SoK: Connecting the Dots in Privacy-Preserving ML – Systematization of MPC Protocols and Conversions Between Secret Sharing Schemes

Anonymous Authors

Abstract—**Privacy-preserving machine learning (PPML) has become increasingly important due to the need to protect sensitive data during training and inference. Secure multiparty computation (MPC) and homomorphic encryption (HE) have emerged as foundational technologies, enabling secure computation over private data. In this work, we provide a systematic comparative overview of MPC frameworks for PPML, focusing on protocols that introduce novel approaches rather than incremental improvements. Frameworks are analyzed based on computational and communication complexity, throughput, security guarantees, and applicability in small-party settings. Each underlying primitive in PPML is examined from an MPC perspective, highlighting its role and trade-offs. We also emphasize the diversity of secret-sharing schemes and associated interoperability challenges, proposing scheme conversions to facilitate efficient hybrid solutions. This Systematization of Knowledge guides researchers in identifying open problems and practitioners in selecting effective MPC-based frameworks for real-world PPML deployment.**

1. Introduction

With the rapid advancement of machine learning (ML) and artificial intelligence (AI), the protection of personal and sensitive data has become a critical concern for researchers, practitioners, and regulatory bodies alike. This heightened awareness underscores the urgent need for privacy-enhancing technologies (PETs) [109], which play a pivotal role in preserving data privacy while leveraging the capabilities of ML and AI algorithms. As organizations increasingly adopt ML to drive decisions and innovation—across application domains such as healthcare diagnostics [3], IoT deployments [1], and financial modelling [19]—the imperative to safeguard private information becomes ever more pressing. Concurrently, the regulatory landscape is evolving, with stringent data protection laws and regulations compelling organizations to embed privacy considerations into their ML applications [21], [67].

Among the broad spectrum of PETS, secure multiparty computation (MPC) [108], homomorphic encryption (HE) [4], and differential privacy (DP) [58] stand out as foundational techniques for enabling privacy-preserving machine learning (PPML). MPC allows multiple parties to jointly compute functions over private inputs without revealing them, making it well-suited for domains where data sharing is restricted, such as healthcare and finance. HE enables computations directly on encrypted data, particularly useful in non-interactive server–client settings. DP provides a statistical notion of privacy by injecting carefully calibrated noise into outputs, ensuring that individual contributions remain indistinguishable.

The increasing deployment of ML/AI in data-sensitive sectors has intensified interest in PPML, with MPC- and HE-based approaches receiving particular attention. This is evident from the substantial body of research in the area, with over 125 cryptography-based PPML works published in the last decade and continuing momentum [41]. While several surveys have systematized these works primarily from an ML perspective, the cryptographic motivations behind this surge remain less explored [9], [96]. In this work, we focus on systematizing research on MPC for PPML, providing a comparative overview of frameworks with respect to computational and communication costs, throughput, and security guarantees across different stages of the ML pipeline. HE-based approaches are also acknowledged where relevant, although their techniques are not covered in depth.

Given the large volume of works in this domain, we focus on systematically analyzing MPC-based approaches for PPML that propose new protocols, rather than those that primarily improve the performance of existing schemes or extend their scope. For in-depth discussion, we restrict our scope to the widely studied two-party (2PC), three-party (3PC), and four-party (4PC) settings with one corruption each. These settings serve as foundational models for dishonest-majority ($t < N$), honest-majority ($t < N/2$), and super-honest-majority ($t < N/3$) scenarios, representing up to t corruptions among N participating parties. Our analysis examines each underlying primitive in PPML from a cryptographic perspective, emphasizing its functional role, and efficiency trade-offs. In analyzing these frameworks, we highlight the diverse secret-sharing schemes employed and the resulting interoperability challenges. To address these gaps, we propose generalized functionalities for common ML operations, as well as efficient protocols for switching between secret-sharing schemes. This Systematization of Knowledge (SoK) is intended both for researchers seeking an overview of MPC-based PPML and for practitioners evaluating frameworks for deployment. By consolidating the state of the art and identifying open challenges, our work aims to support the development of more practical and interoperable PPML solutions.

TABLE 1: High-level comparison of our SoK with prior works, covering supported techniques, deployment settings, ML tasks, and unique contributions. A more detailed comparison is provided in Table 8.

Reference	Year	Techniques				Setting			ML		Added value
		MPC	FSS	FHE	Hybrid	C/S	TTP	Outsourcing	Inference	Training	
Riazi et al. [144]	2019	●	○	○	●	●	●	●	●	●	Overview of attacks on ML
Azraoui et al. [13]	2019	●	○	●	●	●	○	●	●	●	Limited experimental evaluation
Cabrero-Holgueras et al. [35]	2021	●	○	●	●	●	○	●	●	●	Bridge between research and industry
Ng et al. [126]	2023	●	○	●	●	●	○	●	●	●	Genealogy
Khan et al. [96]	2024	●	○	●	○	○	○	●	○	●	Experimental evaluation
This Work	**2025**	●	●	○	●	●	●	●	●	●	**Low-level protocol analysis, Conversion protocols, Genealogy**

1.1. Related Work

In this section, we compare our work with existing surveys and SoK studies on PPML frameworks. Table 1 provides a comparative overview of our work relative to prior SoKs, while a more detailed summary, including relevant surveys, is presented in Table 8 in Appendix A.

Surveys. The growing interest in PPML has led to a series of surveys that provide broad overviews of the field. Early surveys [24], [143], [161], [163], [178] primarily outlined the stages of PPML and briefly mentioned cryptographic tools such as MPC, HE, DP, and Trusted Execution Environments (TEEs). Later works expanded this perspective by providing detailed treatments of cryptographic techniques [9], [136] and by addressing security and privacy aspects, including attacks and defenses in PPML [115].

With the field's maturation, more recent surveys integrate both system stages and security/privacy concerns while consolidating new contributions [41], [121], [160]. Some surveys focus exclusively on HE [132], [133], [134], others on MPC [189], and a few treat HE as a subclass of MPC [195]. A subset of these efforts further emphasize practical implementation, including benchmark reproduction on standard ML models [102], [127], [189]. Overall, these surveys typically cover a mix of approaches—PPML stages, security aspects, and cryptographic methods—providing a high-level overview of the PPML landscape. However, they offer limited insights into system-level frameworks, protocol-level implementations, or the interplay of cryptographic building blocks within practical ML workflows.

SoKs. Though there are several surveys in the domain of PPML, SoKs in this area remain limited. Riazi et al. [144] presented one of the earliest SoKs in PPML, surveying cryptography-based frameworks for both training and inference in client-server and outsourcing settings. Their study also reviewed attacks on ML models (e.g., inference and extraction) and discussed how various frameworks mitigate them. Concurrently, Azraoui et al. [13] examined the interplay between neural networks (NNs) and cryptographic techniques, offering limited experimental evaluation of selected schemes.

Cabrero-Holgueras and Pastrana [35] conducted an extensive SoK bridging research prototypes and industry deployment. They analyzed open-source PPML projects and MPC libraries, highlighting efficiency, usability, and reproducibility issues that hinder real-world adoption, and emphasized the lack of standardization and integration challenges in popular ML frameworks.

Ng and Chow [126] conducted a large-scale SoK covering over 50 cryptographic PPML frameworks from 2016 to 2022. They mapped the genealogy of PPML research, categorizing frameworks by problem setting (e.g., oblivious inference, outsourced training, multi-party training) and showing how ideas evolved. They also re-evaluated several protocols under WAN conditions, highlighting the impact of communication latency on performance. While offering valuable lineage and high-level guidance, their work omitted low-level operational details, cross-protocol integrations, and non-cryptographic approaches such as TEEs and differential privacy.

Khan et al. [96] recently presented an SoK on open-source PPML frameworks for deep learning training, focusing on HE- and MPC-based solutions. They conducted an extensive empirical evaluation, reproducing results on standard benchmarks and comparing frameworks by runtime, communication, and model accuracy. Their key contribution was exposing the gap between theoretical proposals and practical usability through an assessment of security assumptions, open-source availability, and reproducibility. The study, however, was limited to training (excluding inference) and a limited set of schemes.

1.2. Our Contributions

Though prior SoKs in PPML have provided valuable surveys of frameworks and cryptographic techniques, they have largely remained siloed, focusing on high-level frameworks, specific cryptographic primitives, or narrow use cases. Notably, none has examined how to *connect the dots* between protocols, e.g., via cross-scheme conversions or reusable low-level components. Our work fills this gap by offering a unifying perspective that integrates multiple cryptographic paradigms, systematically decomposes frameworks, and highlights connections overlooked in prior efforts.

We advance beyond previous SoKs by analyzing both training and inference workloads across MPC and hybrid MPC–HE approaches under diverse trust models. Our study emphasizes not only frameworks, but also the low-level building blocks, interoperability challenges, and deployment considerations that shape real-world applicability. We cover diverse workloads, including linear and non-linear layers, transformers, and LLMs, integrating the latest advances through 2025.

To ensure novelty and relevance, we conducted a thorough literature review focusing on protocols and frameworks introducing fundamentally new approaches rather than incremental extensions. Novelty and significance were our primary inclusion criteria, and protocols were categorized by ML applicability and improvements over prior work, enabling systematic evaluation of how each framework advances the state-of-the-art across MPC and ML dimensions.

Our main contributions are summarized as follows:

- **Design & deployment dimensions:** We systematize MPC-based PPML protocols along key dimensions—algebraic structure, threat model, execution phase, deployment mode, and network—highlighting their efficiency and security trade-offs.
- **Low-level protocol analysis:** We decompose PPML frameworks into core cryptographic primitives (e.g., fixed-point multiplication, comparison), quantify their costs, and identify performance bottlenecks that drive system-level efficiency.
- **Coverage of advanced ML models:** We extend beyond prior SoKs to include recent protocols for transformers and LLMs, analyzing Function Secret Sharing (FSS)-based [76], [91] and additive-secret-sharing approaches [104], [107].
- **Unification and conversions:** Through the *MPC Puzzle*, we unify 2-, 3-, and 4-party secret-sharing schemes and present the first comparative study of the conversion techniques among them, including an analysis of their communication costs.
- **Comparative mapping:** We systematically map MPC frameworks to diverse ML tasks (regression, classification, neural networks), presenting comparative tables of adversary models, complexities, and deployment settings—offering practical guidance for researchers and system builders.
- **Updated genealogy and taxonomy:** We extend prior genealogies [126] to include works through 2025 and propose a new taxonomy based on cryptographic composition and layer-wise techniques, capturing both breadth and evolutionary trends.

In summary, our work provides a comprehensive SoK that not only surveys existing PPML frameworks but also bridges their underlying cryptographic foundations. By connecting low-level protocols, analyzing conversion techniques, and incorporating state-of-the-art advances, our theoretical study serves as a stepping stone toward practical deployment, offering a solid foundation for future research and implementation of MPC-based PPML systems.

Organization. Section 2 introduces preliminaries and background on PPML and MPC. Section 3 presents a systematization from the MPC perspective, focusing on design and deployment aspects such as algebraic structures, threat models, and execution phases. Section 4 systematizes existing works according to ML architectures, covering both linear and non-linear layers and their associated computations. Section 5, *The MPC Puzzle*, unifies secret-sharing schemes across 2-, 3-, and 4-party settings and analyzes conversion techniques between them. Section 6 concludes with key findings and insights gained from our study. Given the large volume of relevant works, each main section is complemented by additional material in the appendix.

2. Preliminaries and Background

This section introduces the foundations of privacy-preserving machine learning (PPML), including its goals, the role of secure multiparty computation (MPC), and methods for representing real numbers. We also summarize the notation used throughout the paper and outline our scope of coverage. Further details on MPC primitives are provided in Appendix B.

Privacy-Preserving Machine Learning. PPML encompasses two main tasks: private inference and private training. In inference, a user queries a model owner's private model to obtain predictions, while ensuring confidentiality of both the user's input and the provider's model. Private inference typically follows one of two approaches: (i) the *client–server* model, where the client and model provider interact directly with asymmetric roles; or (ii) *outsourcing*, where a cluster of servers performs computation on secret-shared inputs provided by both parties. In both cases, only the designated recipient learns the output. For training, the setting is analogous: computation may be outsourced to untrusted servers, or data owners may act as the computing parties themselves. Unlike inference, however, the setting is more symmetric, as the goal is for all participants to jointly obtain the trained model.

Secure Multiparty Computation. The concept of MPC originates from Yao's seminal work [180], which introduced a two-party protocol based on garbled circuits to privately compare values (the millionaire problem). In general, MPC enables mutually distrusting parties to jointly compute a public function without revealing their private inputs, beyond what can be inferred from the output. MPC provides two main guarantees: *privacy*, ensuring no information is leaked beyond what can be inferred from the output and a party's own input, and *correctness*, ensuring that all honest parties obtain the correct function output [108].

Machine learning is computationally intensive even in the plaintext setting, where extensive optimizations have already been explored. MPC-based PPML research focuses on reducing the additional overhead of privacy while narrowing the gap between plaintext and secure computation. Since not all ML tasks are naturally MPC-friendly, some approximations are often required, with the goal of minimizing the resulting accuracy loss.

Handling Real Numbers. A key challenge in applying MPC to PPML lies in the mismatch between the algebraic structures supported by MPC and the real numbers used in machine learning. MPC protocols natively operate over modular arithmetic, either in an ℓ-bit ring $\mathbb{Z}_{2^\ell}$ or a prime field $\mathbb{F}_p$, whereas ML computations typically require real-valued operations.

The most widely used approach is *fixed-point arithmetic (FPA)* [36], [37]. A real number $\tilde{x} \in \mathbb{R}$ is approximated by

$$x = \lfloor \tilde{x} \cdot 2^k \rfloor \in \mathbb{Z},$$

where k denotes the number of precision bits. Since this representation is approximate, the choice of k is critical for accuracy and efficiency. For example, experimental results show that when using 64-bit rings, at least 16 precision bits are needed for stable training and inference with deep models such as ResNet-50 [82].

Although less common, MPC can also support alternative representations of real numbers. One option is *floating-point arithmetic* [8], [31], [135], where a value is represented by a sign bit, exponent, and mantissa. This approach offers higher accuracy than FPA but typically incurs significant performance overhead due to its computational complexity.

Notation. For clarity and consistency, all notation and abbreviations used throughout the paper are summarized in Table 2.

TABLE 2: List of notation and abbreviations.

Notation	
N	Number of parties
t	Corruption threshold
ℓ	Size of the ring $\mathbb{Z}_{2^\ell}$ in bits
p	Size of the field $\mathbb{F}_p$ in bits
ℓ_x	Input magnitude in fixed-point arithmetic
n	Vector size, unless mentioned explicitly
μ/ν	Input/Output bitwidth in mixed computation
$f(\cdot)$	Function
λ	Computation security parameter
κ	Statistical security parameter
s	slack in SPDZ2k [46]; $s + \log s = \kappa$
CT	Size of an HE ciphertext
$\mathsf{a/r/m}$	additive / replicated / masked secret sharing
A^C / A^S	Malicious client / server
A / F / R	Abort / Fairness / Robustness
$\xleftarrow{\$_{i,j}} M$	Uniform random sampling from a set M from a shared PRNG seed between parties P_i and P_j.
Abbreviations	
Sec.	Security
Tech.	Technique
Comm.	Communication
SH	Semi-honest
HM / DM	Honest / Dishonest Majority
TTP	Trusted Third Party
BT	Beaver Triples
A-B-G	Arithmetic-Boolean-Garbled
PRNG	Pseudo-random number generator

Exclusions. Our study focuses exclusively on MPC-based PPML protocols. Other privacy-preserving approaches such as Trusted Execution Environments (TEEs), pure homomorphic encryption (HE), and functional encryption are outside our scope. Note that MPC ensures privacy only from inputs to outputs of the computation, whereas in ML stronger threats exist, such as inference attacks that treat PPML services as black boxes. Addressing these requires complementary techniques like differential privacy (DP). Finally, while we cover PPML on deep neural networks, including transformers, private ML based on Federated Learning (FL) represents a distinct line of work, which we leave to future systematization.

3. Systematization from an MPC Design and Deployment Perspective

We systematize MPC-based PPML works along key facets of MPC that shape the properties of the resulting protocols. Specifically, we consider the following dimensions: algebraic structure, threat model, execution phase, deployment mode, and network—each introducing distinct efficiency trade-offs. Table 3 summarizes these dimensions, with further details presented in Appendix C.

3.1. Threat Model

A central dimension in MPC is the threat model, which specifies the nature and extent of adversarial corruption. PPML protocols mainly target either semi-honest (passive) adversaries, where corrupted parties follow the protocol but try to infer extra information, or malicious (active) adversaries, where they may arbitrarily deviate. While semi-honest security yields far more efficient protocols [125], [130], [142], achieving malicious security incurs significant overhead [98], [124], [171], [183], [190], which becomes prohibitive for large networks such as deep NNs. Another line of work studies *asymmetric corruption*, motivated by practical deployments, where some parties may be corrupted maliciously while others only semi-honestly. In the 2PC client–server setting, works such as [38], [103], [174] considered a malicious client, while [61] considered a malicious server. DEtI [33] extended this notion to 3PC, allowing one of the three parties to be maliciously corrupted.

With respect to the extent of corruption, early PPML works primarily focused on the 2PC setting under dishonest-majority assumptions ($t < N$, with at most t corruptions among N parties) and semi-honest security [125], [130], [142]. Even in this setting, protocols required expensive public-key primitives such as oblivious transfer [140], [141], [142] or homomorphic encryption [69], [86], [88], [107], [193] to support operations such as matrix multiplications. Malicious security in 2PC incurs even higher computational and communication costs [2], [183], [190]. Function Secret Sharing (FSS) [26] based protocols later emerged in this setting to reduce communication at the expense of heavier computation, often relying on a trusted dealer to prepare and distribute FSS keys [75], [76], [77], [91], [150].

To improve performance, subsequent works explored the 3PC setting under honest-majority assumptions ($t < N/2$) and the 4PC setting in super honest-majority ($t < N/3$). Semi-honest 3PC protocols [39], [124], [170] achieve high efficiency, but malicious security incurs significant overhead due to consistency checks, and techniques like cut-and-choose, and distributed zero-knowledge proofs that are used within [98], [131], [171]. In contrast, 4PC protocols benefit from an additional honest party, reducing these expensive computations and enabling more efficient maliciously secure PPML [34], [40], [51], [82], [100].

Insight: Adversarial Trade-offs

Choosing the number of parties and security model directly impacts efficiency: honest-majority multi-party settings reduce cryptographic overhead, while stronger malicious guarantees increase computation and communication.

3.2. Algebraic Structure

Most PPML frameworks operate over rings, typically $\mathbb{Z}_{2^\ell}$, due to efficient hardware support and fast modular arithmetic—crucial for large deep learning circuits [52]. For MPC non-friendly operations, such as comparisons, protocols often switch to the Boolean domain ($\mathbb{Z}_2$) (see Section 4.2) and revert to $\mathbb{Z}_{2^\ell}$ for the rest [124], [130]. For malicious security, some works use extended rings [46], [51], while others adopt fields ($\mathbb{F}_p$) to leverage stronger algebraic guarantees at the cost of performance [12], [38], [103], [174]. In an orthogonal line, hybrid MPC–HE approaches combine fields for HE-based linear layers with rings for MPC-based non-linear layers [69], [88], [117].

Insight: Rings vs. Fields

Rings ($\mathbb{Z}_{2^\ell}$) are preferred for efficiency, while fields ($\mathbb{F}_p$) offer stronger guarantees at higher cost; extended rings provide a middle ground for malicious security.

3.3. Execution Phase

PPML protocols differ in how they split computation between offline (input-independent) and online (input-dependent) phases, introducing trade-offs in latency, communication, and deployment complexity. Online-only protocols [88], [117], [124], [129], [140], [141] require no setup and execute entirely at runtime, making them suitable for streaming inference, where inputs arrive sequentially and there is no opportunity for preprocessing, or client-server scenarios where preprocessing is infeasible. Their main drawback is higher communication and computation cost during online inference.

