection list.
- rabbit_fp: Similar to rabbit_list but assumes a bounded prime-order domain close to a power of 2.
- rabbit_conv: Utilizes the conversion circuits from SCALE-MAMBA to transform $\langle x \rangle$ on $\mathbb{F}_p$ to $\mathbb{Z}_{2^k}$.
- rabbit_less_than: Takes 2 modulo $\mathbb{Z}_{2^k}$ inputs and directly evaluates the binary circuit from Rabbit [43] using logical gates from SCALE-MAMBA.
- dabits_ltz: Instantiates the main contribution from [4], combining the original Catrina and de Hoogh [13] construction with Boolean evaluation from Zaphod using daBits for random bit sampling.

The library is integrated into several NN modules within the FANNG compiler, including relu_lib.py, which implements several variations of ReLU's . According to [4], dabits_ltz remains the fastest operation mode in the library, particularly when garbling is conducted offline, a capability supported by FANNG. Our results for private comparisons are presented in Table 3.

TABLE 3: Private comparisons using rabbit_lib in a 2 party setting. Time is measured in seconds per 1000 operations (*Online Garbling; †Offline Garbling).

Mode	Local	Ping	WAN
rabbit_slack	6.41	13.46	295.76
rabbit_list	120	457	33,365
rabbit_fp	64	138	18,315
rabbit_conv	12.8	19	300
rabbit_less_than	11.81	60.50	5,225
Default Mode			
dabits_ltz*	3.89	8.43	148.66
dabits_ltz†	**0.97**	**2.07**	**62.52**

The library also includes two VHDL versions of the underlying Boolean circuits, detailed in [4]. The difference lies in the presence or absence of XOR gates, with the former incurring an increase in the number of gates. While the former allows us to benefit from typical garbled circuit optimizations, the latter provides more parallelism and can be advantageous when employing replicated secret sharing for Boolean evaluation.

3.5.2. Combining ReLU's with n Truncations. FANNG supports Fixed Point Arithmetic (FPA) in the same way as SCALE-MAMBA and MP-SPDZ, specifically implementing the approach outlined in [14]. The challenge of truncation in fixed point arithmetic is a widespread issue in ML applications (see for instance [6]). This section elaborates on the combination of a predetermined number of truncation operations with a comparison operation.

In FPA, for a fixed point value x represented with mantissa α_x and precision d, we have $x = \alpha_x \cdot 2^d$. Then, the multiplication of two fixed-point values x and y produces

the following:

$$x \cdot y = \alpha_x \cdot \alpha_y \cdot 2^{2 \cdot d}. \quad (8)$$

To prevent the scaling factor from quickly saturating the domain space, it is necessary to truncate it by a factor of 2^d after each multiplication. The probabilistic truncation mechanism proposed in [14] is commonly used in the literature. It involves a single round of communication and utilizes pre-processed authenticated material, hereafter referred to as *truncation masks*.

In recent works (e.g., [53]), it was noted that ReLU and truncation can be effectively combined in a single call. The truncation protocol bears some similarities with the comparison protocol in [13] and its extensions in [4]. Both protocols involve random sampling in the pre-processing phase and necessitate a much smaller word space compared to the domain size. Leveraging these observations, we present a protocol that concurrently performs truncation and the comparison protocol from [4]. Notably, this truncation addresses scaling issues arising from multiple multiplications. This enables the batching of truncations from multiple multiplications in a single execution, allowing parallel processing with private comparisons in the activation layers.

Figure 6 illustrates parallel execution of comparison and truncation of value c resulted after n sequential multiplications, with k as the domain size and κ as the security parameter [4]. Pre-processing involves using contributions from [20] for authenticated bits (aBits) and Zaphod [5] for daBits. The main idea is that random sampling for truncation and comparison occurs in the same domain and over the same input. Therefore, we can use masked outputs from the comparison directly for truncation. The mask is utilized once for executing both protocols. Following this, we multiply the truncated value with the output bit from the comparison, similar to a traditional ReLU implementation. Notably, the generation of the 2^k secret shared bounded randomness [20] takes place during pre-processing in FANNG. On the other hand, SCALE-MAMBA does not separate this process from the online phase, resulting in a slower truncation process.

Concerning the number of batched truncation operations, the size of mantissa (v) plus $n \cdot d$ (representing the quantity of performed multiplications) is restricted by the value of k. In practice, FANNG and SCALE-MAMBA assume a word size of 64 bits. This upper bound be attributed to the limitations of the GC processor, which was designed to support 64-bit words. This limitation can be viewed as a trade-off between precision and the size of n or batched truncations.

Experimentation and Discussion: Consider a **CNN** setup where the network involves multiplication in the convolution layer, followed by another in the batch normalization layer. With quantization during normalization, if no truncation is applied, the precision expands to $3 \cdot d$ bits. In our experiments, we fixed d at 20 and restricted the represented values to not exceed 8 (3 bits), ensuring compatibility with 64-bit words in our experimentation.

To establish a comparable baseline, we provide equivalent running times and the number of instructions generated using the classical fixed-point truncation [14] and comparison [13] protocols. Given that the use of circuit optimizers results in a substantial increase in the number of instructions for the baseline, we also refrain from employing them in our evaluations. In our setup, we utilized our *General Purpose MPC* testbed and evaluated results across Local, Ping, and WAN setups (cf. §3). We distinguish between two types of executions: with and without vectorized inputs. Table 4 showcases the performance of the `trunc_ltz` operation, representing the evaluation of a comparison operation along with either one (`1-T`) or two (`2-T`) truncations.

TABLE 4: Performance of comparison in conjunction with truncation (`trunc_ltz`) in seconds per 1000 operations. (†Vectorized).

Mode	Optimized			Non-Optimized		
	Local	Ping	WAN	Local	Ping	WAN
Full Threshold (2p)						
[13] + 1-T	1.61	2.71	114	4.66	18.1	793
[13] + 2-T	1.68	3.15	135	4.69	19	**814**
trunc_ltz(1-T)	3.35	3.3	170	==	==	==
trunc_ltz(2-T)	4.3	3.37	190	==	==	==
trunc_ltz(1-T)†	0.84	2.1	62	==	==	==
trunc_ltz(2-T)†	0.86	2.2	**62.7**	==	==	==
Full Threshold (3p)						
[13] + 1-T	2.69	4.46	116	8.4	23.1	798
[13] + 2-T	3.09	5.09	136	8.56	23.2	**820**
trunc_ltz(1-T)	5.3	3.67	265	==	==	==
trunc_ltz(2-T)	8.6	3.77	276	==	==	==
trunc_ltz(1-T)†	1.34	2.44	103	==	==	==
trunc_ltz(2-T)†	1.35	2.9	**103.8**	==	==	==
Shamir (3p)						
[13] + 1-T	1.37	3.93	115	5.52	17.02	792
[13] + 2-T	1.61	4.48	136	6.02	17.07	814
trunc_ltz(1-T)	1.27	4.89	339	==	==	==
trunc_ltz(2-T)	1.5	5.13	349	==	==	==
trunc_ltz(1-T)†	1.23	4.1	306	==	==	==
trunc_ltz(2-T)†	1.25	4.2	308	==	==	==

We observe that vectorization is crucial for optimizing TCP-based communications, as large data structures can be segmented into TCP packets, maximizing the allowed packet size. This explains the performance degradation in the non-vectorized case when ping time increases. Additionally, as pointed out by Aly et al. [4], the network size somewhat constrains the effectiveness of circuit optimizers. Therefore, any practical scenario utilizing SCALE-MAMBA has to depend on non-optimized circuits, leading to limited performance as indicated by the times in **red**.

FANNG addresses this issue by providing a vectorized version of `trunc_ltz`, and the best timings are annotated in **green**. The results demonstrate that vectorization significantly improves both latency and throughput. For instance, our protocols can perform 1000 comparisons $13\times$ faster in the two-party full threshold (FT) setting over a WAN setup. In this context, we have also quantified the number of instructions generated by the compiler in Table 5.

Compared to SCALE-MAMBA, we've significantly reduced the total number of instructions for compilation. Besides vectorization, this reduction is attributed to a series of optimizations implemented for generating aBits, batched truncation, and our `dabits_ltz` protocol. This reduction benefits RAM usage and compilation time, especially in average/large neural networks.

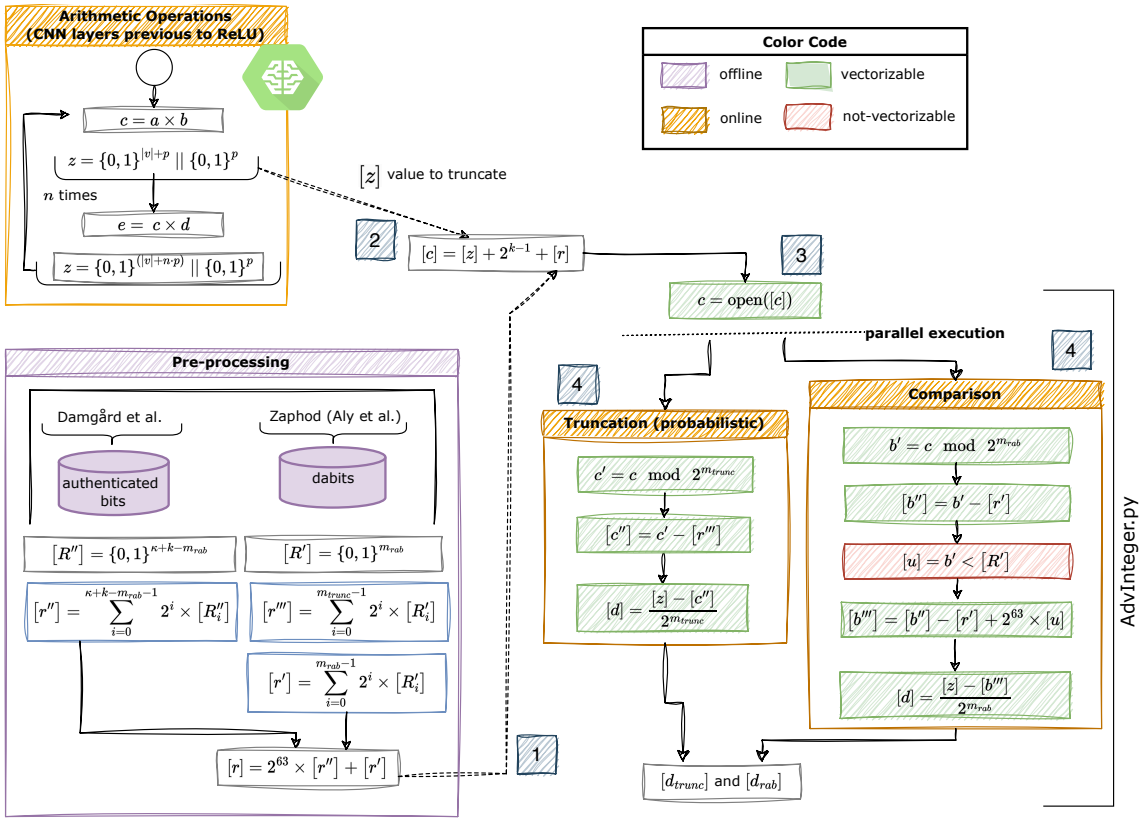

Figure 6: Protocol for batching secure truncation in parallel with private comparison.

TABLE 5: Number of instructions per ReLU Mode (†Vectorized).

Mode	Instructions	
	1-T	2-T
[13] + [14]	2,638,000	3,028,000
`trunc_ltz`	480,000	482,000
`trunc_ltz`†	76,000	76,000

There is a performance gap between the evaluation of `trunc_ltz`, when executed on Shamir or FT. This difference arises from our reliance on offline garbling for FT, which is not possible on Shamir due to the absence of garbling in the evaluation of the Boolean circuit [5]. Lastly, transitioning from 1 to 2 batched truncations has a minimal impact on performance across all setups, a significant improvement considering the typical impact of truncation execution on quantized networks [34].