Preprocessing-based protocols offload costly cryptographic operations to an offline phase, enabling a lightweight and fast online phase. Preprocessing can be function-independent [51], [78], [92], [107], [111], [125], [142], [171], [183], [190], where generated randomness or masks can be reused across computations, or function-dependent [33], [34], [39], [81], [98], [130], [131], where the preprocessed material is tailored to a specific circuit or model. Function-independent preprocessing is flexible and reusable, while function-dependent preprocessing yields faster online execution albeit being circuit-specific. For larger circuits with partially unknown components, function-dependent preprocessing can be leveraged by combining with secret-sharing conversions (see Section 5). The offline phase is sometimes outsourced to a semi-trusted third party (a.k.a. dealer-based or commodity-based setup) [76], [77], [90], [91], [113], [114], [150] or to multiple independent parties for distributed execution [2].

Insight: Precompute or Stream?

PPML protocols trade off flexibility and setup: online-only maximizes adaptability, while preprocessing reduces online cost at the expense of setup.

3.4. Deployment Mode

Client-server protocols, typical in 2PC settings, are ideal for interactive inference where both parties (e.g., user and model provider) are online with their inputs in the clear [17], [75], [76], [78], [79], [113], [139], [142]. They are asymmetric: the server handles most computation while the client remains lightweight. This mode is easy to deploy but limits delegation and asynchronous execution. Extensions include 3PC client-server protocols [33], [156] and hybrid MPC–HE variants [38], [69], [88], [89], [92], [103], [107], [111], [117], [117], [123], [129], [162], [174], [179], [193], where the client performs more computation. In contrast, outsourcing protocols enable delegation, allowing data and model owners to remain offline while distributing encrypted or secret-shared inputs to untrusted compute parties (e.g., cloud servers) [39], [40], [51], [62], [81], [93], [98], [100], [114], [124], [125], [170]. This mode supports asynchronous execution, multi-client batching, and suits large-scale, audit-driven environments.

Insight: Deployment Flexibility

Client-server suits interactive inference; outsourcing enables delegation and asynchronous large-scale computation.

3.5. Network

The performance of MPC-based PPML protocols is heavily influenced by network conditions, particularly bandwidth and latency. High-throughput protocols [40], [51], [100], [130], [131] aim to process multiple inputs concurrently, making them suitable for batch inference or offline training, over high-bandwidth links. These protocols typically use secret sharing and rely on preprocessing and batching to amortize communication. In contrast, low-latency protocols are designed for real-time inference, where individual predictions must complete quickly. These are better suited for interactive or client-server settings, such as mobile deployments or WAN scenarios. They often rely on garbled circuits [17], [44], [111], [149], FSS [75], [76], [77], [91],

[150], [169], [177], or HE [38], [92], [103], [123], [174]. Some protocols adopt a hybrid approach for performance, using high-throughput methods for majority of the computation and low-latency techniques for other components, often preprocessing [88], [90], [100], [142].

Insight: Network Matters

PPML performance hinges on the trade-off between high-throughput (batch-oriented) and low-latency (real-time) protocols, with some systems combining both.

TABLE 3: Categorization of PPML frameworks by MPC design. Selected works are shown; a complete categorization is presented in Table 14.

Dimension	Method		Frameworks
Threat Model	DM	SH	[74], [86], [93], [107], [179], [182]
		A	[75], [169], [183], [190]
		A^C	[38], [103], [174]
		A^S	[61]
	HM	SH	[7], [15], [63], [81], [110], [156], [167]
		A/F/R	[12], [39], [124], [131], [171], [191]
	Super HM	SH	[48], [105]
		A/F/R	[34], [40], [51], [81], [98], [100]
Algebraic Structure	$\mathbb{Z}_{2^\ell}$		[43], [60], [76], [78], [91], [150], [193]
	$\mathbb{Z}_{2^\ell}$ and $\mathbb{Z}_2$		[51], [98], [100], [124], [125], [130]
	$\mathbb{Z}_{2^\ell}$ and $\mathbb{F}_p$		[62], [101], [170], [171]
	$\mathbb{F}_p$		[12], [38], [103], [174]
	$\mathbb{Z}_2$		[149]
Execution Phase	Online-only		[88], [117], [124], [129], [140], [141]
	Pre-processing	f-dependent	[33], [34], [39], [81], [98], [130], [131]
		f-independent	[51], [78], [92], [107], [111], [125], [142], [171], [183], [190]
	Dealer	TTP	[76], [91], [113], [114], [150], [169]
		Distributed	[2]
Deployment Mode	Outsourcing		[39], [40], [51], [62], [81], [93], [98], [100], [114], [124], [125], [170]
	Client-Server		[17], [75], [76], [78], [79], [113], [139]
Network	High Throughput		[40], [51], [100], [130], [131]
	Low Latency		[17], [44], [75], [76], [77], [91], [111], [149], [150], [169], [177]
	Mixed		[88], [90], [100], [142]

4. Systematization by ML Architecture

The architecture of neural networks (NNs), including modern designs such as Transformers (see Appendix D.4), can be broadly divided into two categories: *linear layers* and *non-linear layers*. In this section, we systematically analyze how PPML frameworks handle these layers, with further details provided in Appendix D.

4.1. Linear Layers

A linear layer in ML computes an affine transformation of the form $\vec{y} = \mathbf{W} \cdot \vec{x} + \vec{b}$, where $\mathbf{W}$ is a learned weight matrix, $\vec{b}$ is a bias vector, and $\vec{x}$ is the input [125]. Convolutional layers can be expressed in the same form, with $\mathbf{W}$ representing local filters applied across spatial dimensions [157], [170].

Linear layers mainly involve multiplications—commonly dot products, matrix multiplications, or convolutions—and, under fixed-point arithmetic, are typically followed by a truncation primitive. In what follows, we examine existing PPML works in relation to these building blocks.

4.1.1. Dot Product (Scalar Product). For two n-length vectors $\vec{x}, \vec{y} \in \mathbb{Z}_{2^\ell}^n$, the dot product (or scalar product) is defined as

$$z = \vec{x} \odot \vec{y} = \sum_{i=1}^{n} x_i \cdot y_i.$$

In the context of PPML, protocols for secure dot-product computation can be broadly classified into three categories: (i) *online-only protocols*, which perform all computations during the online phase; (ii) *preprocessing/offline-based protocols*, which leverage an offline phase to generate correlated randomness or cryptographic material; and (iii) *dealer/helper-assisted protocols*, which rely on an additional trusted party or auxiliary helper to facilitate computation.

Online-only. In the 2PC client-server setting, most online-only dot-product protocols rely on homomorphic encryption (HE), where the client sends encrypted inputs and the server performs computations on ciphertexts [22], [78], [88], [92], [107], [111], [117], [129], [142]. Communication cost is dominated by the large size of HE ciphertexts, and for efficiency, the server often assumes access to the model in the clear. In contrast, 3PC and 4PC protocols exploit replicated secret sharing: each party locally computes all n partial products $x_i \cdot y_i$, then aggregates results in a single communication step, making the communication cost independent of the vector length [12], [49], [51], [63], [81], [124], [171].

To achieve malicious security, Boyle et al. [29] augmented a semi-honest 3PC protocol with zero-knowledge proofs (ZK). While this adds computational overhead, the amortized communication remains the same as in the semi-honest setting, and later works leveraged this idea for offline phases to further optimize online efficiency [98], [131].

Preprocessing/Offline-based. In this setting, parties first execute an input-independent preprocessing phase to generate correlated randomness, which is later consumed during the input-dependent online phase. In 2PC, the standard primitive is the Beaver triple (BT) [20] $(a, b, a \cdot b)$, typically produced via OT- or HE-based methods, yielding an online communication cost of $2n$ public reconstructions [89], [125]. Client–server frameworks similarly pre-compute dot product layers using HE [18], [69], [123], [162], [188], [193]. ABY2.0 [130] utilized function-dependent preprocessing, enabling online communication that is independent of vector length.

In 3PC, preprocessing is often used to strengthen malicious security by generating random multiplication triples for verifying online computations [124], [171]. Other works push parts of the computation to the offline phase through function-dependent preprocessing, improving online efficiency and in some cases reducing the number of active parties [39], [98], [131]. Similar strategies have also been explored in 4PC protocols [34], [40], [81], [100].

Dealer/Helper-assisted. In this model, sometimes referred to as *commodity-based MPC*, an external dealer provides correlated randomness, most commonly Beaver triples [113], [114]. In FSS-based protocols, the dealer is instead responsible for generating and distributing the FSS keys [76], [77], [79], [91], [150], [169]. Some works retain the dealer as a semi-honest helper even during the online phase to further reduce communication [196], [197]. FANNG-MPC [2] distributes this role across a set of dealers that collaboratively perform the precomputation.

TABLE 4: Categorization of frameworks based on their approach to linear layers evaluation.

Category	Technique	Works
Online-only	SS	[48], [49], [62], [63], [105], [124], [171]
	HE	[61], [78], [88], [92], [107], [111], [117], [129]
Preprocessing / Offline-based	BT-based	[125], [140], [141]
	HE	[86], [123], [142], [162], [193]
	SS	[34], [39], [40], [49], [51], [81], [98], [100], [124], [130], [131], [171]
Dealer/Helper assisted	Dealer	[76], [77], [91], [113], [114], [150], [169]
	Helper	[101], [146], [167], [170], [196], [197]

4.1.2. Matrix Multiplication and Convolution. Matrix multiplication and convolutions can both be represented as structured collections of dot products. This shared structure enables protocol designers to optimize secure computation by reusing building blocks and reducing overhead. Existing MPC protocols can broadly be classified into two categories: purely MPC-based and hybrid MPC–HE solutions.

MPC-based Solutions. SecureML [125] introduced one of the first optimizations in this area by extending Beaver triples (BT) to matrices in 2PC setting, enabling direct matrix multiplication and significantly improving efficiency. This idea of matrix triples has since been widely adopted in SS-based protocols in 3PC and 4PC settings [51], [82], [100], [124].

In the FSS-based domain, Boyle et al. [28] proposed an efficient BT-style matrix multiplication protocol, later incorporated into works such as LLAMA [77], Orca [91], and SIGMA [76]. These protocols achieve low online communication at the cost of a heavy FSS key setup, often relying on a trusted dealer for key distribution.

For convolutions, the standard approach is to reduce them to instances of matrix multiplication [157]. This reduction is widely adopted in MPC-based PPML frameworks, as it simplifies the design and leverages the efficiency of existing secure matrix multiplication protocols. In the preprocessing model, some works further optimize this by extending matrix triples to convolution triples, thereby improving online efficiency [2], [42], [148].

Hybrid MPC–HE Solutions. In this line of work, matrix multiplications and convolutions are performed using HE, either in the preprocessing phase to support MPC-based online computation or directly in the online phase as part of a mixed circuit. In the preprocessing setting, Chen et al. [42] extended SecureML's matrix triple approach to the malicious setting, while Rivinius et al. [148] integrated convolutions into Overdrive [95], the first actively secure MPC protocol with direct convolution support.

Efficient HE usage requires encoding tensor operations into polynomial arithmetic, mainly through two approaches: *SIMD (slot-based)* encoding, which packs multiple elements into ciphertext slots to enable parallel multiplications and slot-rotations [18], [22], [92], [111], [123], [188], [193], and *coefficient encoding*, which embeds inputs and weights directly into polynomial coefficients, often using row- or column-major layouts [78], [83], [88], [117]. Recent works such as Nimbus [107] and Sphinx [162] combine both strategies to balance rotation costs and layout efficiency.

Table 5 summarizes the methods for convolutions, with further details in Appendix D.1.3.

TABLE 5: Categorization of encoding strategies and techniques used for evaluating convolutions.

Tech.	Method	Frameworks
MPC	Matrix multiplication	[51], [98], [124], [130], [170], [171]
	Convolution triples	[2], [42], [148]
HE	SIMD encoding	[18], [22], [92], [111], [123], [188], [193]
	Coefficient encoding	[78], [88], [117], [129]
	Hybrid	[107], [162]

Insight: Choosing the Right Encoding

SIMD encodings preserve layer-wise data layout but incur extra rotations and multiplications, while *coefficient encodings* minimize rotations and per-layer cost, ideal for hybrid MPC–HE pipelines.

4.1.3. Truncation. Most PPML frameworks use Fixed-Point Arithmetic (FPA) to represent real numbers. Since the precision effectively doubles after each multiplication, it is necessary to perform *truncation* to restore the original precision for subsequent computations. Formally, over an arithmetic ring $\mathbb{Z}_{2^\ell}$, truncation for k-bit FPA precision corresponds to division by 2^k, or equivalently, an arithmetic right shift by k bits, as defined in Definition 1.

Definition 1 (Truncation). *Let $x \in \mathbb{Z}$, and write x as $x = x_1 2^k + x_2$, with $x_2 \in [0, 2^k)$. The truncation of x by k bits is defined as $\lfloor x \rfloor = x_1$.*

Truncation is a non-linear operation in MPC and therefore computationally expensive. Unlike activation functions, which appear only occasionally, truncation must be applied after every multiplication to prevent overflow, making efficient protocols critical for ML performance in MPC. While traditionally truncation protocols were combined with multiplication protocols, latest works considered fusing the truncation layer with activation functions or delaying truncations in the context of PPML for efficiency [2], [80].

Truncation protocols are generally classified into three approaches—faithful, stochastic, and probabilistic—which we discuss in detail below. Further details are provided in Appendix D.2, and a comprehensive study can be found in the recent SoK by Harth-Kitzerov et al. [80].

Faithful. Faithful (or precise) truncation [66] produces deterministic, exact results. In the 2PC setting, CryptFlow2 [142] introduced an OT-based faithful truncation protocol, later adapted by several works [74], [141], [199]. Another line of research uses FSS for faithful truncation, first proposed by Boyle et al. [25] and refined in protocols such as LLAMA [77], SIGMA [76], and Shark [75]. Faithful truncation is relatively limited because it is computationally expensive and, for large neural networks, small truncation errors typically do not affect overall model accuracy, making approximate methods sufficient in practice [88].

Stochastic. Stochastic truncation introduces a small error on the least significant bit (LSB), with the probability of error proportional to the magnitude of the truncated part. Formally, $\mathsf{trc}(x) = \lfloor x \rfloor + c$, where c is set with probability $\frac{x_2}{2^k}$, x_2 being the truncated part of x and k the fixed-point precision. These protocols typically use probabilistic techniques, often requiring additional steps to remove the probabilistic behavior. Existing approaches include secret-sharing-based mixed-circuit methods [12], [16], [49], [51], [109], OT-based [74], [88], and FSS-based techniques [91].

Probabilistic. Probabilistic truncation introduces a stochastic error whose likelihood depends on the ratio of the input magnitude to the ring size. Formally, $\mathsf{trc}(x) = \lfloor x \rfloor + c$ with probability $1 - 2^{\ell_x + 1 - \ell}$, where ℓ_x is the bit-length of x and ℓ the ring size. SecureML [125] proposed a non-interactive 2PC protocol, showing that the probabilistic behavior does not significantly affect accuracy for small neural networks. This was later extended to 3PC in ABY3 [124] with preprocessing, and subsequent works improved efficiency by reducing the cost of preprocessing or integrating it with multiplication [34], [40], [98], [100]. MaSTer [185] eliminated the need for preprocessing in ABY3 at the expense of slightly weaker adversarial assumptions.

Security concerns have also been addressed: Li et al. [106] identified issues in earlier 2PC/3PC probabilistic truncation protocols, and Curl [151] proved their security under a custom functionality allowing limited leakage. These insights prompted explicit, formally defined functionalities in later works [185], [190], [198].

Insight: Efficiency vs. accuracy

Faithful and stochastic truncations provide high fidelity but are computationally expensive. In practice, probabilistic truncation is often preferred, striking a balance between efficiency and acceptable model accuracy.

4.2. Non-Linear Layers

Most ML models, including neural networks, use non-linear layers—mainly activation functions—to introduce expressivity. Common activations include ReLU [5] and Sigmoid [47], while Transformers often use more complex components such as Softmax [30] (in attention mechanism) or GELU [84].

$$\mathrm{ReLU}(x) = \max(0, x), \quad \mathrm{Sigmoid}(x) = \frac{1}{1+e^{-x}},$$
$$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}, \quad \mathrm{GELU}(x) = \frac{x}{2}\Big(1 + \mathrm{erf}\big(x/\sqrt{2}\big)\Big)$$

where erf is the Gaussian error function. Other non-linear components are normalization layers, which stabilize training and typically involve element-wise division:

$$\mathrm{LayerNorm}(\vec{x}) = \frac{\vec{x} - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation computed over specific dimensions of $\vec{x}$. In MPC, these operations are generally implemented using square roots and divisions, both of which are computationally expensive.

In the remainder of this section, we focus on activation functions, with particular emphasis on secure comparison, the core building block for most activations. A detailed discussion of our categorization is provided in Appendix D.3.

Mixed-circuit computation. This approach combines Boolean and arithmetic computations to evaluate activation functions [57]. For example, ReLU can be computed via a Boolean comparison $u = (x > 0)$ followed by $\mathrm{ReLU}(x) = u \cdot x$. In the 2PC setting, SecureML [125] adopted this method from ABY using constant-round garbled circuits, which was subsequently employed in several works [39], [146], [174]. Subsequent works in the three and four party setting, such as ABY3 [124] and Trident [40], replaced GCs with non-constant-round Boolean circuits such as Parallel Prefix Adders (PPAs) and their variants, achieving much higher throughput. This Boolean-circuit approach has since been widely adopted [33], [34], [98], [100], [105], [113], [114], [131]. Further improvements leverage edaBits [48], [51], [66] or Rabbit-based conversions [120], [156]. While these methods reduce computation, they often incur higher communication and round complexity.

Garbled Circuits. Here, non-linear layers are evaluated entirely with GCs, yielding constant-round protocols. Early works include MiniONN [111], DeepSecure [149], and Gazelle [92] in the 2PC setting. Ball et al. [17] demonstrated the practicality of garbled NNs in the multiparty setting. To reduce GC usage, Delphi [123] adopted a hybrid approach that approximates some ReLUs with the square function. Subsequent works such as Muse [103], SIMC [38], and SIMC2.0 [174] extended this paradigm to client-malicious security in the 2PC client-server model. More recently, APINT [44] accelerated GC-based transformer models, highlighting their continued relevance.

Arithmetic Secret Sharing. This line of work relies solely on arithmetic secret sharing for non-linear layers, avoiding conversions to Boolean circuits [62], [101], [170], [171]. While this reduces communication overhead, it typically incurs higher round complexity. In an orthogonal direction, some works exploit truncation as a comparison primitive to improve both communication and rounds, albeit requiring a semi-honest helper party [196], [197].

Look-Up Tables (LUTs). CryptFlow2 [142] introduced a 2PC secure comparison protocol based on Garay et al. [72], decomposing the comparison into a tree of smaller blocks and evaluating them via 1-out-of-2^m OTs in a LUT-style manner [59]. SirNN [141] extended this to more complex functions such as exponentiation, sigmoid, tanh, and inverse square root. Subsequent works adapted LUTs for floating-point computations [139], [140]. A key drawback is the communication cost, which grows linearly with both input and output sizes. Recent advances such as FLUTE [32] and ROTL [87] indicate that LUT-based approaches could soon be more practical.

Function Secret Sharing. AriaNN [150] introduced FSS-based comparison in the 2PC semi-honest setting with a trusted dealer. Later works [76], [77], [91] extended support to complex math functionalities for activations and improved efficiency. Pika [169] realized maliciously secure three-party FSS, while Shark [75] proposed 2PC maliciously secure protocols with a semi-honest dealer. FSS incurs higher computation and large preprocessing keys but benefits from constant-round online phases. Recent enhancements include extension to 3PC [43], GPU acceleration [91] and distributed key generation [173], [177], thereby removing the need for a trusted dealer.

Hybrid MPC–HE. Fregata [179] employs a hybrid approach, preprocessing non-linear layers using HE and integrating them into the online phase with arithmetic secret sharing. In an orthogonal research, mainly in client-server setting, approximate non-linear activations with piecewise polynomials, evaluating each polynomial via HE while using MPC-based comparisons to determine the appropriate polynomial range [60], [107], [117], [129].

Plaintext evaluation. Recent works such as PermLLM [192] and Fission [166] evaluate non-linear functions non-cryptographically by applying random permutations and computing activations on clear data. While this approach is shown to be statistically secure, it lacks formal cryptographic guarantees, demanding further investigation. Sphinx [162] on the other hand, enables the client evaluate activation functions locally by revealing output of each layer.

Insight: Trade-offs in Non-Linear Layers

Securely evaluating non-linear layers involves balancing round complexity, communication, computation, storage, and accuracy. The most efficient frameworks combine multiple cryptographic techniques judiciously to optimize these trade-offs.

TABLE 6: Classification of frameworks according to their approaches for non-linear layer evaluation. Notation and abbreviations are defined in Table 2.

Approach	Method	Frameworks
Mixed	A-G	[39], [125], [146], [174]
	A-B	[33], [34], [40], [98], [100], [105], [113], [114], [124], [131]
GCs	Exact	[17], [38], [44], [92], [103], [111], [123], [149]
ASS	Quotient Transfer	[62], [101], [170], [171]
	Truncation-based	[196], [197]
LUTs	OTs	[88], [139], [140], [141], [142]
FSS		[75], [76], [77], [91], [169], [177]
MPC-HE	HE offline	[179]
	Piecewise Pol.	[60], [107], [117], [129]
Plaintext	Permutation	[166], [192]
	Client-side	[162]

5. The MPC Puzzle: Unifying Secret-Sharing and Conversions

Our survey of existing MPC-based PPML frameworks reveals that many of them employ similar secret-sharing semantics. However, these similarities often go unnoticed due to differences in notation, terminology, and presentation styles, which hinder direct comparison across works. This inconsistency not only makes it challenging for new readers to understand the true contributions of each work relative to prior works, but also prevents researchers from identifying existing optimizations that could be readily adapted from other systems employing equivalent sharing semantics.

A unified representation of these semantics offers two key benefits: it clarifies the conceptual relationships among various MPC approaches and it simplifies the design of

TABLE 7: Overview of the secret sharing schemes considered in this work. For improved readability, some schemes are presented in a unified form without altering their semantics. Notations: a - additive, r - replicated, m - masked. Threat model: ○- semi-honest, ●- malicious. Abbreviations: Not. - Notations, Sec. - Security.

Not.	Secret Share Distribution				Semantics	Sec.	Works
	P_1	P_2	P_3	P_4			
$[x]^{\mathsf{a}}$	x_1	x_2	-	-	$x = x_1 + x_2$	○	[57], [88], [111], [125], [141], [145], [146]
$[x]^{\mathsf{a},1}$	(x_1, Δ_1)	(x_2, Δ_2)	-	-	$x = x_1 + x_2$, $\Delta = \alpha \cdot x$, $\Delta = \Delta_1 + \Delta_2$	●	[183][1,2]
$[x]^{\mathsf{m}}$	(m_x, λ_x^1)	(m_x, λ_x^2)	-	-	$x = m_x - \lambda_x$; $\lambda_x = \lambda_x^1 + \lambda_x^2$	○	[130]
$[\![x]\!]^{\mathsf{a}}$	x_1	x_2	x_3	-	$x = x_1 + x_2 + x_3$	○	
$[\![x]\!]^{\mathsf{r}}$	(x_1, x_2)	(x_2, x_3)	(x_3, x_1)	-	$x = x_1 + x_2 + x_3$	○,●	[124], [171]
$[\![x]\!]^{\mathsf{r},1}$	$(x_1, x_2, \Delta_1, \Delta_2)$	$(x_2, x_3, \Delta_2, \Delta_3)$	$(x_3, x_1, \Delta_3, \Delta_1)$	-	$x = x_1 + x_2 + x_3$, $\Delta = \alpha \cdot x$, $\Delta = \Delta_1 + \Delta_2 + \Delta_3$	●	[51][1,2]
$[\![x]\!]^{\mathsf{m}}$	$(m_x, \lambda_x^1, \lambda_x^2)$	$(m_x, \lambda_x^2, \lambda_x^3)$	$(m_x, \lambda_x^3, \lambda_x^1)$	-	$x = m_x - \lambda_x$; $\lambda_x = \lambda_x^1 + \lambda_x^2 + \lambda_x^3$	●	[33], [62], [98], [131][2]
$[\![x]\!]^{\mathsf{m},1}$	(m_x, λ_x^1)	(m_x, λ_x^2)	$(\lambda_x^1, \lambda_x^2)$	-	$x = m_x - \lambda_x$; $\lambda_x = \lambda_x^1 + \lambda_x^2$	○	[33], [39]
$[\![x]\!]^{\mathsf{m},2}$	$(m_{x,2}, \lambda_x^1)$	$(m_{x,1}, \lambda_x^2)$	$(\lambda_x^1, \lambda_x^2)$	-	x = $m_{x,1} - \lambda_x^1$; $x = m_{x,2} - \lambda_x^2$	○	[81]
$\langle\!\langle x \rangle\!\rangle^{\mathsf{a}}$	x_1	x_2	x_3	x_4	$x = x_1 + x_2 + x_3 + x_4$	○	
$\langle\!\langle x \rangle\!\rangle^{\mathsf{a},1}$	x_1	x_1 or x_2	x_2 or x_1	x_2	$x = x_1 + x_2$	○	[48]
$\langle\!\langle x \rangle\!\rangle^{\mathsf{r}}$	(x_1, x_2, x_3)	(x_2, x_3, x_4)	(x_3, x_4, x_1)	(x_4, x_1, x_2)	$x = x_1 + x_2 + x_3 + x_4$	●	[51]
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m}}$	$(m_x, \lambda_x^1, \lambda_x^2, \lambda_x^3)$	$(m_x, \lambda_x^2, \lambda_x^3, \lambda_x^4)$	$(m_x, \lambda_x^3, \lambda_x^4, \lambda_x^1)$	$(m_x, \lambda_x^4, \lambda_x^1, \lambda_x^2)$	$x = m_x - \lambda_x$; $\lambda_x = \lambda_x^1 + \lambda_x^2 + \lambda_x^3 + \lambda_x^4$	●	
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m},1}$	$(m_x, \lambda_x^1, \lambda_x^2)$	$(m_x, \lambda_x^2, \lambda_x^3)$	$(m_x, \lambda_x^3, \lambda_x^1)$	$(\lambda_x^1, \lambda_x^2, \lambda_x^3)$	$x = m_x - \lambda_x$; $\lambda_x = \lambda_x^1 + \lambda_x^2 + \lambda_x^3$	●	[40], [81], [98], [100][3]
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m},2}$	$(m_{x,1}, m_{x,2}, \lambda_x^1)$	$(m_{x,1}, m_{x,2}, \lambda_x^2)$	$(m_{x,1}, \lambda_x^1, \lambda_x^2)$	$(m_{x,2}, \lambda_x^1, \lambda_x^2)$	x = $m_x - \lambda_x$; $\lambda_x = \lambda_x^1 + \lambda_x^2$; $m_x = m_{x,1} + m_{x,2}$	●	[34]

[1] α - global MAC key. [2] Requires operation over larger ring $\mathbb{Z}_{2^{\ell+s}}$ for statistical security parameter s.
[3] Used the unified sharing semantics for [98], [131] from [158].

secret-sharing conversions, thereby enabling more flexible and hybrid architectures, as discussed in Section 5.1.

The unified semantics proposed in this work are summarized in Table 7. We consider settings with up to four parties, denoted by P_1, P_2, P_3, and P_4, and describe how a secret value x is shared among them. Our discussion includes three widely used secret-sharing approaches: additive (a), replicated (r), and masked (m) sharing, applied across two, three, and four-party protocols. We begin with a brief overview of each approach below. For simplicity, we assume all operations are performed over an ℓ-bit ring $\mathbb{Z}_{2^\ell}$ and we use $\langle \cdot \rangle$ to denote a generic secret sharing.

- *Additive sharing* ($\langle \cdot \rangle^{\mathsf{a}}$): A value $x \in \mathbb{Z}_{2^\ell}$ is said to be *additively shared* among n parties $P_1, \ldots, P_n$ if each party P_i holds a share $x_i \in \mathbb{Z}_{2^\ell}$ such that $x = \sum_{i=1}^{n} x_i$.
- *Replicated sharing* ($\langle \cdot \rangle^{\mathsf{r}}$): A value $x \in \mathbb{Z}_{2^\ell}$ is said to be in *replicated sharing* among n parties with a threshold t such that $n > 2t + 1$, if x is split into $q = \binom{n}{t}$ additive shares $x_1, \ldots, x_q$ such that $x = \sum_{i=1}^{q} x_i$ and x_i is given to all the parties in the complementary set corresponding to x_i. This results in each share being held by multiple parties, enabling reconstruction of x among subsets of size $t + 1$ or more.
- *Masked sharing* ($\langle \cdot \rangle^{\mathsf{m}}$): A value $x \in \mathbb{Z}_{2^\ell}$ is said to be in *masked sharing* among n parties if there exist values $m_x, \lambda_x \in \mathbb{Z}_{2^\ell}$ such that $m_x = x + \lambda_x$, where m_x is a masked value known to all parties, and λ_x is a random mask that is secret-shared among the parties using either additive ($\langle \lambda_x \rangle^{\mathsf{a}}$) or replicated ($\langle \lambda_x \rangle^{\mathsf{r}}$) sharing.

Although the secret-sharing schemes listed in Table 7 broadly fall under the three categories discussed above, several works introduce semantic variations within each category that contribute to improved protocol efficiency. We annotate such variants using numeric superscripts. In some cases, the share distributions have been slightly modified during unification for presentation clarity; however, these adjustments do not affect the underlying semantics. Further details on these modifications are provided in Appendix E.1.

5.1. The Need for Share Conversion

While unifying the various secret-sharing schemes, we identified a key gap in the literature: the absence of a general mechanism for converting between different sharing semantics. Several works analyzed in our study introduce efficient protocols for cryptographic building blocks that are of independent interest. In addition, many works focus on optimizing specific sub-protocols, which could potentially replace corresponding components within existing frameworks. However, most of these protocols adopt a fixed sharing scheme—producing outputs under the same semantic—which inherently limits interoperability and reuse across systems. In this section, we elaborate on several representative cases identified in the MPC-based PPML literature where systematic share conversions play a critical role.

5.1.1. Hybrid Architectures. Efficient share conversions can significantly enhance PPML protocols that employ hybrid architectures, where different components of a model are evaluated under distinct MPC paradigms. For instance, Matchmaker [90] integrates Linear Secret Sharing (LSS) and FSS to enable fast inference across heterogeneous CPU–GPU deployments. Its FSS component, based on additive secret sharing, supports constant-round evaluation of non-linear functions. In contrast, ABY2.0 [130] adopts masked secret sharing to achieve vector-size-independent, efficient dot-product computation. Although both frameworks target the two-party semi-honest setting, their differing sharing semantics—additive in Matchmaker versus masked in ABY2.0—make it non-trivial to combine their respective advantages. Similarly, Cheetah [88] and its follow-up works [69] leverage FHE for linear layers and MPC for non-linear layers. Replacing the additive secret sharing in Cheetah for the non-linear layers with the masked sharing from ABY2.0 could enable multi-input Boolean gate evaluation, thereby improving its online performance.

5.1.2. MPC-as-a-Service. In the *MPC-as-a-Service* paradigm, a designated set of dealers precompute correlated randomness that can later be consumed by the online MPC parties to accelerate computation. For example, TaaS [155] provides the generation of Beaver triples, enabling efficient secure multiplications. Moreover, this paradigm often benefits from amortization, as large batches of correlated materials can be processed in a single step, thereby reducing the overall cost. This model is particularly valuable in PPML settings, where computations tend to be heavy. The multi-party PPML framework FANNG-MPC [2], for instance, employs a separate group of dealers to perform preprocessing using FHE, with the resulting materials consumed during the online MPC phase. Since the preprocessing and online phases may rely on different sharing semantics, efficient share conversion mechanisms are essential to ensure compatibility between these components.

5.1.3. Committee-based Protocols. This class of protocols typically employs a committee by sampling a subset of clients to execute specific stages of computation and subsequently share the results with the remaining participants. Another variant groups clients into clusters, where each cluster performs computations over its collective data, and the resulting outputs are later aggregated. For example, in the context of FL, WW-FL [122] combines MPC clusters instantiated under different secret-sharing schemes within a federated learning architecture to enable scalable and efficient global training. Such committee-based designs are particularly effective for handling heterogeneous client data in FL settings.

5.1.4. Dynamic Participants. Protocols supporting dynamic participants allow the set of computing parties to change over time—either by completely replacing the participants between rounds or by enabling parties to leave and later rejoin the computation. The Fluid MPC framework [45] and its follow-ups form an example of the latter model. In both scenarios, share conversion plays a crucial role in securely transferring the private information collectively held by one set of parties to another.

A related class of works targets stronger security models by adapting player elimination, wherein the protocol continues with a smaller subset of parties upon detecting corruption. For example, the 4PC protocol in Fantastic Four [51] transitions to a 3PC protocol with active security when a malicious behavior is detected.

Overlooking Conversions. Another strong motivation for a systematic analysis of share conversions is that, while such conversions may appear straightforward, they are often error-prone in the absence of a formal understanding. As an example, we revisit the conversion from replicated to masked secret sharing in the 3PC setting from Rep3 Reloaded [94].

Replicated secret sharing[1] assigns party P_i the share (x_{i-1}, x_{i+1}). For the parties P_0, P_1, P_2, this translates to

$$P_0 : (x_2, x_1), \quad P_1 : (x_0, x_2), \quad P_2 : (x_1, x_0).$$

The authors then present a mapping to the masked secret sharing used in ASTRA [39], where for $m_x = x + \lambda_x^1 + \lambda_x^2$, the mapping is defined as

$$m_x = x_0, \quad \lambda_x^1 = -x_2, \quad \lambda_x^2 = -x_1.$$

While the conversion is semantically correct, the proposed method overlooks key advantages of masked sharing. In particular, masked sharing leverages the availability of input-independent λ_x shares in the preprocessing phase to

1. We follow the notation of Rep3 Reloaded [94] and correct a typographical error in their preprint: P_1 holds (x_0, x_2) rather than (x_0, x_1).

prepare preprocessed data. With the proposed conversion, however, the λ_x shares are only available online, since all x_i shares are defined during the online phase. We provide an effective conversion between these schemes, which is discussed later in Section 5.2.

These use cases highlight the need for mechanisms to convert between secret-sharing schemes. Towards this, we propose *MPC Puzzle*, a transformation layer supporting efficient conversions across multiple secret-sharing semantics. We begin by analyzing a simple, broadly applicable approach to share conversion.

Naive Approach. To convert secret shares from a source s, denoted by $\langle\cdot\rangle^{\mathsf{s}}$, to a destination d, denoted by $\langle\cdot\rangle^{\mathsf{d}}$, a straightforward approach is to *re-share the secret-shares* [2], [122]. Specifically, each share under $\langle\cdot\rangle^{\mathsf{s}}$ is treated as a new input to the input-sharing protocol corresponding to $\langle\cdot\rangle^{\mathsf{d}}$. Once the re-sharing is complete, the resulting shares of the original secret can be obtained by applying a linear combination consistent with the semantics of $\langle\cdot\rangle^{\mathsf{s}}$.

As an example, consider a 3PC additive sharing $\langle x\rangle^{\mathsf{a}} = (x_1, x_2, x_3)$ such that $x = x_1 + x_2 + x_3$. To convert this to a replicated sharing $\langle y\rangle^{\mathsf{r}}$ of the same value $y = x$, each party P_i invokes the input-sharing protocol for $\langle\cdot\rangle^{\mathsf{r}}$ to share its additive share x_i, producing $\langle x_i\rangle^{\mathsf{r}}$. Then, each party locally computes the replicated share of y by summing the individual replicated shares: $\langle y\rangle^{\mathsf{r}} = \langle x_1\rangle^{\mathsf{r}} + \langle x_2\rangle^{\mathsf{r}} + \langle x_3\rangle^{\mathsf{r}}$.

Note that this approach applies to most secret-sharing schemes due to their linearity, though it may incur non-optimal costs in practice.

Insight: Naive Approach

Naive conversion, which re-shares inputs and linearly combines results, works across different sets of parties but is often inefficient for conversions among the same set.

We next present conversions among the standard variants of the three secret-sharing schemes in the three-party setting (see Section 5.2, assuming a fixed set of parties. The analysis naturally extends to two- and four-party settings. This is followed by conversions involving different sets of parties (specifically, two and three) in Section 5.3. Additionally, we assume a PRF-based shared key-setup among the parties, enabling them to sample random values non-interactively [11], [158]. For simplicity, we use the notations P_{i+1} and P_{i-1} to denote the parties in a circular sequence, where P_1 comes after P_n. For completeness, remaining conversion protocols in different party settings for standard secret schemes listed in Table 7 are provided in Section E.1.

5.2. Standard 3PC Secret-Share Conversions

We focus on conversions among $[\![\cdot]\!]^{\mathsf{a}}$, $[\![\cdot]\!]^{\mathsf{r}}$, and $[\![\cdot]\!]^{\mathsf{m}}$ shares of a secret $x \in \mathbb{Z}_{2^\ell}$ in a 3PC setting with a fixed set of parties. This section provides baseline intuitions for the conversions; exact protocols depend on additional factors such as sharing semantics and security, and are detailed in Section E.1.

For ease of presentation, we use y to denote the converted secret, which has the same value as x. Parties utilizes the non-interactive Π_{zero} protocol from the literature [11], [39], which enables additive sharing of zero using a shared-key setup.

$[\![x]\!]^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{r}}$. To convert $[\![x]\!]^{\mathsf{a}}$, where P_i holds x_i such that $x = x_1 + x_2 + x_3$, into $[\![y]\!]^{\mathsf{r}}$, each P_i sends x_i to P_{i-1}, requiring 3ℓ bits of communication over $\mathbb{Z}_{2^\ell}$. Each P_i then locally sets $y_i = x_i$ and $y_{i+1} = x_{i+1}$.

$[\![x]\!]^{\mathsf{r}} \rightarrow [\![y]\!]^{\mathsf{a}}$. This is a local conversion, as an (n, t) replicated sharing inherently contains an additive sharing among any subset of size $t+1$ or more. In the 3PC honest-majority setting, any two parties together hold complete information about the secret x. Here, each P_i can locally set $y_i = x_i$ to complete the conversion.

$[\![x]\!]^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{m}}$. This is interactive, as computing the input-dependent term m_y requires knowledge of $y = x$, while x is additively shared among P_1, P_2, and P_3.

During preprocessing, parties non-interactively generate their λ_y shares using the shared-key setup as follows:

$$P_1, P_3 : \lambda_y^1, \quad P_1, P_2 : \lambda_y^2, \quad P_2, P_3 : \lambda_y^3$$

In the online phase, parties reconstruct $m_y = y + \lambda_y$ by having each P_i send $x_i + \lambda_y^i + \zeta_i$ to the other two parties, where ζ_i denotes the additive share of zero obtained by P_i from the Π_{zero} protocol, as discussed above.

This requires 6ℓ bits of communication in total. The communication can be reduced to 4ℓ bits at the cost of one additional round, by using the standard approach of sending shares to a designated party and receiving the reconstructed value from that party [55], [99].

$[\![x]\!]^{\mathsf{m}} \rightarrow [\![y]\!]^{\mathsf{a}}$. This is a non-interactive protocol, similar to the $[\![x]\!]^{\mathsf{r}} \rightarrow [\![y]\!]^{\mathsf{a}}$ conversion. The λ_x shares are already in the $[\![\cdot]\!]^{\mathsf{r}}$ form and inherently possess an additive structure.

To complete the conversion from $[\![x]\!]^{\mathsf{m}}$, a designated party—say, P_1—sets $y_1 = m_x - \lambda_x^1$, while the other parties P_i set $y_i = -\lambda_x^i$. Correctness follows directly, as:

$$y = y_1 + y_2 + y_3 = (m_x - \lambda_x^1) - \lambda_x^2 - \lambda_x^3 = m_x - \lambda_x = x.$$

$[\![x]\!]^{\mathsf{r}} \rightarrow [\![y]\!]^{\mathsf{m}}$. This conversion is analogous to the $[\![x]\!]^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{m}}$ protocol. In the preprocessing phase, parties non-interactively generate $[\![\lambda_y]\!]^{\mathsf{r}}$ shares using the shared-key setup. Then, in the online phase, parties publicly reconstruct $m_y = y + \lambda_y$. Since x is $[\![\cdot]\!]^{\mathsf{r}}$-shared, the communication for reconstruction is more efficient: each P_i receives $x_{i-1} + \lambda_y^{i-1} + \zeta_{i-1}$ from P_{i-1}, requiring only 3ℓ bits in total.