3.5.3. ML libraries. The framework includes specialized libraries to perform efficient linear transformations, specifically in convolutional and fully connected layers. FANNG also encompasses libraries for folding (average and max pooling), normalization and standardization, and output layers (e.g., softmax). The main contribution of these new libraries lies in their integration of FANNG's novel functionalities, such as matrix and convolutional triples, preprocessing of truncation masks, and novel private comparison protocols. Moreover, these libraries are implemented with optimal communication rounds, eliminating the need for an optimizer during compilation.

3.6. Handling pre-processed materials

FANNG introduces a set of new instructions designed to handle the generation and loading of pre-processed materials for use in the online phase. In the following, we discuss details of the most relevant ones.

Bounded Randomness: Protocols, such as comparisons, are based on statistical security, and rely on masking bounded by some power of 2, say 2^k. `SCALE-MAMBA` generates masks programmatically, and for instance, a 40-bit mask necessitates 148 instruction lines per invocation in the bytecode. In applications such as machine learning, a substantial number of these masks are required, significantly affecting the amount of RAM needed for program compilation and loading into memory. To address this, we introduce three new instructions that not only transition the process entirely from the online phase but also reduce the number of instructions per invocation to one, independent of the size of the bound.

- OSRAND `(k, amount)`: Generate and persist any specified `amount` of random masks bounded by 2^k using our DB Support.

- LOADSRAND(k, amount): Load into memory a specified amount of random masks bounded by 2^k.
- SRAND(output, k): Extract a random mask bounded by 2^k from memory and assign it to the register amount. In SCALE-MAMBA language, this vectorizable instruction can fill multiple registers in a single call.

daBits: In order to use the daBits generated in the pre-processing, we introduce two new instructions:

- ODABIT(amount): Persists any specified amount of daBits using the pre-existing daBits factory.
- LOADDABIT(amount): Loads a specified amount of daBits into memory. Note that the *choicebits* used for generating the loaded daBits must be the same as those used for the Garbled Circuits and for FANNG execution.

We also modified existing DABIT instruction to now retrieve the next available daBits from memory. This *vectorizable* instruction can return multiple daBits if necessary. In cases where none are available, FANNG will resort to the daBits factory, a principle applicable to all instructions in FANNG that consume pre-process material.

Matrix Triples: To enable the client and model owner to utilize the specialized convolutional and matrix triples [17], [48], generated by the dealers, during the online phase, we need a mechanism to load them into memory once persisted. FANNG introduces two new instructions for this purpose.

- LOADCT(type_id, amount): Loads any amount of Matrix Triple of specified type_id into memory.
- CT_DYN(A,B,C, type_id): Extracts a Matrix Triple of specified type_id from memory and assigns it to the vectorized registers A, B, C.

Test Modes for Pre-processed Material: FANNG focuses on a Dealer/pre-processing model, offering users practical implementations of both. However, for development and simulation purposes, users may prefer to simulate these models. Following SCALE-MAMBA practices, we offer various testing modes in the config.h file. For example, the test mode for bounded randomness, daBits and GCs can be activated or deactivated as shown in Listing 1.

```
/* Ignores shares in memory for SRAND when
    set to 1. Used for testing to avoid
    consuming shares in DB. */
#define return_shares_zero 1

/* Ignores shares in memory for dabits.
    When set to 1, returns only 0's. */
#define return_dabit_zero 1

/* Ignores share counters for LOADCT,
 * LOADSRAND, and LOADGC when set to 1.
 */
#define ignore_share_db_count 1
```

Listing 1: Test Mode configurations in FANNG.

3.7. General Optimizations

In addition to the mentioned features, FANNG prioritizes usability by minimizing compiler-generated instructions, introducing novel functionality, and enhancing the online phase inherited from SCALE-MAMBA. We explore some of the most relevant items below:

- **Summation Instructions SUMS and SUMC**: Machine operations relying on matrices often involve constant summations. While local users can perform these operations, SCALE-MAMBA's implementation involves instructions that can be significant for a large set of values $\{x_i, \cdots, x_n\}$. This can lead to slower compilation times and increased RAM consumption. To mitigate this, we introduced two novel instructions, SUMS and SUMC, providing outputs of $\sum_{i=1}^{n} x_i$ for private and public inputs, respectively. These instructions simplify development and reduce the entropy of the instruction file, accelerating development time.
- **Communication Bottlenecks**: We observed a limitation in SCALE-MAMBA related to the number of elements it can send per message, specifically through the instructions opening $\mathbb{F}_p$ inputs (START_OPEN and START_CLOSE). For instance, our experiments showed that it can only support up to 100 thousand elements per invocation, constraining the parallelization of instructions per round. To overcome this limitation, FANNG allows for a configurable number of SSL connections. When a user intends to open multiple elements in a single round, FANNG can navigate around this constraint by utilizing as many connections as configured or provided by the user.
- **Graph Theory Library**: We have extended the functionality of the framework beyond machine learning and towards generic MPC, incorporating a novel Graph Theory library. Currently, this library includes SoTA methods in shortest path [2], with plans for additional expansion in the future.

4. Evaluation

This section provides details regarding private ML inference using our FANNG framework. We chose to evaluate three different neural network (NN) architectures to accommodate varying complexities (see §C for additional architectural details).

- LeNet [40]: 5-layer network with 60K parameters, over MNIST dataset.
- A generic CNN: 8-layer network with 1.5 million parameters, over CIFAR10 dataset.
- VGG16 [58]: 16-layer network with 37 million parameters, over CIFAR10 dataset.

We chose to benchmark only the online phase, as the pre-processing phase is executed separately earlier and stored using storage support. Timings for offline operations are provided in previous sections. FANNG is, to our knowledge, the first framework to fully support private ML inference in the dishonest-majority setting with active security. Our fully open-sourced code will be made publicly available upon publication.

Table 6 provides our results for performing private ML inference using our *Machine Learning* testbed evaluated across Local, Ping, and WAN setups (cf. §3). In these tests, pre-processing was disabled using FANNG's test mode support (cf.§3.6). This included simulating the generation of matrix and convolutional triples, GCs, and truncation masks by activating the relevant testing flags. Timings for generating other pre-processing elements, such as beaver triples, daBits, and singles, were not included in the results.

TABLE 6: Timings for private ML inference using FANNG. Values are reported in seconds.

-	LeNet	CIFAR10 CNN	VGG16
Full-Threshold (2p)			
Local	10	109	534
Ping	20	516	1,159
WAN	410	12,175	18,074
Shamir (3p)			
Local	11	459	979
Ping	48	1,490	2,561
WAN	1,996	60,384	87,632
Full-Threshold (2p) - Without Activation			
Local	0.62	43	440
Ping	0.62	44	472
WAN	0.86	44	483

Although we achieve impressive runtimes for both Local and Ping setups, we note that the runtimes increase significantly over a WAN when transitioning from LeNet to the more complex VGG16. This is attributed to the absence of support for parallelizing Boolean circuit evaluations in SCALE-MAMBA affecting activation layers. Despite FANNG incorporating support for vectorization at various stages in ReLU computation, it relies on the Boolean circuit evaluation inherited as a black box from SCALE-MAMBA for activation functions. To be more specific, the sequential evaluation of garbled circuits in the case of full threshold and boolean circuits for Shamir contributes to this slowdown.

We plan to address this limitation in the future. However, to offer insight into its impact on our performance results, we have also included the evaluation results in Table 6 for the two-party full threshold setting after excluding the activation layers. In this scenario, we observe that the time complexity for Ping and WAN setups is comparable to that of a Local setup. This is reasonable, as the communication rounds for the three networks without activation layers are only 22 (LeNet), 44 (CNN for CIFAR10), and 577 (VGG16).

The performance gap between the full-threshold setting and Shamir is attributed to the absence of parallelization in the activation layer. In SCALE-MAMBA, Shamir utilizes replicated secret sharing over $\mathbb{Z}_2$ for Boolean circuit evaluation, a process inherited by FANNG. In contrast, full-threshold employs [HSS17] [27] for Boolean circuits in both SCALE-MAMBA and FANNG, with the difference that FANNG can push the garbling phase to pre-processing as described in §3.3.1. The private comparison circuit over $\mathbb{Z}_2$ incurs 16 communication rounds, whereas the online phase in [HSS17] requires only 2 rounds. These communication rounds are sequential per ReLU, making full-threshold significantly faster than Shamir for activation layers.

The similar performance of two networks trained on CIFAR10, despite a significant difference in size, can be attributed also to the dominance of activation layers. Specifically, the CIFAR10 CNN has 1.5 million parameters and 196,640 ReLUs, whereas the larger VGG16 has 37 million parameters but only 285,672 ReLUs.

Analytical Evaluation: Due to the absence of support for the parallelization of comparison circuits in FANNG, the current runtime obtained may not accurately represent the achievable performance. Rather, they constitute an overestimation, resulting in timings that are much higher than the potential achievable values, particularly for communication-dominant setups such as WAN. To address this limitation, we conduct a theoretical analysis by focusing on two key parameters: i) *Batch size*, representing the number of comparison circuits that FANNG could parallelize in its anticipated capability, and ii) *Overhead Factor*, indicating the increase in runtime when handling a batch of circuits in parallel compared to a single execution. The results are plotted in Figure 7.

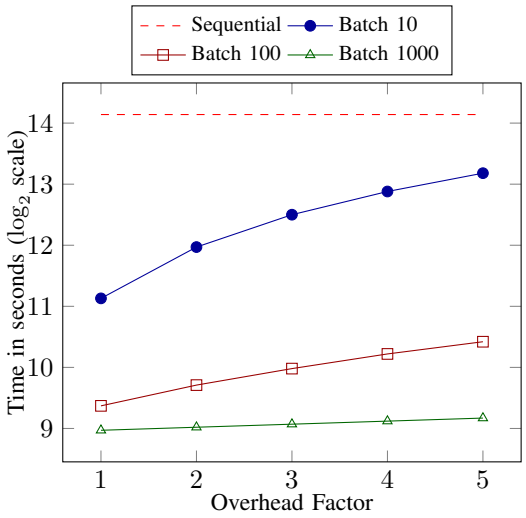

Figure 7: Analytical evaluation of VGG16 inference using FANNG over a WAN setup with batch sizes (Batch) $\in \{10, 100, 1000\}$ and Overhead Factor $\in \{1, 2, 3, 4, 5\}$. 'Sequential' denotes the baseline evaluation without parallelization.

As shown in Figure 7, parallelizing the comparisons within activation layers can substantially enhance the runtime. For example, when the batch size is set to 100 and the overhead factor is set to 4, FANNG will require less than 20 minutes to execute a private inference on the VGG16 network over a WAN setup. This time can be further reduced to less than 10 minutes by using a higher batch size of 1000.

Summary: The results highlight FANNG's remarkable capability to achieve private inference for MNIST within seconds and for CIFAR10 within minutes, considering both Local and Ping times. This performance can be considered as state-of-the-art for Full-Threshold settings, as, to the best

of the authors' knowledge, no other implementation in this setting has provided better metrics. The potential for achieving comparable performance in WAN setups is evident by parallelizing activation layers, a goal set for future work. Furthermore, the evaluation, excluding ReLUs, reveals the potential to classify MNIST in under a second and CIFAR10 in under a minute, emphasizing the need for ongoing efforts to enhance the efficiency of activation layers.

5. Conclusion & Future Work

In this work, we present FANNG-MPC, an MPC framework developed with a focus on private machine learning (ML) inference. FANNG extends the capabilities of the well-established SCALE-MAMBA framework, which supports various actively secure MPC protocols over fields. After identifying limitations in SCALE-MAMBA concerning ML inference, we introduce several innovations. These include dealer support for preprocessing and storage support to streamline the preprocessing phase. Our contributions, both in theory and engineering, are substantiated through a comprehensive evaluation that closely simulates real-world execution rather than a simple prototype. The results demonstrate the practicality of private ML inference within the actively secure setting in MPC with a dishonest majority.