One might consider avoiding preprocessing by assigning certain x_i shares to λ_y^j shares on the fly, thereby constructing a valid $[\![y]\!]^{\mathsf{m}}$ sharing directly during the online phase [94]. However, this approach is incompatible with the standard masked sharing semantics, as λ_y^j shares would only be defined during online phase. As a result, the preprocessing phase—central to the efficiency of masked secret sharing—would be lost, defeating one of the primary advantages of using $[\![\cdot]\!]^{\mathsf{m}}$ shares.

$[\![x]\!]^{\mathsf{m}} \rightarrow [\![y]\!]^{\mathsf{r}}$. Since the λ_x shares are already in $[\![\cdot]\!]^{\mathsf{r}}$ form, this conversion can be performed locally. Let two parties—say, P_1 and P_3—hold the common replicated share $y_1 = m_x - \lambda_x^1$. The remaining shares are set as $y_2 = \lambda_x^2$ and $y_3 = \lambda_x^3$. Correctness follows directly, analogous to the $[\![x]\!]^{\mathsf{m}} \rightarrow [\![y]\!]^{\mathsf{a}}$ conversion.

5.3. 2PC-3PC Secret-Share Conversions

We now focus on standard conversions between 2PC and 3PC settings. Conversions from 3PC to 2PC arise in robust protocols that utilize player elimination approach upon corruption detection [51], while 2PC to 3PC conversions are common in committee-based protocols such as FANNG-MPC [2] and WW-FL [122], where secrets move across party groups. These conversions also help bridge techniques between 2PC and 3PC—for example, extending 2PC probabilistic truncation from SecureML [125] to other settings [39], [40], [124], [185].

In the remainder of this section, we consider the conversion of a secret x held by parties $\mathcal{P} = \{P_i\}$ to a new sharing of the same secret y among a different party set $\mathcal{P}' = \{P'_j\}$.

$\mathcal{P} : [x]^{\mathsf{a}} \rightarrow \mathcal{P}' : [\![y]\!]^{\mathsf{a}}$. This conversion is performed via a standard re-sharing approach. Each $P_i \in \{P_1, P_2\}$ splits its share x_i into three additive shares: $x_i = x_{i,1} + x_{i,2} + x_{i,3}$, and sends $x_{i,j}$ to party $P'_j \in \{P'_1, P'_2, P'_3\}$. Then, each P'_j locally computes $y_j = x_{1,j} + x_{2,j}$.

$\mathcal{P} : [x]^{\mathsf{a}} \rightarrow \mathcal{P}' : [\![y]\!]^{\mathsf{r}} / [\![y]\!]^{\mathsf{m}}$. This conversion is similar to the one above, except that each $P_i \in \{P_1, P_2\}$ generates a $[\![x_i]\!]^{\mathsf{r}}$ or $[\![x_i]\!]^{\mathsf{m}}$ sharing of its input. Each party $P'_j \in \{P'_1, P'_2, P'_3\}$ then locally adds the shares it receives from all parties in $\mathcal{P}$ to obtain its share of y.

$\mathcal{P} : [x]^{\mathsf{m}} \rightarrow \mathcal{P}' : [\![y]\!]^{\mathsf{a}} / [\![y]\!]^{\mathsf{r}} / [\![y]\!]^{\mathsf{m}}$. This conversion is analogous to the case where x is shared as $[x]^{\mathsf{a}}$, since parties in $\mathcal{P}$ can locally derive an additive sharing of x from $[x]^{\mathsf{m}}$ by setting $x_1 = m_x - \lambda_x^1$ and $x_2 = -\lambda_x^2$. The resulting additive shares x_1 and x_2 can then be re-shared to $\mathcal{P}'$ using the appropriate conversion protocol, as in previous cases.

$\mathcal{P} : [\![x]\!]^{\mathsf{a}} \rightarrow \mathcal{P}' : [y]^{\mathsf{a}}$. This conversion can be viewed as the reverse of the $[x]^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{a}}$ conversion discussed above. Each party $P_j \in \{P_1, P_2, P_3\}$ generates a $[\cdot]^{\mathsf{a}}$ sharing of its additive share x_j, and each $P'_i \in \{P'_1, P'_2\}$ locally adds the shares received from all P_j to obtain its share of $[y]^{\mathsf{a}}$.

$\mathcal{P} : [\![x]\!]^{\mathsf{a}} \rightarrow \mathcal{P}' : [y]^{\mathsf{m}}$. Unlike the conversion discussed above, this is a two-step process. First, parties apply the $[\![x]\!]^{\mathsf{a}} \rightarrow [y]^{\mathsf{a}}$ conversion to generate a $[y]^{\mathsf{a}}$ sharing among $\mathcal{P}'$. Then, parties in $\mathcal{P}'$ perform a $[y]^{\mathsf{a}} \rightarrow [y]^{\mathsf{m}}$ conversion, following steps analogous to the $[\![y]\!]^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{m}}$ protocol described in Section 5.2.

$\mathcal{P} : [\![x]\!]^{\mathsf{r}} / [\![x]\!]^{\mathsf{m}} \rightarrow \mathcal{P}' : [y]^{\mathsf{a}} / [y]^{\mathsf{m}}$. Since parties in $\mathcal{P}$ can locally convert their $[\![x]\!]^{\mathsf{r}}$ or $[\![x]\!]^{\mathsf{m}}$ shares into additive shares $[\![x]\!]^{\mathsf{a}}$ (cf. Section 5.2), this conversion reduces to the corresponding $[\![x]\!]^{\mathsf{a}} \rightarrow [y]^{\mathsf{a}}$ or $[\![x]\!]^{\mathsf{a}} \rightarrow [y]^{\mathsf{m}}$ conversions discussed above.

Insight: Share Conversions

Conversions from additive shares or to masked shares require interaction, even within the same party set.

6. Summary and Insights

No single MPC-based PPML protocol can simultaneously optimize for all dimensions—deployment model, adversarial setting, and performance objectives. Techniques that excel in one dimension, such as high-throughput inference, often incur prohibitive costs in another, such as low-latency execution or robustness against stronger adversaries. Similarly, linear and non-linear layers demand distinct algebraic domains and primitives, and efficient frameworks must combine multiple protocols, switching between arithmetic, Boolean, and field-based sharing as needed. Such designs critically rely on fast and secure conversion mechanisms between representations, enabling each layer to be executed in its most efficient domain.

Our systematization reveals several important insights. Design trade-offs are unavoidable: algebraic structures, threat models, and network assumptions fundamentally shape efficiency, motivating hybrid approaches. Comparison operations emerge as persistent bottlenecks, often dominating performance costs. Interoperability remains central, as efficient conversions between secret-sharing schemes are key to enabling flexible and composable frameworks. Support for advanced ML models such as transformers and LLMs has grown, but integration into PPML workflows is still limited, particularly when relying on techniques such as FSS or additive sharing. Finally, the choice of security model and functionality definition has profound implications, influencing both the strength of guarantees and the achievable efficiency [151].

Emerging Trends. We observe several promising research directions. First, eliminating trusted dealers in Function Secret Sharing and improving its integration with linear secret sharing schemes are active areas, aiming to balance efficiency and practicality. Second, malicious security is gaining traction, with new protocols seeking stronger guarantees without prohibitive overhead. Third, frameworks are increasingly optimizing for throughput in inference workloads, though lightweight designs for low-latency, single-query settings remain open. Fourth, the growing focus on very large models such as Transformers is driving the need for scalable MPC protocols, efficient handling of non-linearities, and new trade-offs between accuracy, efficiency, and security. Finally, genealogies of PPML frameworks reveal a trend toward composability: future systems are likely to be built from reusable low-level components rather than monolithic designs.

Artifacts

Given the large number of research works covered in our SoK, we have included detailed comparisons—such as concrete performance costs of PPML primitives—in an open-source repository available at https://anonymous.4open.science/r/sok-ppml-mpc-frameworks-922. This repository will be maintained to reflect future developments in the field, allowing researchers to stay up to date.

Ethics Considerations

We have carefully reviewed the call for papers, submission, and ethics guidelines. We confirm that our study does not involve sensitive information or pose any potential risks. Instead, our research contributes to the design of secure and privacy-preserving machine learning systems.

References

[1] Najwa Aaraj, Abdelrahaman Aly, Alvaro Garcia-Banda, Chiara Marcolla, Victor Sucasas, and Ajith Suresh. Privacy-Preserving Machine Learning for Massive IoT Deployments. *Security and Privacy for 6G Massive IoT*, 2025.

[2] Najwa Aaraj, Abdelrahaman Aly, Tim Güneysu, Chiara Marcolla, Johannes Mono, Rogerio Paludo, Iván Santos-González, Mireia Scholz, Eduardo Soria-Vazquez, Victor Sucasas, and Ajith Suresh. FANNG-MPC: Framework for Artificial Neural Networks and Generic MPC. *CHES*, 2025.

[3] Aysajan Abidin, Enzo Marquet, Jerico Moeyersons, Xhulio Limani, Erik Pohle, Michiel Van Kenhove, Johann M. Márquez-Barja, Nina Slamnik-Krijestorac, and Bruno Volckaert. MOZAIK: An End-to-End Secure Data Sharing Platform. In *DEC*, 2023.

[4] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Computing Surveys*, 2018.

[5] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *CoRR*, abs/1803.08375, 2018.

[6] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In *CCS*, 2019.

[7] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. Privformer: Privacy-preserving Transformer with MPC. In *EuroS&P*, 2023.

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

[9] Corinne G. Allaart, Saba Amiri, Henri E. Bal, Adam Belloum, Leon Gommans, Aart van Halteren, and Sander Klous. Private and Secure Distributed Deep Learning: A Survey. *ACM Comput. Surv.*, 57(4), 2025.

[10] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier. In *S&P*, 2017.

[11] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *CCS*, 2016.

[12] Nuttapong Attrapadung, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Takahiro Matsuda, Ibuki Mishina, Hiraku Morita, and Jacob C. N. Schuldt. Adam in Private: Secure and Fast Training of Deep Neural Networks with Adaptive Moment Estimation. *PETS*, 2022.

[13] Monir Azraoui, Muhammad Barham, Beyza Bozdemir, Sébastien Canard, Eleonora Ciceri, Orhan Ermis, Ramy Masalha, Marco Mosconi, Melek Önen, Marie Paindavoine, Boris Rozenberg, Bastien Vialla, and Sauro Vicini. SoK: Cryptography for Neural Networks. In *Privacy and Identity Management. Data for Better Living: AI and Privacy - IFIP WG*, 2019.

[14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[15] Alessandro N. Baccarini, Marina Blanton, and Chen Yuan. Multi-Party Replicated Secret Sharing over a Ring with Applications to Privacy-Preserving Machine Learning. *PETS*, 2023.

[16] Alessandro N. Baccarini, Marina Blanton, and Chen Yuan. Multi-Party Replicated Secret Sharing over a Ring with Applications to Privacy-Preserving Machine Learning. *PETS*, 2023.

[17] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled Neural Networks are Practical. *IACR Cryptol. ePrint Arch.*, 338, 2019.

[18] Shashank Balla and Farinaz Koushanfar. HELiKs: HE Linear Algebra Kernels for Secure Inference. In *CCS*, 2023.

[19] Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. SoK: Privacy-Enhancing Technologies in Finance. In *AFT*, 2023.

[20] Donald Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO*, 1991.

[21] Sebastian Becker, Christoph Bösch, Benjamin Hettwer, Thomas Hoeren, Merlin Rombach, Sven Trieflinger, and Hossein Yalame. Multi-Party Computation in Corporate Data Processing: Legal and Technical Insights. *IACR Cryptol. ePrint Arch.*, 463, 2025.

[22] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: A mixed-protocol machine learning framework for private inference. In *ARES*, 2020.

[23] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. In *CRYPTO*, 2019.

[24] Amine Boulemtafes, Abdelouahid Derhab, and Yacine Challal. A review of privacy-preserving techniques for deep learning. *Neurocomputing*, 384, 2020.

[25] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation. In *EUROCRYPT*, 2021.

[26] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing. In *EUROCRYPT*, 2015.

[27] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *CCS*, 2016.

[28] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure Computation with Preprocessing via Function Secret Sharing. In *TCC*, 2019.

[29] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs. In *CCS*, 2019.

[30] John S. Bridle. Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters. In *NeurIPS*, 1989.

[31] Andreas Brüggemann, Robin Hundt, Thomas Schneider, Ajith Suresh, and Hossein Yalame. FLUTE: Fast and Secure Lookup Table Evaluations. In *IEEE S&P*, 2023.

[32] Andreas Brüggemann, Robin Hundt, Thomas Schneider, Ajith Suresh, and Hossein Yalame. FLUTE: Fast and Secure Lookup Table Evaluations. In *S&P*, 2023.

[33] Andreas Brüggemann, Oliver Schick, Thomas Schneider, Ajith Suresh, and Hossein Yalame. Don't Eject the Impostor: Fast Three-Party Computation With a Known Cheater. In *S&P*, 2024.

[34] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *PETS*, 2020.

[35] José Cabrero-Holgueras and Sergio Pastrana. SoK: Privacy-Preserving Computation Techniques for Deep Learning. *PETS*, 2021.

[36] Octavian Catrina and Sebastiaan de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *SCN*, 2010.

[37] Octavian Catrina and Amitabh Saxena. Secure Computation with Fixed-Point Numbers. In *FC*, 2010.

[38] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. SIMC: ML Inference Secure Against Malicious Clients at Semi-Honest Cost. In *USENIX Security*, 2022.

[39] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In *CCSW@CCS*, 2019.

[40] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS*, 2020.

[41] Congcong Chen, Lifei Wei, Jintao Xie, and Yang Shi. Privacy-Preserving Machine Learning Based on Cryptography: A Survey. *ACM Trans. Knowl. Discov. Data*, 2025. Just Accepted.

[42] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In *ASIACRYPT*, 2020.

[43] Ke Cheng, Yuheng Xia, Anxiao Song, Jiaxuan Fu, Wenjie Qu, Yulong Shen, and Jiaheng Zhang. Mosformer: Maliciously Secure Three-Party Inference Framework for Large Transformers. *IACR Cryptol. ePrint Arch.*, 2025.

[44] Hyunjun Cho, Jaeho Jeon, Jaehoon Heo, and Joo-Young Kim. APINT: A Full-Stack Framework for Acceleration of Privacy-Preserving Inference of Transformers based on Garbled Circuits. In *ICCAD*, 2024.

[45] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure Multiparty Computation with Dynamic Participants. In *CRYPTO*, 2021.

[46] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority. In *CRYPTO*, 2018.

[47] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 1989.

[48] Tianxiang Dai, Li Duan, Yufan Jiang, Yong Li, Fei Mei, and Yulian Sun. Force: Making 4PC $> 4 \times$ PC in Privacy Preserving Machine Learning on GPU. *IACR Cryptol. ePrint Arch.*, 493, 2023.

[49] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure Evaluation of Quantized Neural Networks. *PETS*, 2020.

[50] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure Evaluation of Quantized Neural Networks. *PETS*, 2020.

[51] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security. In *USENIX Security*, 2021.

[52] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In *S&P*, 2019.

[53] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In *S&P*, 2019.

[54] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Homomorphic encryption and secure comparison. *Int. J. Appl. Cryptogr.*, 1(1), 2008.

[55] Ivan Damgård and Jesper Buus Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In *CRYPTO*, 2007.

[56] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*, 2012.

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

[58] Damien Desfontaines and Balázs Pejó. SoK: Differential Privacies. *PETS*, 2020.

[59] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the Communication Barrier in Secure Computation using Lookup Tables. In *NDSS*, 2017.

[60] Yuanchao Ding, Hua Guo, Yewei Guan, Weixin Liu, Jiarong Huo, Zhenyu Guan, and Xiyong Zhang. East: Efficient and Accurate Secure Transformer Framework for Inference. *CoRR*, abs/2308.09923, 2023.

[61] Caiqin Dong, Jian Weng, Jia-Nan Liu, Yue Zhang, Yao Tong, Anjia Yang, Yudan Cheng, and Shun Hu. Fusion: Efficient and Secure Inference Resilient to Malicious Servers. In *NDSS*, 2023.

[62] Ye Dong, Xiaojun Chen, Weizhan Jing, Kaiyun Li, and Weiping Wang. Meteor: Improved Secure 3-Party Neural Network Inference with Reducing Online Communication Costs. In *WWW 2023*, 2023.

[63] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. PUMA: Secure Inference of LLaMA-7B in Five Minutes. *CoRR*, abs/2307.12533, 2023.

[64] Wenliang Du and Mikhail J. Atallah. Protocols for Secure Remote Database Access with Approximate Matching. In *E-Commerce Security and Privacy*. 2001.

[65] Hendrik Eerikson, Marcel Keller, Claudio Orlandi, Pille Pullonen, Joonas Puura, and Mark Simkin. Use Your Brain! Arithmetic 3PC for Any Modulus with Active Security. In *ITC*, 2020.

[66] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In *CRYPTO*, 2020.

[67] European Commission, AI Office. The General-Purpose AI Code of Practice. European Commission / Digital Strategy, 2025.

[68] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.*, 144, 2012.

[69] Jun Feng, Yefan Wu, Hong Sun, Shunli Zhang, and Debin Liu. Panther: Practical Secure Two-Party Neural Network Inference. *IEEE TIFS*, 20, 2025.

[70] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority. In *EUROCRYPT*, 2017.

[71] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO*, 1984.

[72] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and Secure Solutions for Integer Comparison. In *PKC*, 2007.

[73] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. Secure Computation with Low Communication from Cross-Checking. In *ASIACRYPT*, 2018.

[74] Hao Guo, Liqiang Peng, Haiyang Xue, Li Peng, Weiran Liu, Zhe Liu, and Lei Hu. Improved Secure Two-party Computation from a Geometric Perspective. In *USENIX Security*, 2025.

[75] Kanav Gupta, Nishanth Chandran, Divya Gupta, Jonathan Katz, and Rahul Sharma. SHARK: Actively Secure Inference Using Function Secret Sharing. In *S&P*, 2025.

[76] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT Inference with Function Secret Sharing. *PETS*, 2024.

[77] Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. LLAMA: A Low Latency Math Library for Secure Inference. *PETS*, 2022.

[78] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private Inference on Transformers. In *NeurIPS*, 2022.

[79] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, and Tianwei Zhang. FastSecNet: An Efficient Cryptographic Framework for Private Neural Network Inference. *IEEE TIFS*, 18, 2023.

[80] Christopher Harth-Kitzerow, Ajith Suresh, and Georg Carle. SoK: Truncation Untangled: Scaling Fixed-Point Arithmetic for Privacy-Preserving Machine Learning to Large Models and Datasets. *PETS*, 2025.

[81] Christopher Harth-Kitzerow, Ajith Suresh, Yongqin Wang, Hossein Yalame, Georg Carle, and Murali Annavaram. High-Throughput Secure Multiparty Computation with an Honest Majority in Various Network Settings. *PETS*, 2025.

[82] Christopher Harth-Kitzerow, Yongqin Wang, Rachit Rajat, Georg Carle, and Murali Annavaram. PIGEON: A Framework for Private Inference of Neural Networks. In *PETS*, 2025.

[83] Jiaxing He, Kang Yang, Guofeng Tang, Zhangjie Huang, Li Lin, Changzheng Wei, Ying Yan, and Wei Wang. Rhombus: Fast Homomorphic Matrix-Vector Multiplication for Secure Two-Party Inference. In *CCS*, 2024.

[84] Dan Hendrycks and Kevin Gimpel. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR*, abs/1606.08415, 2016.

[85] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GeLUs). *arXiv preprint arXiv:1606.08415*, 2016.

[86] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen-jie Lu, Cheng Hong, and Kui Ren. CipherGPT: Secure Two-Party GPT Inference. *IACR Cryptol. ePrint Arch.*, 1147, 2023.

[87] Xiaoyang Hou, Jian Liu, Jingyu Li, Jiawen Zhang, and Kui Ren. Faster Lookup Table Evaluation with Application to Secure LLM Inference. *IACR Cryptol. ePrint Arch.*, 2024.

[88] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *USENIX Security*, 2022.

[89] Siam Umar Hussain, Mojan Javaheripi, Mohammad Samragh, and Farinaz Koushanfar. COINN: Crypto/ML Codesign for Oblivious Inference via Neural Networks. In *CCS*, 2021.