In our future work, FANNG will evolve to provide a more capable pre-processing engine for private ML inference. While FANNG currently features a comprehensive instruction set for storing and loading all pre-processing materials, this functionality has not yet been implemented in the dealers. The current version of FANNG supports only matrix triple generation within the dealers, leaving the generation and storage of the remaining offline elements to be handled directly by the client and model owner.

In upcoming iterations, FANNG is set to extend its dealer support to include garbling, along with the generation of truncation masks, Beaver triples, and authenticated singles. Additionally, FANNG dealers will benefit from hardware acceleration through FPGA support. Moreover, as identified in our evaluations, the lack of parallelization for comparisons in SCALE-MAMBA significantly affects overall performance, and this issue will be addressed in the next iterations of FANNG. These enhancements will collectively fortify FANNG's capabilities in facilitating various aspects of privacy-preserving machine learning.

References

[1] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum Key Exchange - A New Hope," in *USENIX Security Symposium (USENIX Security)*, 2016.

[2] A. Aly and S. Cleemput, "A Fast, Practical and Simple Shortest Path Protocol for Multiparty Computation," in *European Symposium on Research in Computer Security (ESORICS)*, 2022.

[3] A. Aly, K. Cong, D. Cozzo, M. Keller, E. Orsini, D. Rotaru, O. Scherer, P. Scholl, N. P. Smart, T. Tanguy, and T. Wood, "SCALE–MAMBA v1. 14: Documentation," *Documentation. pdf*, 2021, https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf.

[4] A. Aly, K. Nawaz, E. Salazar, and V. Sucasas, "Through the Looking-Glass: Benchmarking Secure Multi-party Computation Comparisons for ReLU 's," in *Cryptology and Network Security (CANS)*, 2022.

[5] A. Aly, E. Orsini, D. Rotaru, N. P. Smart, and T. Wood, "Zaphod: Efficiently Combining LSSS and Garbled Circuits in SCALE," in *Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC@CCS)*, 2019.

[6] W. Ao and V. Boddeti, "AutoFHE: Automated Adaption of CNNs for Efficient Evaluation over FHE," *Cryptology ePrint Archive*, 2023.

[7] C. Baum, D. Cozzo, and N. P. Smart, "Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ," in *Selected Areas in Cryptography (SAC)*, 2020.

[8] B. Biasioli, C. Marcolla, M. Calderini, and J. Mono, "Improving and Automating BFV Parameters Selection: An Average-Case Approach," *Cryptology ePrint Archive*, 2023.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," *ACM Transactions on Computer Theory*, 2014.

[10] Z. Brakerski and V. Vaikuntanathan, "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages," in *Annual International Cryptology Conference (CRYPTO)*, 2011.

[11] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning," *Proceedings of Privacy Enhancing Technologies (PoPETs)*, 2020.

[12] J. Cabrero-Holgueras and S. Pastrana, "SoK: Privacy-Preserving Computation Techniques for Deep Learning," *Proceedings of Privacy Enhancing Technologies (PoPETs)*, 2021.

[13] O. Catrina and S. de Hoogh, "Improved Primitives for Secure Multiparty Integer Computation," in *Security and Cryptography for Networks (SCN)*, 2010.

[14] O. Catrina and A. Saxena, "Secure Computation with Fixed-Point Numbers," in *Financial Cryptography and Data Security (FC)*, 2010.

[15] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction," in *ACM SIGSAC Conference on Cloud Computing Security Workshop (CCSW@CCS)*, 2019.

[16] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning," in *Annual Network and Distributed System Security Symposium (NDSS)*, 2020.

[17] H. Chen, M. Kim, I. Razenshteyn, D. Rotaru, Y. Song, and S. Wagh, "Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning," in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2020.

[18] A. Costache and N. P. Smart, "Which Ring Based Somewhat Homomorphic Encryption Scheme is Best?" in *The Cryptographers' Track at the RSA Conference (CT-RSA)*, 2016.

[19] A. P. K. Dalskov, D. Escudero, and M. Keller, "Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security," in *USENIX Security Symposium (USENIX Security)*, 2021.

[20] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation," in *Theory of Cryptography Conference (TCC)*, 2006.

[21] A. Di Giusto and C. Marcolla, "Breaking the power-of-two barrier: noise estimation for BGV in NTT-friendly rings," *Cryptology ePrint Archive*, 2023.

[22] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2017.

[23] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic Evaluation of the AES Circuit," in *Annual International Cryptology Conference (CRYPTO)*, 2012.

[24] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," in *International Conference on Machine Learning (ICML)*, 2016.

[25] K. Gupta, D. Kumaraswamy, N. Chandran, and D. Gupta, "LLAMA: A Low Latency Math Library for Secure Inference," *Proceedings of Privacy Enhancing Technologies (PoPETs)*, 2022.

[26] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "SoK: General Purpose Compilers for Secure Multi-Party Computation," in *IEEE Symposium on Security and Privacy (IEEE S&P)*, 2019.

[27] C. Hazay, P. Scholl, and E. Soria-Vazquez, "Low Cost Constant Round MPC Combining BMR and Oblivious Transfer," in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2017.

[28] Z. Huang, W. jie Lu, C. Hong, and J. Ding, "Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference," in *USENIX Security Symposium (USENIX Security)*, 2022.

[29] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A Low Latency Framework for Secure Neural Network Inference," in *USENIX Security Symposium (USENIX Security)*, 2018.

[30] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[31] M. Keller, E. Orsini, and P. Scholl, "Actively Secure OT Extension with Optimal Overhead," in *Annual International Cryptology Conference (CRYPTO)*, 2015.

[32] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ Great Again," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2018.

[33] M. Keller, D. Rotaru, N. P. Smart, and T. Wood, "Reducing Communication Channels in MPC," in *Security and Cryptography for Networks (SCN)*, 2018.