[90] Neha Jawalkar, Nishanth Chandran, Divya Gupta, Rahul Sharma, and Arkaprava Basu. Matchmaker: Fast Secure Inference across Deployment Scenarios. Cryptology ePrint Archive, Paper 2025/424, 2025.

[91] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: FSS-based Secure Training and Inference with GPUs. In *S&P*, 2024.

[92] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security*, 2018.

[93] Andes Y. L. Kei and Sherman S. M. Chow. SHAFT: Secure, Handy, Accurate and Fast Transformer Inference. In *NDSS*, 2025.

[94] Marcel Keller. Rep3 Reloaded: On the Cost of Function-Dependent Preprocessing in Semi-Honest 3PC with Honest Majority. *IACR Cryptol. ePrint Arch.*, 919, 2025.

[95] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ Great Again. In *EUROCRYPT*, 2018.

[96] Tanveer Khan, Mindaugas Budzys, Khoa Nguyen, and Antonis Michalas. SoK: Wildest Dreams: Reproducible Research in Privacy-preserving Neural Network Training. *PETS*, 2024.

[97] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *NeurIPS*, 2021.

[98] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In *USENIX Security*, 2021.

[99] Nishat Koti, Shravani Mahesh Patil, Arpita Patra, and Ajith Suresh. MPClan: Protocol Suite for Privacy-Conscious Computations. *J. Cryptol.*, 36(3), 2023.

[100] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively Secure 4PC for Secure Training and Inference. In *NDSS*, 2022.

[101] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow Inference. In *S&P*, 2020.

[102] Konrad Kuźniewski, Krystian Matusiewicz, and Piotr Sapiecha. The High-Level Practical Overview of Open-Source Privacy-Preserving Machine Learning Solutions. *International Journal of Electronics and Telecommunications*, 2022.

[103] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure Inference Resilient to Malicious Clients. In *USENIX Security*, 2021.

[104] Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric P. Xing, and Hao Zhang. MPCFORMER: Fast, Performant and Provate Transformer Inference with MPC. In *ICLR*, 2023.

[105] Yi Li and Wei Xu. Privpy: General and scalable privacy-preserving data mining. In *KDD*, 2019.

[106] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3PC for Binary Circuits with Application to Maliciously-Secure DNN Inference. In *USENIX Security*, 2023.

[107] Zhengyi Li, Kang Yang, Jin Tan, Wen-jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, and Jingwen Leng. Nimbus: Secure and Efficient Two-Party Inference for Transformers. In *NeurIPS*, 2024.

[108] Yehuda Lindell. Secure Multiparty Computation. *Communications of the ACM*, 2021.

[109] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. When Machine Learning Meets Privacy: A Survey and Outlook. *ACM Comput. Surv.*, 54(2), 2022.

[110] Fengrun Liu, Xiang Xie, and Yu Yu. Scalable Multi-Party Computation Protocols for Machine Learning in the Honest-Majority Setting. In *USENIX Security*, 2024.

[111] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *CCS*, 2017.

[112] Siqi Liu, Zhusen Liu, Donglong Chen, Wangchen Dai, Lu Zhou, Zhe Liu, Ray C. C. Cheung, and Çetin Kaya Koç. MLFormer: a high performance MPC linear inference framework for transformers. *J. Cryptogr. Eng.*, 15(1), 2025.

[113] Xiaoning Liu, Yifeng Zheng, Xingliang Yuan, and Xun Yi. Towards Secure and Lightweight Deep Learning as a Medical Diagnostic Service. In *ESORICS*, 2021.

[114] Xiaoning Liu, Yifeng Zheng, Xingliang Yuan, and Xun Yi. Securely Outsourcing Neural Network Inference to the Cloud With Lightweight Techniques. *IEEE Trans. Dependable Secur. Comput.*, 20(1), 2023.

[115] Ximeng Liu, Lehui Xie, Yaopeng Wang, Jian Zou, Jinbo Xiong, Zuobin Ying, and Athanasios V. Vasilakos. Privacy and Security Issues in Deep Learning: A Survey. *IEEE Access*, 9, 2020.

[116] Tianpei Lu, Bingsheng Zhang, Lichun Li, Yuzhou Zhao, and Kui Ren. Lightning Fast Secure Comparison for 3PC PPML. Cryptology ePrint Archive, Paper 2023/1890, 2023.

[117] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. BumbleBee: Secure Two-party Inference Framework for Large Transformers. In *NDSS*, 2025.

[118] Wen-jie Lu, Zhicong Huang, Qizhi Zhang, Yuchen Wang, and Cheng Hong. Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree. In *USENIX Security*, 2023.

[119] Jinglong Luo, Yehong Zhang, Zhuo Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. SecFormer: Fast and Accurate Privacy-Preserving Inference for Transformer Models via SMPC. In *ACL*, 2024.

[120] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient Comparison for Secure Multi-Party Computation. In *FC*, 2021.

[121] Zoltán Ádám Mann, Christian Weinert, Daphnee Chabal, and Joppe W. Bos. Towards Practical Secure Neural Network Inference: The Journey So Far and the Road Ahead. *IACR Cryptol. ePrint Arch.*, 1483, 2022.

[122] Felix Marx, Thomas Schneider, Ajith Suresh, Tobias Wehrle, Christian Weinert, and Hossein Yalame. WW-FL: Secure and Private Large-Scale Federated Learning, 2024.

[123] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A Cryptographic Inference Service for Neural Networks. In *USENIX Security*, 2020.

[124] Payman Mohassel and Peter Rindal. ABY^3: A Mixed Protocol Framework for Machine Learning. In *CCS*, 2018.

[125] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *S&P*, 2017.

[126] Lucien K. L. Ng and Sherman S. M. Chow. SoK: Cryptographic Neural-Network Computation. In *S&P*, 2023.

[127] Nges Brian Njungle, Eric Jahns, Zhenqi Wu, Luigi Mastromauro, Milan Stojkov, and Michel Kinsy. GuardianML: Anatomy of Privacy-Preserving Machine Learning Techniques and Frameworks. *IEEE Access*, 2025.

[128] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, 1999.

[129] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. In *S&P*, 2024.

[130] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *USENIX Security*, 2021.

[131] Arpita Patra and Ajith Suresh. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS*, 2020.

[132] Robert Podschwadt, Daniel Takabi, Peizhao Hu, Mohammad Hossein Rafiei, and Zhipeng Cai. A Survey of Deep Learning Architectures for Privacy-Preserving Machine Learning With Fully Homomorphic Encryption. *IEEE Access*, 10, 2022.

[133] Luis Bernardo Pulido-Gaytan, Andrei Tchernykh, Jorge M. Cortés-Mendoza, Mikhail G. Babenko, and Gleb I. Radchenko. A Survey on Privacy-Preserving Machine Learning with Fully Homomorphic Encryption. In *CARLA*, 2020.

[134] Luis Bernardo Pulido-Gaytan, Andrei Tchernykh, Jorge M. Cortés-Mendoza, Mikhail G. Babenko, Gleb I. Radchenko, Arutyun Avetisyan, and Alexander Yu. Drozdov. Privacy-preserving neural networks with Homomorphic encryption: Challenges and opportunities. *Peer-to-Peer Netw. Appl.*, 14(3), 2021.

[135] Pille Pullonen and Sander Siim. Combining Secret Sharing and Garbled Circuits for Efficient Private IEEE 754 Floating-Point Computations. In *FC*, 2015.

[136] Hong Qin, Debiao He, Qi Feng, Muhammad Khurram Khan, Min Luo, and Kim-Kwang Raymond Choo. Cryptographic Primitives in Privacy-Preserving Machine Learning: A Survey. *IEEE Trans. Knowl. Data Eng.*, 36(5), 2024.

[137] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[138] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[139] Deevashwer Rathee, Anwesh Bhattacharya, Divya Gupta, Rahul Sharma, and Dawn Song. Secure Floating-Point Training. In *USENIX Security*, 2023.

[140] Deevashwer Rathee, Anwesh Bhattacharya, Rahul Sharma, Divya Gupta, Nishanth Chandran, and Aseem Rastogi. SecFloat: Accurate Floating-Point meets Secure 2-Party Computation. In *S&P*, 2022.

[141] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. SiRnn: A Math Library for Secure RNN Inference. In *S&P*, 2021.

[142] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS*, 2020.

[143] Christian Rechberger and Roman Walch. Privacy-Preserving Machine Learning Using Cryptography. In *Security and Artificial Intelligence: A Crossdisciplinary Approach*. 2022.

[144] M. Sadegh Riazi, Bita Darvish Rouhani, and Farinaz Koushanfar. Deep Learning on Private Data. *IEEE Secur. Priv.*, 17(6), 2019.

[145] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: XNOR-based Oblivious Deep Neural Network Inference. In *USENIX Security*, 2019.

[146] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *AsiaCCS*, 2018.

[147] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 1978.

[148] Marc Rivinius, Pascal Reisert, Sebastian Hasler, and Ralf Küsters. Convolutions in overdrive: maliciously secure convolutions for MPC. *PETS*, 2023.

[149] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: scalable provably-secure deep learning. In *DAC*, 2018.

[150] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis R. Bach. AriaNN: Low-Interaction Privacy-Preserving Deep Learning via Function Secret Sharing. *PETS*, 2022.

[151] Manuel B. Santos, Dimitris Mouris, Mehmet Ugurbil, Stanislaw Jarecki, José Reis, Shubho Sengupta, and Miguel de Vega. Curl: Private LLMs through Wavelet-Encoded Look-Up Tables. *IACR Cryptol. ePrint Arch.*, 1127, 2024.

[152] Adi Shamir. How to share a secret. *Communications of the ACM*, 1979.

[153] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[154] Liyan Shen, Xiaojun Chen, Jinqiao Shi, Ye Dong, and Binxing Fang. An Efficient 3-Party Framework for Privacy-Preserving Neural Network Inference. In *ESORICS*, 2020.

[155] Nigel P. Smart and Titouan Tanguy. TaaS: Commodity MPC via Triples-as-a-Service. In *ACM CCSW@CCS*, 2019.

[156] Lushan Song, Jiaxuan Wang, Zhexuan Wang, Xinyu Tu, Guopeng Lin, Wenqiang Ruan, Haoqi Wu, and Weili Han. pMPL: A Robust Multi-Party Learning Framework with a Privileged Party. In *CCS*, 2022.

[157] Stanford. CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/convolutional-networks/.

[158] Ajith Suresh. MPCLeague: Robust MPC Platform for Privacy-Preserving Machine Learning. *CoRR*, abs/2112.13338, 2021.

[159] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU. In *S&P*, 2021.

[160] Harry Chandra Tanuwidjaja, Rakyong Choi, Seunggeun Baek, and Kwangjo Kim. Privacy-Preserving Deep Learning on Machine Learning as a Service - a Comprehensive Survey. *IEEE Access*, 8, 2020.

[161] Harry Chandra Tanuwidjaja, Rakyong Choi, and Kwangjo Kim. A Survey on Deep Learning Techniques for Privacy-Preserving. In *ML4CS*, 2019.

[162] Han Tian, Chaoliang Zeng, Zhenghang Ren, Di Chai, Junxue Zhang, Kai Chen, and Qiang Yang. Sphinx: Enabling Privacy-Preserving Online Learning over the Cloud. In *S&P*, 2022.

[163] Kapil Tiwari, Samiksha Shukla, and Jossy P. George. A Systematic Review of Challenges and Techniques of Privacy-Preserving Machine Learning. In *Data Science and Security*, 2021.

[164] Yiwen Tong, Qi Feng, Min Luo, and Debiao He. Multi-party privacy-preserving decision tree training with a privileged party. *Sci. China Inf. Sci.*, 2024.

[165] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[166] Mehmet Ugurbil, Dimitris Mouris, Manuel B. Santos, José Cabrero-Holgueras, Miguel de Vega, and Shubho Sengupta. Fission: Distributed Privacy-Preserving Large Language Model Inference. *IACR Cryptol. ePrint Arch.*, 653, 2025.

[167] Ali Burak Ünal, Mete Akgün, and Nico Pfeifer. CECILIA: Comprehensive Secure Machine Learning Framework. *CoRR*, abs/2202.03023, 2022.

[168] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[169] Sameer Wagh. Pika: Secure Computation using Function Secret Sharing over Rings. *PETS*, 2022.

[170] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. *PETS*, 2019.

[171] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *PETS*, 2021.

[172] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU Platform for Secure Computation. In *USENIX Security*, 2022.

[173] Pengzhi Xing, Hongwei Li, Meng Hao, Hanxiao Chen, Jia Hu, and Dongxiao Liu. Distributed function secret sharing and applications. In *NDSS 2025*, 2025.

[174] Guowen Xu, Xingshuo Han, Tianwei Zhang, Shengmin Xu, Jianting Ning, Xinyi Huang, Hongwei Li, and Robert H. Deng. SIMC 2.0: Improved Secure ML Inference Against Malicious Clients. *IEEE Trans. Dependable Secur. Comput.*, 21(4), 2024.

[175] Tianshi Xu, Wen-jie Lu, Jiangrui Yu, Yi Chen, Chenqi Lin, Runsheng Wang, and Meng Li. Breaking the Layer Barrier: Remodeling Private Transformer Inference with Hybrid CKKS and MPC. *USENIX Security*, 2025.

[176] Xiangrui Xu, Qiao Zhang, Rui Ning, Chunsheng Xin, and Hongyi Wu. Comet: A Communication-efficient and Performant Approximation for Private Transformer Inference. *CoRR*, abs/2405.17485, 2024.

[177] Peng Yang, Zoe Lin Jiang, Shiqi Gao, Hongxiao Wang, Jun Zhou, Yangyiye Jin, Siu-Ming Yiu, and Junbin Fang. Communication-Efficient Secure Neural Network via Key-Reduced Distributed Comparison Function. In *ProvSec*, 2025.

[178] Ruidi Yang. Survey on Privacy-Preserving Machine Learning Protocols. In *ML4CS*, 2020.

[179] Xuanang Yang, Jing Chen, Yuqing Li, Kun He, Xiaojie Huang, Zikuan Jiang, Hao Bai, and Ruiying Du. Fregata: Fast Private Inference With Unified Secure Two-Party Protocols. *IEEE TIFS*, 19, 2024.

[180] Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *Foundations of Computer Science*, 1982.

[181] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets. In *FOCS 1986*, 1986.

[182] Zhichao You, Xuewen Dong, Ke Cheng, Xutong Mu, Jiaxuan Fu, Shiyang Ma, Qiang Qu, and Yulong Shen. PriFFT: Privacy-preserving Federated Fine-tuning of Large Language Models via Function Secret Sharing. *CoRR*, abs/2503.03146, 2025.

[183] Boshi Yuan, Shixuan Yang, Yongxiang Zhang, Ning Ding, Dawu Gu, and Shi-Feng Sun. MD-ML: Super Fast Privacy-Preserving Machine Learning for Malicious Security with a Dishonest Majority. In *USENIX Security*, 2024.

[184] Martin Zbudila, Aysajan Abidin, and Bart Preneel. Exploring Adversarial Attacks on the MaSTer Truncation Protocol. In *IH&MMSEC*, 2025.

[185] Martin Zbudila, Erik Pohle, Aysajan Abidin, and Bart Preneel. MaSTer: Maliciously Secure Truncation for Replicated Secret Sharing Without Pre-processing. In *CANS*, 2024.

[186] Wenxuan Zeng, Meng Li, Wenjie Xiong, Tong Tong, Wen-jie Lu, Jin Tan, Runsheng Wang, and Ru Huang. Mpcvit: Searching for accurate and efficient mpc-friendly vision transformer with heterogeneous attention. In *CVF*, 2023.

[187] Wenxuan Zeng, Tianshi Xu, Yi Chen, Yifan Zhou, Mingzhe Zhang, Jin Tan, Cheng Hong, and Meng Li. Towards Efficient Privacy-Preserving Machine Learning: A Systematic Review from Protocol, Model, and System Perspectives. *arXiv preprint arXiv:2507.14519*, 2025.

[188] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. GALA: Greedy ComputAtion for Linear Algebra in Privacy-Preserved Neural Networks. In *NDSS*, 2021.

[189] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. Privacy-Preserving Deep Learning Based on Multiparty Secure Computation: A Survey. *IEEE IoT*, 8(13), 2021.

[190] Yansong Zhang, Xiaojun Chen, Ye Dong, Qinghui Zhang, Rui Hou, Qiang Liu, and Xudong Chen. MD-SONIC: Maliciously-Secure Outsourcing Neural Network Inference With Reduced Online Communication. *IEEE TIFS*, 20, 2025.

[191] Yansong Zhang, Xiaojun Chen, Qinghui Zhang, Ye Dong, and Xudong Chen. Helix: Scalable Multi-Party Machine Learning Inference against Malicious Adversaries. *IACR Cryptol. ePrint Arch.*, 347, 2025.

[192] Fei Zheng, Chaochao Chen, Zhongxuan Han, and Xiaolin Zheng. PermLLM: Private Inference of Large Language Models within 3 Seconds under WAN. *CoRR*, abs/2405.18744, 2024.

[193] Mengxin Zheng, Qian Lou, and Lei Jiang. Primer: Fast Private Transformer Inference on Encrypted Data. In *DAC*, 2023.

[194] Yu Zheng, Qizhi Zhang, Sherman S. M. Chow, Yuxiang Peng, Sijun Tan, Lichun Li, and Shan Yin. Secure Softmax/Sigmoid for Machine-learning Computation. In *ACSAC*, 2023.

[195] Ian Zhou, Farzad Tofigh, Massimo Piccardi, Mehran Abolhasan, Daniel Robert Franklin, and Justin Lipman. Secure Multi-Party Computation for Machine Learning: A Survey. *IEEE Access*, 12, 2024.

[196] Lijing Zhou, Qingrui Song, Su Zhang, Ziyu Wang, Xianggui Wang, and Yong Li. Bicoptor 2.0: Addressing Challenges in Probabilistic Truncation for Enhanced Privacy-Preserving Machine Learning. *CoRR*, abs/2309.04909, 2023.

[197] Lijing Zhou, Ziyu Wang, Hongrui Cui, Qingrui Song, and Yu Yu. Bicoptor: Two-round Secure Three-party Non-linear Computation without Preprocessing for Privacy-preserving Machine Learning. In *S&P*, 2023.

[198] Lijing Zhou, Bingsheng Zhang, Ziyu Wang, Tianpei Lu, Qingrui Song, Su Zhang, Hongrui Cui, and Yu Yu. On Probabilistic Truncation in Privacy-preserving Machine Learning. In *AAAI*, 2025.

[199] Huan Zou, Yuting Xiao, and Rui Zhang. Semi-Honest 2-Party Faithful Truncation from Two-Bit Extraction. *IACR Cryptol. ePrint Arch.*, 1159, 2023.

Appendix A. Related Work (Additional)

In this section, we expand on the analysis provided in Section 1.1 by diving deeper into the surveys on privacy-preserving machine learning (PPML). While the main body focused on comparing SoKs, here we provide a structured overview of surveys published between 2019 and 2025. These works vary in scope and emphasis: some offer broad introductory overviews, others analyze specific cryptographic techniques such as MPC or HE, while more recent surveys consolidate the field by integrating threat models, deployment scenarios, and practical challenges.

The earliest surveys—such as those by Tanuwidjaja et al. [161], Boulemtafes et al. [24], and Yang [178]—provided high-level introductions to PPML, categorizing cryptographic methods like MPC and HE and mapping them to ML stages such as training and inference. Their main value was in establishing a taxonomy of methods and highlighting privacy as a central concern in ML, rather than offering in-depth performance insights.

Later surveys placed stronger emphasis on adversarial models and security risks. Liu et al. [115] and Tiwari et al. [163] systematically reviewed attacks such as membership inference, inversion, and poisoning, linking these threats with privacy-preserving defenses. Their added value lies in mapping specific attack vectors to countermeasures, making them security-centric references for PPML.

Several surveys concentrated on single cryptographic tools. Pulido-Gaytan et al. [133], [134] and Podschwadt et al. [132] examined homomorphic encryption for ML, discussing challenges such as handling non-linear operations and quantization. Zhang et al. [189] reviewed MPC-based frameworks, including performance evaluations. These works offered deeper insights into individual techniques, clarifying their trade-offs and applicability in PPML.

Other contributions aimed at broader integration. Tanuwidjaja et al. [160], Rechberger et al. [143], and Mann et al. [121] surveyed PPML across the ML lifecycle, combining system stages with privacy mechanisms. Kuzniewski et al. [102] went further by benchmarking open-source frameworks, highlighting reproducibility issues and gaps between theory and practice. These works emphasized system-level perspectives and practical usability challenges.

More recent surveys reflect the maturity of the field. Qin et al. [136] mapped cryptographic primitives directly to PPML workflows, while Zhou et al. [195] reframed HE within an MPC-centric taxonomy. Allaart et al. [9] analyzed privacy-preserving distributed deep learning, identifying open challenges such as secure aggregation and scalability. Njungle et al. [127] introduced GuardianML, a recommendation system to match frameworks to use cases. Finally, Chen et al. [41] provided a holistic synthesis, covering deployment scenarios, threat models, optimization techniques, and future research directions. These surveys represent the latest efforts to consolidate and operationalize the PPML landscape.

We present a detailed comparison against relevant existing surveys and SoK works in the domain of PPML in Table 8.

Appendix B. Preliminaries - MPC Primitives

Oblivious Transfer. Oblivious transfer (OT), first introduced by Rabin [137], is a fundamental two-party protocol in which a sender transmits information to a receiver in a manner that preserves mutual privacy. Specifically, the receiver learns exactly one of the sender's messages without gaining any information about the others, while the sender remains oblivious to the receiver's choice. This primitive was later generalized to the *1-out-of-N OT*, denoted by OT_1^N. In this setting, the sender's input is a tuple $(x_0, \dots, x_{N-1})$, and the receiver's input is an index $i \in [0, N-1]$. At the end of the protocol, the receiver obtains x_i but learns nothing about $\{x_j\}_{j \neq i}$, while the sender learns no information about the index i.

Garbled Circuits. In his seminal work on MPC [181], Yao introduced a two-party protocol that enables secure computation of any function expressed as a Boolean circuit under the semi-honest security model. Central to this construction is the notion of *garbled circuits* (GC), which, at a high level, conceals the actual values carried on the wires of the Boolean circuit during its evaluation.

A GC is an encrypted representation of a Boolean circuit C_f implementing a public function f. Consider a Boolean circuit where every gate has two input wires and one output wire. For each wire in the circuit, the *garbler* assigns a pair of cryptographic keys (or labels), one corresponding to each semantic value, 0 and 1. The truth table of every gate is then *garbled* by encrypting the label of the output wire under the labels of the input wires. After garbling all gates, the garbler sends the resulting garbled circuit GC_{C_f} to the other party, called the *evaluator*, along with the labels corresponding to the garbler's input. The evaluator acquires the labels for its own input by engaging in a OT_1^2 protocol with the garbler acting as the sender. Finally, the evaluator computes the function by iteratively decrypting the garbled truth tables using the obtained wire labels in topological order.

Secret Sharing. Secret sharing enables the distribution of a secret value among multiple parties such that only a qualified subset of parties can reconstruct the secret, while any subset of parties outside the qualified set learns nothing about it. For instance, in additive secret sharing among n parties, a secret x is divided into n shares x_i, $i = 1, \dots, n$, with party i holding x_i and satisfying $x = \sum_{i=1}^{n} x_i$.

Different variants of secret sharing exist. *Threshold secret sharing*, such as Shamir's scheme [152], or Replicated Secret Sharing (RSS), allows reconstruction only if at least t parties participate. In contrast, *non-threshold schemes*, such as additive secret sharing, do not impose such a threshold requirement. For stronger security guarantees in adversarial models, *Verifiable Secret Sharing* (VSS) and its variants have been proposed.

TABLE 8: Comparison of our SoK to related surveys and SoKs in the domain of PPML.

Reference	Year	SoK/ Survey	Techniques				Setting			ML			Added value
			MPC	FSS	FHE	TEE	Hybrid	C/S	TTP	Outsourcing	Inference	Training	
Riazi et al. [144]	2019	SoK	●	○	○	○	●	●	●	●	●	●	Overview of attacks on ML
Tanuwidjaja et al. [161]	2019	Survey	●	○	●	○	●	●	○	●	●	●	Intro to PPML methods
Azraoui et al. [13]	2019	SoK	●	○	●	○	●	●	○	●	●	●	Limited evaluation
Boulemtafes et al. [24]	2020	Survey	●	○	●	○	○	●	●	●	●	●	Broad taxonomy overview
Liu et al. [115]	2020	Survey	●	○	●	●	●	●	●	●	●	●	Attack and defense analysis
Tanuwidjaja et al. [160]	2020	Survey	●	○	●	●	●	●	●	●	●	●	Review for MLaaS settings
Yang [178]	2020	Survey	●	○	●	○	●	●	○	●	●	●	PPML architecture overview
Pulido-Gaytan et al. [133]	2020	Survey	○	○	●	○	○	○	○	●	●	●	Challenges in HE-based PPML
Tiwari et al. [163]	2021	Survey	●	○	●	○	●	●	○	●	●	●	Threat–defense mapping
Zhang et al. [189]	2021	Survey	●	○	●	●	●	●	●	●	●	●	MPC benchmarks
Pulido-Gaytan et al. [134]	2021	Survey	○	○	●	○	○	○	○	●	●	●	Extended HE survey
Cabrero-Holgueras et al. [35]	2021	SoK	●	○	●	○	●	●	○	●	●	●	Bridging research and industry
Podschwadt et al. [132]	2022	Survey	○	○	●	○	○	○	○	●	●	●	HE taxonomy
Rechberger et al. [143]	2022	Survey	●	○	●	○	●	●	○	●	●	●	End-to-end ML view
Kuzniewski et al. [102]	2022	Survey	●	○	●	●	○	○	●	●	●	○	Limited benchmarks
Mann et al. [121]	2022	Survey	●	○	●	○	●	●	○	●	●	●	Integrated view
Ng et al. [126]	2023	SoK	●	○	●	○	●	●	○	●	●	●	Genealogy
Khan et al. [96]	2024	SoK	●	○	●	○	○	○	○	●	○	●	Experimental evaluation
Qin et al. [136]	2024	Survey	●	○	●	●	●	●	●	●	●	●	Crypto primitives mapping
Zhou et al. [195]	2024	Survey	●	○	○	○	○	●	○	●	●	●	MPC-centric taxonomy
Allaart et al. [9]	2025	Survey	●	○	●	○	●	●	●	●	●	●	Distributed DL challenges
Njungle et al. [127]	2025	Survey	●	○	●	●	●	●	●	●	●	●	Recommendation system
Chen et al. [41]	2025	Survey	●	●	○	○	●	●	●	●	●	●	Holistic synthesis
This Work	**2025**	**SoK**	●	●	○	○	●	●	●	●	●	●	**Low-level protocol analysis + conversion protocols**

In secret-sharing-based protocols, the general paradigm is to begin by secret sharing the inputs. The parties then interactively compute the function in such a way that after every operation, the result remains secret-shared under the same semantics. This process continues until the complete function evaluation is obtained in secret-shared form. Finally, the secret-shared output is reconstructed and revealed to a designated party.

Function Secret Sharing. Function Secret Sharing (FSS) [26] extends the notion of secret sharing from values to functions, by distributing a function among multiple parties such that the function itself remains hidden. This contrasts with traditional secret-sharing schemes, where the focus is on keeping the input values secret. While FSS introduces additional computational overhead, it significantly reduces communication and round complexity: each party transmits only a single group element corresponding to its input in one round of interaction. The primary motivation for using FSS lies in its low communication overhead, thereby reducing reliance on network performance. The computational overhead, in turn, can be alleviated through appropriate hardware acceleration.

FSS has attracted considerable interest in the domain of privacy-preserving machine learning (PPML), particularly in the two-party computation (2PC) setting [76], [77], [91], [150]. Owing to the computational overhead in preprocessing, most of these schemes rely on a helper or dealer to assist the computation. For a comprehensive treatment of constructing secure computation from FSS, we refer the reader to Boyle et al. [28].

Homomorphic Encryption. Homomorphic Encryption (HE) allows computations to be performed directly on encrypted data such that, upon decryption, the result corresponds to the outcome of the same computation on the original plaintext. Formally, a cryptosystem E is said to be $\boxplus$-homomorphic if there exists a ciphertext operation $\boxtimes$ satisfying

$$E(m_1) \boxtimes E(m_2) = E(m_1 \boxplus m_2).$$

Classical examples include the RSA [147] and ElGamal [71] cryptosystems, which are *multiplicatively homomorphic*, and the Paillier cryptosystem [128], which is *additively homomorphic*. A Fully Homomorphic Encryption (FHE) scheme supports both additive and multiplicative homomorphisms, thereby enabling arbitrary computation over encrypted data.

In the context of MPC-based PPML frameworks, HE is often used in the preprocessing phase to generate correlated randomness such as Beaver triples and matrix triples, which are essential for secure multiplication of values and matrices. More recent MPC–HE hybrid approaches leverage HE to evaluate linear layers (e.g., convolutions), while relying on MPC protocols for the computation of non-linear layers.

While HE schemes are attractive due to their low communication overhead, they often incur substantial computational costs. Consequently, a significant body of ongoing research focuses on improving the efficiency of HE, with particular emphasis on accelerating homomorphic encryption through optimized algorithms, system-level improvements, and specialized hardware support.

Appendix C. MPC Design-based Systematization

In this section, we review PPML frameworks from two complementary perspectives. First, we present a systematic classification that compares frameworks in terms of the number of parties, underlying cryptographic techniques, secret-sharing semantics, and security models (Appendix C.1). Second, we provide a genealogy of these frameworks in Figure 1, illustrating how protocol designs have evolved from generic secret-sharing approaches to function-dependent schemes, hybrid MPC–HE constructions, and transformer-specific optimizations (Appendix C.2). Taken together, these perspectives highlight both the current capabilities of PPML systems and the key design trends shaping their development.

C.1. Systematic Overview and Classification

Table 9, Table 10, and Table 11 present a unified classification of secure computation frameworks for PPML in two-party, three- and four-party, and general N-party settings, respectively. Table 9 lists 2PC frameworks chronologically, detailing the number of parties (including dealer-assisted settings), cryptographic techniques—Secret Sharing (SS), Homomorphic Encryption (HE), Oblivious Transfer (OT), Garbled Circuits (GC), and Zero-Knowledge Proofs (ZK)—deployment setting, and security model, ranging from semi-honest to malicious, including dishonest-majority and asymmetric corruption. It further indicates whether the framework supports training, inference, or both, and distinguishes between theoretical and experimental evaluations, noting when implementations are unavailable.

Table 10 extends this classification to three- and four-party frameworks, while Table 11 covers general multi-party settings, contrasting dishonest-majority approaches with honest-majority designs. Finally, Table 12 summarizes neural network layer support in 2PC frameworks—including linear and convolutional layers, fixed-point truncation (probabilistic, stochastic, faithful), activations (ReLU, SoftMax, Sigmoid, GELU, SiLU), and normalization—whereas Table 13 provides the corresponding details for three-, four-, and general N-party settings.

C.2. Genealogy

Figure 1 illustrates the genealogy of MPC protocols and frameworks for PPML, organized from top to bottom into four main categories.

At the top, we present protocols based on additive, function-independent secret sharing. These approaches employ generic preprocessed shares that are independent of the target function, offering broad applicability but often incurring higher online communication costs. The second category comprises masking-based, function-dependent secret sharing protocols, where preprocessing is tailored to the specific target function. This specialization enables the generation of correlated randomness—such as for dot products—in the offline phase, significantly reducing the cost of expensive operations and improving online efficiency. The third tier includes hybrid MPC-HE frameworks, which integrate MPC with HE to optimize different computational phases—typically employing HE (with packing techniques) for arithmetic/linear layers, and MPC for non-linear operations. These hybrids aim to leverage the complementary strengths of both primitives to improve overall performance. Finally, at the bottom, we list transformer-specific frameworks that adapt MPC—often incorporating FSS—to the architectural characteristics and computational patterns of transformer models. These systems frequently introduce specialized optimizations for attention mechanisms and large-scale model processing.

In Figure 1, arrows depict the evolution of protocols, showing which works influenced or directly extended others. Numbers inside circles indicate the number of computing parties, while shield icons mark protocols designed for a malicious security model.

TABLE 9: Systematic overview of 2PC frameworks. (●) denotes availability, (◐) partial availability and (○) unavailability. Abbreviations: O: Outsourcing, C-S: client-server. Additional abbreviations are defined in Table 2.

Framework		MPC										ML		Evaluation	
		N of parties			Techniques					Setting	Sec.				
Scheme	Year	2	3	4	HE	OT	GC	ZK	SS			Training	Inference	Theoretical	Experimental
SecureML [125]	2017	●	◐[2]	○	●	●	○	○	●	O	SH	●	●	○	◐[4]
DeepSecure [149]	2017	●	○	○	○	●	●	○	○	O	SH	●	●	○	◐[4]
MiniONN [111]	2017	●	○	○	●	●	●	○	●	C-S	SH	○	●	○	◐[4]
Chameleon [146]	2017	●	◐[2]	○	○	●	●	○	●	O	SH	○	●	○	◐[4]
Gazelle [92]	2018	●	○	○	●	●	●	○	●	C-S	SH	○	●	○	◐[4]
Delphi [123]	2020	●	○	○	●	●	●	○	●	C-S	SH	○	●	○	●
Cryptflow2 [142]	2020	●	○	○	●	●	○	○	●	C-S	SH	○	●	●	●
MediSC [113]	2021	●	◐[2]	○	○	○	○	○	●	C-S	SH	○	●	○	◐[4]
COINN [89]	2021	●	○	○	○	●	●	○	●	C-S	SH	○	●	●	●
Muse [103]	2021	●	○	○	●	●	●	●	●	C-S	A^C	○	●	○	◐[4]
AriaNN [150]	2021	●	◐[2]	○	○	○	○	○	FSS	O	SH	●	●	◐[7]	●
SiRnn [141]	2021	●	○	○	○	●	○	○	●	O	SH	○	●	◐[6]	●
ABY2.0 [130]	2021	●	○	○	●	●	●	○	●	O	SH	●	●	◐[6]	◐[4]
SONIC [114]	2022	●	◐[2]	○	○	○	○	○	●	O	SH	○	●	●	◐[4]
Sphinx [162]	2022	●	○	○	●	○	○	○	●	C-S	SH	●	●	○	◐[4]
SIMC [38]	2022	●	○	○	●	●	●	●	●	C-S	A^C	○	●	○	●
Cheetah [88]	2022	●	○	○	●	●	○	○	●	C-S	SH	○	●	◐[6]	●
Iron [78]	2022	●	○	○	●	●	○	○	●	C-S	SH	○	●	○	◐[4]
Pika [169]	2022	●	◐[2]	○	○	○	○	○	FSS	O	A	●	●	◐[6]	◐[4]
SecFloat [140]	2022	●	○	○	○	●	○	○	●	O	SH	●	●	◐[6]	●
Llama [77]	2022	●	◐[2]	○	○	○	○	○	FSS	O	SH	○	●	●	●
MPCFormer [104]	2023	●	○	○	●	○	○	○	●	O	SH	○	●	○	●
Primer [193]	2023	●	○	○	●	●	●	○	●	C-S	SH	○	●	○	◐[4]
FastSecNet [79]	2023	●	◐[2]	○	○	○	○	○	FSS	C-S	SH	○	●	○	◐[4]
Rathee et al. [139]	2023	●	○	○	○	●	○	○	●	O	SH	●	●	○	●
Orca [91]	2023	●	◐[2]	○	○	○	○	○	FSS	O	SH	●	●	●	●
FssNN [177]	2023	●	○	○	○	●	○	○	FSS	O	SH	●	●	◐[7]	◐[4]
Sigma [76]	2023	●	◐[2]	○	○	○	○	○	FSS	O	SH	○	●	●	●
Bolt [129]	2023	●	○	○	●	●	○	○	●	C-S	SH	○	●	○	●
Nimbus [107]	2024	●	○	○	●	●	○	○	●	C-S	SH	○	●	○	●
SIMC 2.0 [174]	2024	●	○	○	●	●	●	●	●	C-S	A^C	○	●	○	◐[4]
Fregata [179]	2024	●	○	○	●	○	○	○	●	C-S	SH	○	●	○	◐[4]
SecFromer [119]	2024	●	◐[2]	○	○	○	○	○	●	O	SH	○	●	○	◐[4]
MLFormer [112]	2024	●	○	○	●	○	○	○	●	O	SH	○	●	○	◐[4]
BumbleBee [117]	2025	●	○	○	●	○	○	○	●	C-S	SH	○	●	○	●
SHAFT [93]	2025	●	◐[2]	○	○	○	○	○	●	O	SH	○	●	○	●
Panther [69]	2025	●	○	○	●	●	○	○	●	C-S	SH	○	●	◐[6]	◐[4]
Guo et al. [74]	2025	●	○	○	○	●	○	○	●	O	SH	○	○	●	●
Shark [75]	2025	●	◐[2]	○	○	○	○	○	FSS	O	A	○	●	●	●

[1] Training of LinReg and LogReg only. [2] Dealer in the offline phase. [3] Online phase. [4] Implementation not public.
[5] Federated fine-tuning. [6] Only primitives (no ML functionalities). [7] No costs for offline phase.

TABLE 10: Systematic overview of 3PC and 4PC frameworks. (●) denotes availability, (◐) partial availability and (○) unavailability. Abbreviations: O: Outsourcing, C-S: client-server. Additional abbreviations are defined in Table 2.

Framework		MPC										ML		Evaluation	
		N of parties			Techniques					Setting	Sec.				
Scheme	Year	2	3	4	HE	OT	GC	ZK	SS			Training	Inference	Theoretical	Experimental
ABY3 [124]	2018	○	●	○	○	●	●	○	●	O	SH/A	●	●	◐[5]	●
SecureNN [170]	2018	○	●	○	○	○	○	○	●	O	SH/A	●	●	●	●
ASTRA [39]	2019	◐[3]	●	○	○	●	●	○	●	O	SH/A/F	○	●	◐[5]	◐[4]
Cryptflow [101]	2019	○	●	○	○	○	○	○	●	O	SH/A	○	●	●	●
Shen et al. [154]	2020	○	●	○	○	○	○	○	●	C-S/O	SH	○	●	○	◐[4]
Falcon [171]	2020	○	●	○	○	○	○	○	●	O	SH/A	●	●	◐[5]	●
BLAZE [131]	2020	○	●	○	○	●	●	●	●	O	F	●	●	●	◐[4]
pMPL [156]	2022	●	●	○	○	○	○	○	●	O	SH	●	●	◐[6]	●
AdaminPrivate [12]	2022	○	●	○	○	○	○	○	●	O	SH/A	●	●	●	◐[4]
Meteor [62]	2023	○	●	○	○	○	○	○	●	O	SH	○	●	◐[6]	●
PrivFormer [7]	2023	○	●	○	○	○	○	○	●	O	SH	○	○	●	◐[4]
Trio [81]	2024	○	●	○	○	○	○	○	●	O	SH	○	●	◐[5]	●
Brüggemann et al. [33]	2024	○	●	○	○	○	○	●	●	C-S/O	A	○	●	◐[5]	◐[4]
Mosformer [43]	2025	○	●	○	○	○	○	○	●	O	A	○	●	●	●
Trident [40]	2019	○	◐[3]	●	○	●	●	○	●	O	F	●	●	●	◐[4]
PrivPy [105]	2019	○	○	●	○	●	○	○	●	O	SH	●	●	○	◐[4]
FLASH [34]	2020	○	○	●	○	○	○	○	●	O	A/R	○	●	●	◐[4]
SWIFT [98]	2020	○	●	●	○	○	○	●	●	O	R	◐[1]	●	●	◐[4]
Fantastic Four [51]	2021	○	●	●	○	○	○	○	●	O	A/R	●	●	◐[5]	◐[4]
Tetrad [100]	2021	◐[3]	○	●	○	●	●	○	●	O	F/R	●	●	◐[5]	◐[4]
Quad [81]	2024	○	○	●	○	○	○	○	●	O	A	○	●	◐[5]	●

[1] Training of LinReg and LogReg only. [3] Online phase. [4] Implementation not public.
[5] Only primitives (no ML functionalities). [6] No costs for offline phase.

TABLE 11: Systematic overview of N-party ML frameworks. (●) denotes availability, (◐) partial availability and (○) unavailability. Abbreviations: O: Outsourcing. Additional abbreviations are defined in Table 2.