[34] M. Keller and K. Sun, "Secure Quantized Training for Deep Learning," in *International Conference on Machine Learning (ICML)*, 2022.

[35] K. Kim and H. C. Tanuwidjaja, *Privacy-Preserving Deep Learning - A Comprehensive Survey*. Springer, 2021.

[36] B. Knott, S. Venkataraman, A. Y. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "CrypTen: Secure Multi-Party Computation Meets Machine Learning," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[37] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: Superfast and Robust Privacy-Preserving Machine Learning," in *USENIX Security Symposium (USENIX Security)*, 2021.

[38] N. Koti, S. M. Patil, A. Patra, and A. Suresh, "MPClan: Protocol Suite for Privacy-Conscious Computations," *Journal of Cryptology*, 2023.

[39] N. Koti, A. Patra, R. Rachuri, and A. Suresh, "Tetrad: Actively Secure 4PC for Secure Training and Inference," in *Annual Network and Distributed System Security Symposium (NDSS)*, 2022.

[40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[41] Y. Lindell and B. Pinkas, "Privacy Preserving Data Mining," in *Annual International Cryptology Conference (CRYPTO)*, 2000.

[42] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious Neural Network Predictions via MiniONN Transformations," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[43] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient Comparison for Secure Multi-Party Computation," in *Financial Cryptography and Data Security (FC)*, 2021.

[44] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on Fully Homomorphic Encryption, Theory, and Applications," *Proceedings of the IEEE*, 2022.

[45] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A Cryptographic Inference Service for Neural Networks," in *USENIX Security Symposium (USENIX Security)*, 2020.

[46] P. Mohassel and P. Rindal, "ABY^3: A Mixed Protocol Framework for Machine Learning," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[47] P. Mohassel and Y. Zhang, "SecureML: A System for Scalable Privacy-Preserving Machine Learning," in *IEEE Symposium on Security and Privacy (IEEE S&P)*, 2017.

[48] J. Mono and T. Güneysu, "Implementing and Optimizing Matrix Triples with Homomorphic Encryption," in *Asia Conference on Computer and Communications Security (ASIACCS)*, 2023.

[49] J. Mono, C. Marcolla, G. Land, T. Güneysu, and N. Aaraj, "Finding and evaluating parameters for BGV," in *International Conference on Cryptology in Africa (AFRICACRYPT)*, 2023.

[50] L. K. L. Ng and S. S. M. Chow, "SoK: Cryptographic Neural-Network Computation," in *IEEE Symposium on Security and Privacy (IEEE S&P)*, 2023.

[51] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A New Approach to Practical Active-Secure Two-Party Computation," in *Annual International Cryptology Conference (CRYPTO)*, 2012.

[52] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation," in *USENIX Security Symposium (USENIX Security)*, 2021.

[53] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-Party Secure Inference," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[54] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "XONN: XNOR-based Oblivious Deep Neural Network Inference," in *USENIX Security Symposium (USENIX Security)*, 2019.

[55] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications," in *Asia Conference on Computer and Communications Security (ASIACCS)*, 2018.

[56] D. Rotaru, N. P. Smart, T. Tanguy, F. Vercauteren, and T. Wood, "Actively Secure Setup for SPDZ," *Journal of Cryptology*, 2021.

[57] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: scalable provably-secure deep learning," in *Annual Design Automation Conference (DAC)*, 2018.

[58] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[59] N. P. Smart and T. Tanguy, "TaaS: Commodity MPC via Triples-as-a-Service," in *ACM SIGSAC Conference on Cloud Computing Security Workshop (CCSW@CCS)*, 2019.

[60] N. P. Smart and T. Wood, "Error Detection in Monotone Span Programs with Application to Communication-Efficient Multi-party Computation," in *The Cryptographers' Track at the RSA Conference (CT-RSA)*, 2019.

[61] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-Party Secure Computation for Neural Network Training," *Proceedings of Privacy Enhancing Technologies (PoPETs)*, 2019.

[62] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning," *Proceedings of Privacy Enhancing Technologies (PoPETs)*, 2021.

[63] X. Wang, S. Ranellucci, and J. Katz, "Global-Scale Secure Multiparty Computation," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[64] W. Zheng, R. Deng, W. Chen, R. A. Popa, A. Panda, and I. Stoica, "Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning," in *USENIX Security Symposium (USENIX Security)*, 2021.

Appendix A.
MPC-2PC conversion for TinyOT values

Currently, FANNG covers the conversion of n-party to 2-party SPDZ-like MACs. This suffices for transferring matrix triples and will also be used for transferring beaver triples, authentication singles, and truncation masks when this functionality is incorporated into the dealers. However, to convert GCs, resharing of TinyOT values is required. We consider the following options to move part of the pre-processing to the dealers:

- Run TinyOT in the dealers to generate aBits and aANDs, then convert pairwise MACs for $\{\mathsf{D}_j\}_{j\in[\mathsf{N}_\mathcal{D}]}$ into SPDZ-like MACs for model owner M and client C.
- Run TinyOT in the dealers to generate aBits and aANDs, then convert pairwise MACs for $\{\mathsf{D}_j\}_{j\in[\mathsf{N}_\mathcal{D}]}$ into pairwise MACs for model owner M and client C.

The first option caters to a simple conversion mechanism but would involve modifying how aBits/aANDs are treated in SCALE-MAMBA, as the authentication process would change. The second option allows using aBits/aANDs without any modification in SCALE-MAMBA, but the resharing protocol requires Oblivious Transfer (OT) between the model owner M and the client C, making it more time-consuming. The two following sections demonstrate how to perform the different MAC conversions.

Recap: Pairwise MACs in TinyOT: For $x^j \in \mathbb{F}_2$ held by P_j, define the following two-party MAC representation, as used in 2-party TinyOT [51]:

$$[x] = (x^j, M^{j,i}, K^{i,j}), \qquad M^{j,i} = K^{i,j} + x^j \cdot \Delta^i$$

where P_j holds x^j and a MAC $M^{j,i}$, and P_i holds a local MAC key $K^{i,j}$ as well as the fixed, global MAC key Δ^i.