Framework		MPC								ML		Evaluation	
		Security	Techniques					Setting	Sec.				
Scheme	Year	HM/DM	HE	OT	GC	ZK	SS			Training	Inference	Theoretical	Experimental
MPClan [99]	2023	HM	○	○	○	○	●	O	SH/A	○	●	●	●
Baccarini et al. [15]	2023	HM	○	○	○	○	●	O	SH	○	●	●	●
Liu et al. [110]	2024	HM	○	○	○	○	●	O	SH	○	●	●	◐[1]
MD-ML [183]	2024	DM	○	●	○	○	●	O	A	○	●	◐[2]	●
MD-SONIC [190]	2025	DM	○	●	○	●	●	O	A	○	●	◐[2]	●
FANNG-MPC [2]	2025	DM	●	●	●	●	●	O	A	○	●	○	●

[1] Implementation not public. [2] No costs for offline phase.

TABLE 12: Overview of supported ML functionalities in 2PC frameworks. (●) indicates explicit protocol support, (◐) denotes availability of underlying primitives, and ○indicates no support.

Framework		MPC	ML							
Scheme	Year	N of parties	Supported layers							
			Linear	Conv	Truncation	ReLU	SoftMax	Sigmoid	Normalisation	GeLU
SecureML [125]	2017	2	●	●	Probabilistic	●	●	○	○	○
DeepSecure [149]	2017	2	●	●	○	◐[2]	◐[2]	◐[2]	○	○
MiniONN [111]	2017	2	●	●	●	●	○	●	○	○
Chameleon [146]	2017	2	●	●	●	●	○	●	○	○
Gazelle [92]	2018	2	●	●	○	●	○	○	○	○
Quotient [6]	2019	2	●	●	○	●	○	○	◐	○
Delphi [123]	2020	2	●	●	Probabilistic	●	○	○	○	○
Cryptflow2 [142]	2020	2	●	●	Faithful	●	○	○	○	○
MediSC [113]	2021	2	●	●	Probabilistic	●	○	○	○	○
COINN [89]	2021	2	●	●	○	●	○	○	○	○
Muse [103]	2021	2	●	●	●	●	○	○	○	○
AriaNN [150]	2021	2	●	●	Probabilistic	●	○	○	●	○
SiRnn [141]	2021	2	●	●	Faithful	●	◐[1]	●	●	○
ABY2.0 [130]	2021	2	●	●	Probabilistic	●	●	●	○	○
SONIC [114]	2022	2	●	●	Probabilistic	●	○	○	○	○
Sphinx [162]	2022	2	●	●	○	○	○	○	○	○
SIMC [38]	2022	2	●	●	○	◐[2]	○	○	○	○
Cheetah [88]	2022	2	●	●	Stochastic	●[1]	○	○	○	○
Iron [78]	2022	2	●	●	Faithful	○	●	○	●	●
Pika [169]	2022	2	○	○	Probabilistic	●	●	●	●	○
SecFloat [140]	2022	2	◐[1]	◐[1]	Faithful	◐[1]	◐[1]	◐[1]	●	○
Llama [77]	2022	2	●	●	Faithful	●	●	●	◐[1]	○
MPCFormer [104]	2023	2	●	○	not specified	○	●	○	○	●
Primer [193]	2023	2	●	●	not specified	○	○	○	○	○
FastSecNet [79]	2023	2	●	●	Probabilistic	●	○	○	○	○
Rathee et al. [139]	2023	2	●	●	Faithful	○	○	○	○	○
Orca [91]	2023	2	●	●	Stochastic	●	●	○	○	○
FssNN [177]	2023	2	●	●	not specified	●	○	○	○	○
Sigma [76]	2023	2	●	●	Faithful	●	○	●	●	●
Bolt [129]	2023	2	●	●	Probabilistic	○	●	○	●	●
Nimbus [107]	2024	2	●	●	Stochastic	○	●	○	○	●
SIMC 2.0 [174]	2024	2	●	●	○	◐[2]	○	○	○	○
Fregata [179]	2024	2	●	●	not specified	◐[1]	○	○	○	○
SecFormer [119]	2024	2	●	○	Probabilistic	○	●	○	●	●
MLFormer [112]	2025	2	●	●	Probabilistic	○	◐[1]	◐[1]	◐[1]	◐[1]
BumbleBee [117]	2025	2	●	●	Stochastic	○	●	○	○	●
SHAFT [93]	2025	2	●	○	Probabilistic	○	●	○	○	●
Panther [69]	2025	2	●	●	Stochastic	◐[1]	○	○	○	○
Guo et al. [74]	2025	2	○	○	All	○	○	○	○	○
PriFFT [182]	2025	2	●	●	Probabilistic	◐[1]	●	◐[1]	◐[1]	◐[1]
Shark [75]	2025	2	●	●	Faithful	●	●	●	○	●

[1] Proposes underlying primitives needed for evaluation, but no dedicated protocol for the functionality. [2] Garbled Circuits.

TABLE 13: Overview of supported ML functionalities in multi-party frameworks. (●) indicates explicit protocol support, (◐) denotes availability of underlying primitives, and ○indicates no support.

Framework		MPC	ML							
			Supported layers							
Scheme	Year	N of parties	Linear	Conv	Truncation	ReLU	SoftMax	Sigmoid	Normalisation	GeLU
ABY3 [124]	2018	3	●	○	Probabilistic	●	○	●	○	○
SecureNN [170]	2018	3	●	●	Probabilistic	●	○	●	●	○
ASTRA [39]	2019	3	●	○	○	◐*	○	●	○	○
Cryptflow [101]	2019	3	●	●	Probabilistic	●	○	○	○	○
Shen et al. [154]	2020	3	●	●	Probabilistic	●	○	●	○	○
Falcon [171]	2020	3	●	●	Probabilistic	●	○	○	●	○
BLAZE [131]	2020	3	●	●	Probabilistic	●	○	●	○	○
pMPL [156]	2022	3	●	○	Probabilistic	●	○	●	○	○
AdaminPrivate [12]	2022	3	●	●	Faithful	●	●	○	●	○
Meteor [62]	2023	3	●	●	Probabilistic	●	○	○	○	○
PrivFormer [7]	2023	3	●	○	Probabilistic	●	○	○	●	○
Trio [81]	2024	3	●	○	Probabilistic	◐[1]	○	○	○	○
Brüggemann et al. [33]	2024	3	●	●	Probabilistic	●	○	○	○	○
Mosformer [43]	2025	3	●	●	Probabilistic	●	●	○	●	●
Trident [40]	2019	4	●	●	Probabilistic	●	●	●	◐[2]	○
PrivPy [105]	2019	4	●	●	Probabilistic	◐[1]	○	●	●	○
FLASH [34]	2020	4	●	●	Probabilistic	●	○	●	○	○
SWIFT [98]	2020	3/4	●	●	Probabilistic	●	○	●	○	○
Fantastic Four [51]	2021	3/4	●	●	Stochastic	○	○	○	○	○
Tetrad [100]	2021	4	●	●	Probabilistic	●	●	●	◐[2]	○
Quad [81]	2024	4	●	○	Probabilistic	◐[1]	○	○	○	○
MPClan [99]	2023	n	●	●	Probabilistic	●	○	○	○	○
Baccarini et al. [15]	2023	n	●	●	Probabilistic	○	○	○	○	○
Liu et al. [110]	2024	n	●	●	Stochastic	●	○	○	○	○
MD-ML [183]	2024	n	●	●	Probabilistic	◐[1]	○	○	○	○
MD-SONIC [190]	2025	n	●	●	Probabilistic	●	○	○	○	○
FANNG-MPC [2]	2025	n	●	●	Probabilistic	●	◐[1]	○	◐[1]	○

[1] Proposes underlying primitives needed for evaluation, but no dedicated protocol for the functionality. [2] Garbled Circuits.

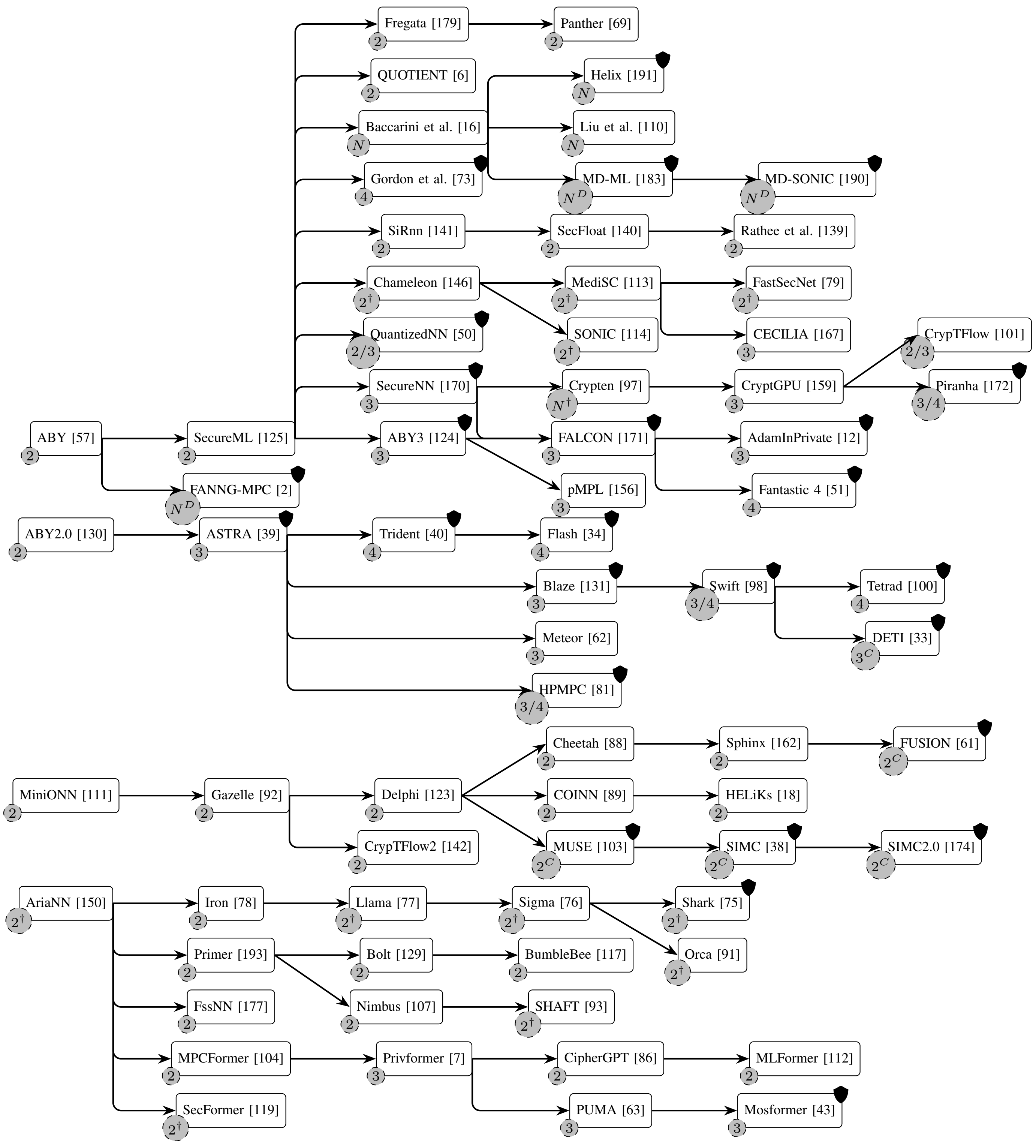

Figure 1: Genealogy of MPC protocols and frameworks for PPML. Shield icons indicate a malicious security model. Numbers in circles show the number of parties. Superscript † marks use of a helper in the offline phase; D denotes a dishonest-majority setting; C indicates a known corrupted party in an asymmetric setting. Protocols supporting both semi-honest and malicious models are shown as malicious.

TABLE 14: Categorization of PPML frameworks by MPC design. ‘’-’ denotes not applicable.

Work	Year	Algebraic Structure		Threat Model			Execution Phase			Deployment Mode		Network		
		Ring	Field	DM	HM	Super HM	Online-only	Pre-processing	Dealer	Outsourcing	Client-Server	High Throughput	Low Latency	Mixed
SecureML [125]	2017	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-independent	TTP	●	○	●	○	○
DeepSecure [149]	2017	$\mathbb{Z}_2$	○	SH	○	○	○	f-dependent	○	●	○	○	●	○
MiniONN [111]	2017	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-independent	○	○	●	○	●	○
Chameleon [146]	2017	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	○	TTP	●	○	●	○	○
Gazelle [92]	2018	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-independent	○	○	●	○	●	○
ABY3 [124]	2018	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	SH/A	○	○	f-independent	○	●	○	●	○	○
SecureNN [170]	2018	$\mathbb{Z}_{2^\ell}$	$\mathbb{F}_p$	○	SH/A	○	●	○	○	●	○	●	○	○
ASTRA [39]	2019	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	SH/A/F	○	○	f-dependent	○	●	○	●	no	○
Cryptflow [101]	2019	$\mathbb{Z}_{2^\ell}$	$\mathbb{F}_p$	○	SH/A	○	●	○	○	●	○	●	○	○
Trident [40]	2019	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	○	F	○	f-dependent	○	●	○	●	○	○
PrivPy [105]	2019	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	○	SH	●	○	○	●	○	●	○	○
Delphi [123]	2020	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-independent	○	○	●	○	○	●
Cryptflow2 [142]	2020	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	●	○	○	○	●	○	○	●
Shen et al. [154]	2020	$\mathbb{Z}_{2^\ell}$	○	○	SH	○	●	○	○	●	●	●	○	○
Falcon [171]	2020	$\mathbb{Z}_{2^\ell}$	$\mathbb{F}_p$	○	SH/A	○	○	f-independent	○	●	○	●	○	○
BLAZE [131]	2020	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	A/F	○	○	f-dependent	○	●	○	●	○	○
FLASH [34]	2020	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	○	A/R	●	f-dependent	○	●	○	●	○	○
SWIFT [98]	2020	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	R	R	○	f-dependent	○	●	○	●	○	○
MediSC [113]	2021	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-dependent	TTP	○	●	●	○	○
COINN [89]	2021	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-dependent	○	○	●	○	○	●
Muse [103]	2021	$\mathbb{Z}_2$	$\mathbb{F}_p$	A^C	○	○	○	f-dependent	○	○	●	○	○	●
AriaNN [150]	2021	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	○	TTP	●	○	○	●	○
Sirnn [141]	2021	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	●	○	○	●	○	●	○	○
ABY2 [130]	2021	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-dependent	○	●	○	●	○	○
Fantastic Four [51]	2021	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	○	A/R	●	f-independent	○	●	○	●	○	○
Tetrad [100]	2021	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	F/R	○	○	f-dependent	○	●	○	●	○	○
SONIC [114]	2022	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	○	TTP	●	○	●	○	○
Sphinx [162]	2022	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	f-dependent	○	○	●	○	●	○
Simc [38]	2022	$\mathbb{Z}_2$	$\mathbb{F}_p$	A^C	○	○	○	f-independent	○	○	●	○	○	●
Cheetah [88]	2022	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	●	○	○	○	●	○	○	●
Iron [78]	2022	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	●	○	○	○	●	●	○	○
Pika [169]	2022	$\mathbb{Z}_{2^\ell}$	○	A	○	○	○	○	●	●	○	○	●	○
SecFloat [140]	2022	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	●	○	○	●	○	●	○	○
Llama [77]	2022	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	○	TTP	●	○	○	●	○
pMPL [156]	2022	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	○	TTP	●	○	○	●	○
AdamInPrivate [12]	2022	○	$\mathbb{F}_p$	○	A	○	○	f-independent	○	●	○	●	○	○
MPCFormer [104]	2023	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	f-independent	○	○	●	●	○	○
Primer [193]	2023	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	f-dependent	○	○	●	○	○	●
FastSecNet [79]	2023	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	○	TTP	○	●	○	●	○
Rathee et al. [139]	2023	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	●	○	○	●	○	●	○	○
Orca [91]	2023	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	○	TTP	●	○	○	●	○
FssNN [177]	2023	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	f-dependent	○	●	○	○	●	○
Sigma [76]	2023	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	○	TTP	●	○	○	●	○
BOLT [129]	2023	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	●	○	○	○	●	○	○	●
Meteor [62]	2023	$\mathbb{Z}_{2^\ell}$	$\mathbb{F}_p$	○	SH	○	○	f-dependent	○	●	○	●	○	○
PrivFormer [7]	2023	$\mathbb{Z}_{2^\ell}$	$\mathbb{F}_p$	○	SH	○	○	f-independent	○	●	○	●	○	○
Baccarini et al. [15]	2023	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	SH	○	○	f-independent	○	●	○	●	○	○
MPClan [99]	2023	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	SH/A	○	○	f-dependent	○	●	○	●	○	○
Force [48]	2023	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	○	SH	○	f-independent	○	●	○	●	○	○
Fusion [61]	2023	-	-	A^S	○	○	-	-	-	○	●	-	-	-
Nimbus [107]	2024	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	f-dependent	○	○	●	○	●	○
SIMC2.0 [174]	2024	$\mathbb{Z}_2$	$\mathbb{F}_p$	A^C	○	○	○	f-independent	○	○	●	○	○	●
Fregata [179]	2024	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	○	f-dependent	○	○	●	○	●	○
SecFromer [119]	2024	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	○	TTP	●	○	●	○	○

Continued on next page

TABLE 14: Categorization of PPML frameworks by MPC design (continued).

Work	**Year**	**Algebraic Structure**		**Threat Model**			**Execution Phase**			**Deployment Mode**		**Network**		
		Ring	Field	DM	HM	Super HM	Online-only	Pre-processing	Dealer	Outsourcing	Client-Server	High Throughput	Low Latency	Mixed
MLFormer [112]	2024	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	○	TTP	●	○	●	○	○
Trio [81]	2024	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	SH	○	○	f-dependent	○	●	○	●	○	○
Quad [81]	2024	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	○	A	○	f-dependent	○	●	○	●	○	○
Brüggemann et al. [33]	2024	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	○	A	○	○	f-dependent	○	●	●	●	○	○
Liu et al. [110]	2024	○	$\mathbb{F}_p$	○	SH	○	○	f-independent	○	●	○	●	○	○
MD-ML [183]	2024	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	A	○	○	○	f-dependent	○	●	○	●	○	○
BumbleBee [117]	2025	$\mathbb{Z}_{2^\ell}$	○	SH	○	○	●	○	○	○	●	○	●	○
SHAFT [93]	2025	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	○	TTP	●	○	●	○	○
Panther [69]	2025	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	●	○	○	○	●	○	○	●
Guo et al. [74]	2025	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	●	○	○	●	○	●	○	○
Shark [75]	2025	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	A	○	○	○	○	TTP	●	○	○	●	○
MD-SONIC [190]	2025	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	A	○	○	○	f-independent	○	●	○	●	○	○
Fanng-MPC [2]	2025	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	$\mathbb{F}_p$	A	A	A	○	○	Distributed	●	○	●	●	●
Matchmaker [90]	2025	$\mathbb{Z}_{2^\ell}, \mathbb{Z}_2$	○	SH	○	○	○	○	TTP	●	○	○	○	●
Mosformer [43]	2025	$\mathbb{Z}_{2^\ell}$	○	○	A	○	○	f-dependent	○	●	○	○	●	○

TABLE 15: Theoretical cost analysis in total bits for dot-product functionality with two parties. Superscript Tr denotes a cost for a protocol for dot-product with truncation. Notation and abbreviations are defined in Table 2.