Similarly, we define the n-party representation of an additively shared value $x = x^1 + \cdots + x^n$:

$$[x] = (x^j, \{M^{j,i}, K^{i,j})\}_{i\neq j})_{j\in[n]}, \quad M^{j,i} = K^{i,j} + x^j \cdot \Delta^i$$

Pairwise MAC to single MAC: Resharing TinyOT (aAND) values used in daBits: To reshare the pairwise values $M^{j,i}$, $K^{i,j}$ and the global keys Δ^i, recombine the pairwise reshared values accordingly as shares of the new single global mac key $\Delta = \sum_i \Delta^i$. It is worth highlighting that for each client, we may need a different global MAC key Δ. This could be potentially pose an issue for combinatorial parameters in the pre-processing material, such as the bucketing parameter in the TinyOT pre-processing. However, given that our concrete application involves the order of millions of aBits, this already allows us to choose $B = 3$ for bucketing.

Pairwise MAC to pairwise MAC: Resharing TinyOT values for garbling: Pairwise MACs pose a greater challenge and are more expensive. Implementing any of these options as a stepping stone could not only be more costly but also more challenging than transitioning the Garbled Circuit implementation to the pre-processing phase.

In the following, we assume that the values that are reshared are the result of the *function-dependent* phase of [HSS17], rather than simply TinyOT triples. As we will see, even in this case things are already far from ideal. If we were to reshare the aAND triples and then perform the function-dependent phase of [HSS17] in the online phase of our implementation, the complexity only increases.

Given an n-party representation of a TinyOT value, the dealers can execute the resharing of x^j into $x^j = x^{j,\mathsf{M}} + x^{j,\mathsf{C}}$ and Δ^i into $\Delta^i = \Delta^{i,\mathsf{M}} + \Delta^{i,\mathsf{C}}$ in parallel, as described in §3.2. Note that we want the model owner M and client C to hold a 2-party TinyOT value share of the following form:

$$[x] = (x^j, M^{j,i}, K^{i,j}), \qquad M^{\mathsf{M},\mathsf{C}} = K^{\mathsf{C},\mathsf{M}} + x^M \cdot \Delta^C$$

$$\text{and} \quad M^{\mathsf{C},\mathsf{M}} = K^{\mathsf{M},\mathsf{C}} + x^C \cdot \Delta^M$$

Since $x^M = \sum_{j=1}^n x^{j,\mathsf{M}}$ and $\Delta^C = \sum_{i=1}^n \Delta^{i,\mathsf{C}}$, what is missing is the generation of the values $M^{\mathsf{M},\mathsf{C}}$ and $K^{\mathsf{C},\mathsf{M}}$. These values are not easily obtained through resharing the n-party TinyOT representation of x. At this point, M and C can obtain $M^{\mathsf{M},\mathsf{C}}$ and $K^{\mathsf{C},\mathsf{M}}$ by engaging in a correlated OT protocol, with the input being the choice bit x^M and a fixed correlation Δ^C.

Appendix B.
I/O Configuration File

```
Storage_type =    MySQLDatabase
MySQL_url =       tcp://my.dbserver.ae:3306
MySQL_user =      my_user
MySQL_password =  ****
MySQL_database =  mpclib-database-dealer
```

Listing 2: Configuration file tuned for MySQL access.

```
Storage_type =                  FileSystem
File_system_storage_directory = Data
```

Listing 3: Configuration file tuned for File System access.

Appendix C.
Network Architectures

Layer	Input Size	Description	Output
Convolution	$32 \times 32 \times 1$	Window size 5×5, Stride $(1,1)$, Padding $(0,0)$, output channels 6	$28 \times 28 \times 6$
ReLU Activation	$28 \times 28 \times 6$	ReLU$(\cdot)$ on each input	$28 \times 28 \times 6$
Max Pooling	$28 \times 28 \times 6$	Window size 2×2, Stride $(2,2)$	$14 \times 14 \times 6$
Convolution	$14 \times 14 \times 6$	Window size 5×5, Stride $(1,1)$, Padding $(0,0)$, output channels 16	$10 \times 10 \times 16$
ReLU Activation	$10 \times 10 \times 16$	ReLU$(\cdot)$ on each input	$10 \times 10 \times 16$
Max Pooling	$10 \times 10 \times 16$	Window size 2×2, Stride $(2,2)$	$5 \times 5 \times 16$
Convolution	$5 \times 5 \times 16$	Window size 5×5, Stride $(1,1)$, Padding $(0,0)$, output channels 120	$1 \times 1 \times 120$
ReLU Activation	$1 \times 1 \times 120$	ReLU$(\cdot)$ on each input	$1 \times 1 \times 120$
Fully Connected Layer	120	Fully connected layer	84
ReLU Activation	84	ReLU$(\cdot)$ on each input	84
Fully Connected Layer	84	Fully connected layer	10

Figure 8: LeNet network architecture [40] for training over MNIST dataset.

Layer	Input Size	Description	Output
Convolution	$32 \times 32 \times 3$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 32	$32 \times 32 \times 32$
Batch Normalization	$32 \times 32 \times 32$	BN$(\cdot)$ on each input	$32 \times 32 \times 32$
ReLU Activation	$32 \times 32 \times 32$	ReLU$(\cdot)$ on each input	$32 \times 32 \times 32$
Convolution	$32 \times 32 \times 32$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 64	$32 \times 32 \times 64$
Batch Normalization	$32 \times 32 \times 64$	BN$(\cdot)$ on each input	$32 \times 32 \times 64$
ReLU Activation	$32 \times 32 \times 64$	ReLU$(\cdot)$ on each input	$32 \times 32 \times 64$
Max Pooling	$32 \times 32 \times 64$	Window size 2×2, Stride $(2,2)$	$16 \times 16 \times 64$
Convolution	$16 \times 16 \times 64$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 128	$16 \times 16 \times 128$
Batch Normalization	$16 \times 16 \times 128$	BN$(\cdot)$ on each input	$16 \times 16 \times 128$
ReLU Activation	$16 \times 16 \times 128$	ReLU$(\cdot)$ on each input	$16 \times 16 \times 128$
Convolution	$16 \times 16 \times 128$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 128	$16 \times 16 \times 128$
Batch Normalization	$16 \times 16 \times 128$	BN$(\cdot)$ on each input	$16 \times 16 \times 128$
ReLU Activation	$16 \times 16 \times 128$	ReLU$(\cdot)$ on each input	$16 \times 16 \times 128$
Max Pooling	$16 \times 16 \times 128$	Window size 2×2, Stride $(2,2)$	$8 \times 8 \times 128$
Convolution	$8 \times 8 \times 128$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 256	$8 \times 8 \times 256$
Batch Normalization	$8 \times 8 \times 256$	BN$(\cdot)$ on each input	$8 \times 8 \times 256$
ReLU Activation	$8 \times 8 \times 256$	ReLU$(\cdot)$ on each input	$8 \times 8 \times 256$
Convolution	$8 \times 8 \times 256$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 256	$8 \times 8 \times 256$
Batch Normalization	$8 \times 8 \times 256$	BN$(\cdot)$ on each input	$8 \times 8 \times 256$
ReLU Activation	$8 \times 8 \times 256$	ReLU$(\cdot)$ on each input	$8 \times 8 \times 256$
Max Pooling	$8 \times 8 \times 256$	Window size 2×2, Stride $(2,2)$	$4 \times 4 \times 256$
Fully Connected Layer	4096	Fully connected layer	32
ReLU Activation	32	ReLU$(\cdot)$ on each input	32
Fully Connected Layer	32	Fully connected layer	10

Figure 9: CNN network architecture for private classification over CIFAR-10 dataset. It contains 1.5 million paramerters

Layer	Input Size	Description	Output
Convolution	$32 \times 32 \times 3$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 64	$32 \times 32 \times 64$
ReLU Activation	$32 \times 32 \times 64$	ReLU$(\cdot)$ on each input	$32 \times 32 \times 64$
Convolution	$32 \times 32 \times 64$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 64	$32 \times 32 \times 64$
ReLU Activation	$32 \times 32 \times 64$	ReLU$(\cdot)$ on each input	$32 \times 32 \times 64$
Max Pooling	$32 \times 32 \times 64$	Window size 2×2, Stride $(2,2)$	$16 \times 16 \times 64$
Convolution	$16 \times 16 \times 64$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 128	$16 \times 16 \times 128$
ReLU Activation	$16 \times 16 \times 128$	ReLU$(\cdot)$ on each input	$16 \times 16 \times 128$
Convolution	$16 \times 16 \times 128$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 128	$16 \times 16 \times 128$
ReLU Activation	$16 \times 16 \times 128$	ReLU$(\cdot)$ on each input	$16 \times 16 \times 128$
Max Pooling	$16 \times 16 \times 128$	Window size 2×2, Stride $(2,2)$	$8 \times 8 \times 128$
Convolution	$8 \times 8 \times 128$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 256	$8 \times 8 \times 256$
ReLU Activation	$8 \times 8 \times 256$	ReLU$(\cdot)$ on each input	$8 \times 8 \times 256$
Convolution	$8 \times 8 \times 256$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 256	$8 \times 8 \times 256$
ReLU Activation	$8 \times 8 \times 256$	ReLU$(\cdot)$ on each input	$8 \times 8 \times 256$
Convolution	$8 \times 8 \times 256$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 256	$8 \times 8 \times 256$
ReLU Activation	$8 \times 8 \times 256$	ReLU$(\cdot)$ on each input	$8 \times 8 \times 256$
Max Pooling	$8 \times 8 \times 256$	Window size 2×2, Stride $(2,2)$	$4 \times 4 \times 256$
Convolution	$4 \times 4 \times 256$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 512	$4 \times 4 \times 512$
ReLU Activation	$4 \times 4 \times 512$	ReLU$(\cdot)$ on each input	$4 \times 4 \times 512$
Convolution	$4 \times 4 \times 512$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 512	$4 \times 4 \times 512$
ReLU Activation	$4 \times 4 \times 512$	ReLU$(\cdot)$ on each input	$4 \times 4 \times 512$
Convolution	$4 \times 4 \times 512$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 512	$4 \times 4 \times 512$
ReLU Activation	$4 \times 4 \times 512$	ReLU$(\cdot)$ on each input	$4 \times 4 \times 512$
Max Pooling	$4 \times 4 \times 512$	Window size 2×2, Stride $(2,2)$	$2 \times 2 \times 512$
Convolution	$2 \times 2 \times 512$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 512	$2 \times 2 \times 512$
ReLU Activation	$2 \times 2 \times 512$	ReLU$(\cdot)$ on each input	$2 \times 2 \times 512$
Convolution	$2 \times 2 \times 512$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 512	$2 \times 2 \times 512$
ReLU Activation	$2 \times 2 \times 512$	ReLU$(\cdot)$ on each input	$2 \times 2 \times 512$
Convolution	$2 \times 2 \times 512$	Window size 3×3, Stride $(1,1)$, Padding $(1,1)$, output channels 512	$2 \times 2 \times 512$
ReLU Activation	$2 \times 2 \times 512$	ReLU$(\cdot)$ on each input	$2 \times 2 \times 512$
Max Pooling	$2 \times 2 \times 512$	Window size 2×2, Stride $(2,2)$	$1 \times 1 \times 512$
Fully Connected Layer	512	Fully connected layer	4096
ReLU Activation	4096	ReLU$(\cdot)$ on each input	4096
Fully Connected Layer	4096	Fully connected layer	4096
ReLU Activation	4096	ReLU$(\cdot)$ on each input	4096
Fully Connected Layer	4096	Fully connected layer	1000
ReLU Activation	1000	ReLU$(\cdot)$ on each input	1000

Figure 10: VGG16 network architecture [58] for training over CIFAR-10 dataset. It contains 37 million parameters.