N	Sec.	Protocol	Communication Offline	Communication Online	Tech
		Chameleon [146]	2ℓ	$4n\ell$	STP
		SecureML [125]	$2n\ell(\kappa+\ell)$	$4n\ell$	SS
		Delphi [123]	$\mathcal{O}(n\mathsf{CT}_{\mathsf{Pai}})$	$n\ell$	LHE
		Cheetah [88]	-	$\mathcal{O}(n\mathsf{CT}_{\mathsf{BFV}})$	FHE
		MediSC [113]	$5n\ell$	$n\ell$	STP
		COINN [89]	$n\ell^2$	$n\ell$	OT
		SONIC [114]	$(4n+2)\ell$	$4n\ell$	STP
		Sphinx [162]	$\mathsf{CT}_{\mathsf{CKKS}}$	$n\ell$	FHE
		AriaNN [150]	$(4n+2)\ell$	$4n\ell$	STP
		ABY2.0 [130]	$2n\ell(\kappa+\ell)$	2ℓ	SS
		SirNN [141]	-	$\mathcal{O}(n\Lambda\ell)$	OT
	SH	pMPL [156]	$\mathcal{O}(n\ell)$	$6n\ell$	SS
		pMPL$^{\mathsf{Tr}}$ [156]	$\mathcal{O}(n\ell\log\ell)$	$6n\ell+\ell$	SS
		Iron [78]	-	$\mathcal{O}(\sqrt{n}\mathsf{CT}_{\mathsf{BFV}})$	FHE
		SecFloat [140]	-	$\mathcal{O}(\Lambda n(\mu+\nu))$	SS
		Llama [77]	$\mathcal{O}(n\Lambda(\mu+\nu))$	$4\ell+2\mu+2\nu$	STP
		Primer [193]	$\mathcal{O}(\mathsf{CT}_{\mathsf{BFV}})$	$n\ell$	FHE
		FastSecNet [79]	$2n\ell$	$n\ell$	STP
		Rathee et al. [139]	-	$\mathcal{O}(\Lambda n\ell)$	SS
		FssNN [177]	$\mathcal{O}(n\ell)$	$4n\ell$	OT
		Sigma [76]	$(4n+2)\ell$	$4n\ell$	STP
		Bolt [129]	-	$\mathcal{O}(n\mathsf{CT}_{\mathsf{BFV}})$	FHE
		CipherGPT [86]	$\mathcal{O}(\mathsf{CT}_{\mathsf{BFV}})$	$n\ell$	FHE, OT
		Nimbus [107]	-	$\mathcal{O}(\mathsf{CT}_{\mathsf{BFV}})$	FHE
		BumbleBee [117]	-	$\mathcal{O}(n\mathsf{CT}_{\mathsf{BFV}})$	FHE

Appendix D. ML Architecture based Systematization

This section presents supplementary details on the ML-architecture-based systematization introduced in Section 4.

D.1. Multiplication and Convolution

In this section, we outline the main approaches to evaluating multiplications and convolutions, which form the computational core of linear layers. We distinguish between MPC-based methods (Appendix D.1.1) and MPC-HE-based methods (Appendix D.1.2), the latter being predominantly used in asymmetric client–server scenarios. Appendix D.1.3 provides a detailed discussion of HE-based encoding schemes for efficient matrix multiplication and convolution. We omit a dedicated analysis of MPC-based matrix and convolution evaluation, as their construction follows naturally from the techniques described in Appendix D.1.1, as discussed in Section 4.1.2. Table 15 summarizes the costs of various 2PC dot-product protocols, while Table 16 presents results for settings with more than two parties.

D.1.1. MPC-based Multiplication. In this section, we analyze multiplication methods based on MPC techniques, organized by the number of parties involved—two, three, four, and general N-party settings—since the available approaches vary significantly with the assumed corruption thresholds.

Two-party setting. SecureML [125] was the first ML-focused framework for two-party computation, building on ABY [57]. ABY introduced mixed-protocol computation with efficient conversions across arithmetic, Boolean, and garbled circuit domains. Multiplication relies on Beaver triples [20], generated either via LHE-based pre-processing (using Paillier [128] or DGK [54]), which reduces communication but incurs costly public-key operations, or via OT-based methods, which are cheaper computationally but increase communication and rounds. SecureML further extended Beaver triples to matrix-matrix multiplication, greatly improving efficiency for large models.

Chameleon [146] proposed an alternative based on the Du–Atallah protocol [64], originally from Sharemind, which uses a semi-honest helper. Chameleon shifts this helper's role to the offline phase, an idea later generalized via function secret sharing (FSS) [28], where the helper provides FSS keys (essentially Beaver-like randomness). This approach underlies several recent frameworks [76], [79], [150], [169], [182].

Among secret-sharing-based protocols, ABY2.0 introduced a masked sharing scheme leveraging function-dependent preprocessing and achieved constant online communication for dot products. Subsequent works optimized communication via mixed-precision protocols: SirNN [141] extended Beaver-triple-style multiplication to mixed bit-widths using OT-based preprocessing, further advanced by SecFloat [140] and Rathee et al. [139] for floating-point

TABLE 16: Theoretical cost analysis in total bits for dot-product functionality in three, four and N party settings. Superscript Tr denotes a cost for a protocol for dot-product with truncation. Notation and abbreviations are defined in Table 2.

N	Sec.	Protocol	Communication Offline	Communication Online	Tech
3	SH	ASTRA [39]	ℓ	2ℓ	SS
		Falcon [171]	-	3ℓ	SS
		Meteor [62]	$3n\ell$	3ℓ	SS
		SecureNN [170]	-	$(8n+2)\ell$	SS
		Cryptflow [101]	-	$(4n+2)\ell$	SS
		pMPL [156]	$\mathcal{O}(n\ell)$	$12n\ell$	SS
		pMPL$^{\mathsf{Tr}}$ [156]	$\mathcal{O}(n\ell)$	$12n\ell+2\ell$	SS
		AdaminPrivate [12]	-	3ℓ	SS
		CECILIA [167]	$6n\ell$	$4n\ell$	Helper
	A	Boyle et al. [29]	-	3ℓ	ZK
		ABY3$^{\mathsf{Tr}}$ [124]	$\mathcal{O}(n\ell)$	$(9n+3)\ell$	SS
		AdaminPrivate [12]	-	$6n\ell$	SS
	F	ASTRA [39]	$21n\ell$	$(2n+2)\ell$	SS
		BLAZE [131]	$3n\ell$	3ℓ	ZK
		BLAZE$^{\mathsf{Tr}}$ [131]	$(3n+2)\ell$	3ℓ	ZK
	R	SWIFT [98]	3ℓ	3ℓ	ZK
		SWIFT$^{\mathsf{Tr}}$ [98]	15ℓ	3ℓ	ZK
		Fantastic Four [51]	-	$6(\ell+s)$	ZK
		Fantastic Four$^{\mathsf{Tr}}$ [51]	$\mathcal{O}(\ell+s)$	$9\ell+6s$	ZK
4	SH	Force$^{\mathsf{Tr}}$ [48]	-	4ℓ	SS
		PrivPy$^{\mathsf{Tr}}$ [105]	-	$8n\ell$	SS
	F	Gordon et al. [73]	4ℓ	2ℓ	SS
		Trident [40]	3ℓ	3ℓ	SS
		Trident$^{\mathsf{Tr}}$ [40]	6ℓ	3ℓ	SS
		Tetrad$^{\mathsf{Tr}}$ [100]	2ℓ	3ℓ	SS
	R	FLASH [34]	6ℓ	6ℓ	SS
		FLASH$^{\mathsf{Tr}}$ [34]	8ℓ	6ℓ	SS
		SWIFT [98]	3ℓ	3ℓ	SS
		SWIFT$^{\mathsf{Tr}}$ [98]	4ℓ	3ℓ	SS
		Fantastic Four [51]	-	6ℓ	SS
		Fantastic Four$^{\mathsf{Tr}}$ [51]	ℓ	9ℓ	SS
		Fantastic Four$^{\mathsf{Tr}}$ [51]	2ℓ	16ℓ	SS
		Tetrad$^{\mathsf{Tr}}$ [100]	2ℓ	3ℓ	SS
N	SH	MPClan [99]	$t\ell$	$2t\ell$	SS
		Baccarini et al. [15]	-	$tN\ell$	SS
		Liu et al. [110]	$\frac{3N^2}{t+1}\ell$	$2N\ell$	SS
	A	MPClan [99]	$3t\ell$	$3t\ell$	SS
		MD-ML [183]	$\mathcal{O}(N^2(\ell+s)^2)$	$N(\ell+s)$	SS
		MD-SONIC [190]	$\mathcal{O}(N^2(\ell+s)^2)$	$N(\ell+s)$	SS

arithmetic. In the FSS paradigm, LLAMA [77] achieved vector-size-independent dot products for mixed-bitwidths, a strategy later adopted by Orca [91].

Three-party setting. In the semi-honest setting, ABY3 [124] extended 2PC protocols of SecureML to the 3-party setting using the RSS scheme of Araki et al. [11]. SecureNN introduced Beaver-triple-based multiplication with a helper, which CrypTFlow [101] optimized using pair-wise PRNGs and further extended with a compiler that upgrades semi-honest protocols to malicious security via trusted execution environments (TEEs). Other frameworks, including Falcon [171], SecureQ8 [50], and PUMA [63], leverage RSS to enable efficient one-round multiplication and dot-product protocols. In an orthogonal line of research, works like ASTRA [39], Meteor [62], and Trio [81] reduce online cost further via masked RSS, enabling two-party online computation. This approach is also adopted by Lu et al. [116], while pMPL [156]and SecureCART [164] explore a privileged-party model for reconstruction using BT-based multiplication in a vector-space sharing setting.

For malicious security, RSS-based multiplication protocols largely build on the work of Araki et al. [11], employing verified Beaver triples or cut-and-choose techniques [10], [46], [65], [70]. Frameworks such as ABY3, Falcon, and ASTRA extend these ideas to PPML protocols over rings. More recent approaches, including Boneh et al. [23], Boyle et al. [29], BLAZE [131], and SWIFT [98], leverage distributed ZK proofs to achieve communication-efficient dot products, albeit with higher computational cost. Among these, protocols like ASTRA, BLAZE, and SWIFT rely on function-dependent preprocessing to reduce online communication in multiplication and dot products.

Finally, DEtI [33] adapts 2PC client–server protocols by adding a semi-honest client, thereby tolerating one malicious corruption in an honest-majority setting. Its design builds on ASTRA [39], combining the triple-sacrifice technique of Eeriksen et al. [65] with distributed ZK proofs from [23], [29].

Four-party setting. In the 4PC setting, the presence of an additional honest party eliminates expensive pre-processing of a malicious 3PC protocols and enables constant communication for dot products. Consequently, all 4PC frameworks considered here achieve preprocessing and online communication costs independent of vector size. The common structure across protocols is that parties first compute locally, obtaining the output in a different sharing format, and then perform a re-sharing step.

In the semi-honest model, PrivPy [105] introduced a 4PC RSS-based protocol, mirroring the 3PC approach: local computation followed by re-sharing to RSS. Force [48] minimized communication by using a 2-party additive sharing scheme, where multiplication is local and re-sharing requires only one ring element per party. Gordon et al. [73] were the first to propose a 4PC framework with abort and GOD security, though relying on costly public-key cryptography. Trident [40] introduced the first concretely efficient malicious PPML framework by exploiting three honest parties, avoiding these expensive techniques.

Robustness in 4PC frameworks typically depends on identifying a trusted party. FLASH [34], SWIFT [98], and Tetrad [100] all follow this approach, adapting it to their respective sharing schemes. Quad [81] builds on Tetrad's scheme to support heterogeneous networks, with security against abort. Fantastic Four [51] considered a restricted form of *private robustness*, wherein player elimination is performed instead of relying on a trusted party in the event of a corruption detection.

Multi-party setting. In the N-party dishonest-majority setting, FANNG-MPC [2] was the first concretely efficient PPML framework. Its key contribution is accelerating pre-processing by relying on multiple untrusted dealers, while trusting only a subset. Linear layers are evaluated using standard Beaver multiplication, with pre-processing based on leveled homomorphic encryption (LHE) in an SPDZ$_{2^k}$-like manner [46]. Concurrently, MD-ML [183] also built on SPDZ$_{2^k}$ techniques but without additional dealers. More recently, MD-SONIC [190] reduced communication complexity further by integrating zero-knowledge proofs.

D.1.2. Hybrid MPC-HE-based Multiplication. In the two-party client-server computation model, linear layers are evaluated using leveled homomorphic encryption (LHE) to accommodate computational asymmetry. In this section we categorize the frameworks based on their high-level approaches to linear layers in general. In Appendix D.1.3 we expand on our analysis focusing on concrete instantiations for evaluating matrix multiplications and specialized protocols for convolutions.

The first framework in this domain, MiniONN [111], introduced oblivious inference, employing standard Paillier encryption. Gazelle significantly improved upon MiniONN by leveraging a more efficient PAHE instantiated with the BFV scheme [68]. Delphi [123] further enhanced online efficiency by shifting the costly LHE operations to the offline phase.

CrypTFlow2 [142] offers two approaches for evaluating linear layers based on the specific setting: an OT-based Beaver triple multiplication method and an LHE-based preprocessing model inspired by Delphi. GALA [188] introduces a 'plug-and-play' HE-based linear layer, optimizing matrix multiplication permutations—identified as a bottleneck in client-server 2PC—to reduce computational overhead. This framework focuses solely on scalar multiplication using PHE based on BFV. Similarly, HELiKs [18] design new approaches for matrix multiplication and 3D convolutions building on RLWE-based LHE schemes.

Cheetah [88] builds upon Delphi to enhance LHE-based linear layer efficiency. The state-of-the-art in semi-honest 2PC client-server model is Panther [69], enhancing the efficiency of linear layers by proposing a custom HE scheme.

All the above frameworks assume a semi-honest security model. However, Muse [103] demonstrated an attack by a malicious client, highlighting the need for stronger security guarantees. To maintain efficient constructions while addressing malicious security, Muse assumes a semi-honest server. This setting has gained popularity, with subsequent works such as SIMC [38] and SIMC2.0 [174] introducing further optimizations. To ensure security against malicious clients, these frameworks rely on SPDZ [56]-style MACs for authenticated shares and zero-knowledge proofs (ZK-proofs) to validate ciphertext formation on the client side. The core structure of these schemes is derived from the Overdrive MPC framework [95], albeit simplified by leveraging a semi-honest server.

On the other hand, a malicious server might learn sensitive information from a client's input, or deliberately evaluate an inference with a lower accuracy model. As a result, Fusion [61] is the only work, to the best of our knowledge, focusing on a setting assuming malicious server and a semi-honest client. They leverage a *cut-and-choose*-like techniques, with the client shuffling its input among random publicly available samples. Upon query result, the client verifies correctness by verifying the results on the public samples. They evaluate linear layers using standard semi-honest protocols from previous works [57], [88], [123], [142].

D.1.3. HE-based Matrix Multiplication and Convolution.

In the following we dive into encoding-specific approaches of hybrid MPC-HE frameworks and their techniques for evaluating large matrix multiplications and convolutions. We differentiate between two main approaches: *SIMD slot-based encoding* and *coefficient encoding*.

Conv/MatMul in SIMD Form. Starting with Gazelle [92], many frameworks use SIMD slot packing for linear layers via the diagonal method: each generalized diagonal of the weight array is encoded in a separate plaintext polynomial. The linear transform is computed by repeatedly rotating the encrypted input to align slots, multiplying by the corresponding diagonal plaintext, and accumulating the partial sums; the result is packed into a single ciphertext [18], [22], [123], [188], [193]. This requires roughly one plaintext–ciphertext multiplication per diagonal and about the same number of rotations, with rotations being the dominant cost. Gazelle mitigates this with hoisted automorphisms (decompose once, reuse across many rotations) to amortize rotation overhead. BOLT [129] reduces expensive homomorphic rotations and communication by (1) introducing a compact column-packed encoding for ciphertext–plaintext multiplication that fully utilizes ciphertext slots, and (2) adapting the baby-step giant-step (BSGS) strategy to perform rotations later on partial sums at the column level. BLB [175] improves private Transformer inference by (1) fusing fine-grained adjacent linear operators to reduce MPC-HE conversions, (2) introducing a secure CKKS-MPC conversion protocol to support fused linear operations under CKKS, and (3) designing a rotation-efficient protocol for fused matrix multiplications that leverages attention structure (multi-head packing) and BSGS techniques to lower homomorphic rotations.

Conv/MatMul in Coefficient Form. Cheetah [88] demonstrated convolution using polynomial multiplication directly. It encodes input tensors as polynomial coefficients and encodes filters in reversed order so that the polynomial product corresponds to the convolution sum. This coefficient representation avoids costly HE rotations entirely. For matrix-vector multiplication (MVM), Cheetah used a row-major coefficient encoding. Rhombus [83] introduced a column-major variant that is more efficient for "tall" matrices. For larger matrix-matrix multiplication (MMM), frameworks extend these ideas: Iron [78] transforms convolutions into equivalent matrix multiplications, then packs matrix blocks into polynomial coefficients such that each HE multiplication yields a partial output block. This approach eliminates ciphertext rotations but requires duplicating certain input data across coefficients, introducing redundancy. BumbleBee [117] builds on Cheetah's $\mathsf{MVM}_{\text{coeff}}$ by employing ciphertext interleaving through PackRLWEs to support general MMM. This enables dense packing of output coefficients and significantly reduces communication overhead by minimizing wasted ciphertext slots and lowering the need for costly homomorphic automorphisms. Each coefficient-based method (Cheetah [88], Rhombus [83], Iron [78], BumbleBee [117]) thus navigates a balance between communication (ciphertext count and size) and computation (HE multiplications and automorphisms), depending on matrix dimensions.

D.2. Truncation

SecureML [125] proposed a 2-party probabilistic truncation protocol requiring only local operations. This efficient method was later adopted in 2PC frameworks such as Delphi [123], AriaNN [150], ABY2.0 [130], and Pika [169], as well as extended to 3- and 4-party settings [48], [81].

CrypTFlow2 [142] introduced the first concretely efficient protocol for faithful truncation beyond garbled circuits. Subsequent works, such as Cheetah [88] and Panther [69], improved performance by relaxing requirements and adopting stochastic truncation. SirNN [141] extended CrypTFlow2's approach to mixed-bitwidth arithmetic, while SecFloat [140] adapted it for floating-point computations. In the FSS domain, LLAMA [77] and Sigma [76] develop faithful truncation protocols based on DCFs and DPFs, respectively. Motivated by security concerns raised by Li et al. [106], Orca [91] developed a secure stochastic truncation protocol.

In the three-party setting, ABY3 [124] generalized SecureML's 2PC probabilistic truncation to replicated secret sharing, building on the works of Catrina and de Hoogh [36] and Damgård et al. [53]. This approach was later adopted in several frameworks [62], [63], [101], [105], [170], [171]. However, the heavy communication cost of its preprocessing phase remains a major bottleneck. To mitigate this, later works proposed improvements leveraging masked secret sharing schemes [34], [40], [98], [131]. MaSTer [185] further eliminates preprocessing by introducing a lightweight post-processing check, allowing an adversary to alter the least significant bit (LSB) of the output. Follow-up work by Zbudila et al. [184] shows that this extended adversarial model remains secure in practical settings.

SecureQ8 [50] introduced a stochastic truncation scheme that avoids the need for a large gap between share magnitudes and the ring size, later adapted by Fantastic Four [51]. To the best of our knowledge, no faithful truncation exists for the multi-party setting over rings.

We summarize the theoretical costs of different truncation protocols in Table 17, including only those frameworks that propose novel truncation methods with sufficient detail for precise cost analysis.

D.3. Non-linear Layers

In this section, we examine protocols for key functionalities in non-linear layers, including ReLU, Sigmoid, Softmax, GELU, and normalization. We first introduce the core primitive underlying these operations—the comparison protocol.

D.3.1. Comparison. SecureML [125], the first work in privacy-preserving machine learning, addresses the challenge of computing non-linear comparison function using mixed-protocol computation from ABY [57], where parties

TABLE 17: Theoretical cost analysis in total bits for truncation functionality. We depict the costs for truncation of k bits in a ring $\mathbb{Z}_{2^\ell}$. Notation and abbreviations are defined in Table 2.

N	Sec.	Protocol	Offline Comm.	Online Comm.	Online Rounds	Result	Tech.
2	SH	SecureML [125]	-	-	-	Probabilistic	SS
		Cryptflow2 [142]	-	$\mathcal{O}(\Lambda\ell)$	$\lceil log(\ell)\rceil+1$	Faithful	OT
		Cheetah [88]	-	$16\ell+11k$	$\lceil log(\ell)\rceil$	Faithful	OT
		Cheetah [88]	-	13ℓ	$\lceil log(\ell)\rceil$	Stochastic	OT
		Zou et al. [199]	-	$\mathcal{O}(\Lambda\ell)$	$\lceil log(\ell)\rceil+1$	Faithful	OT
		SirNN [141]	-	$\mathcal{O}(\Lambda\ell)$	$\log\ell+3$	Faithful	OT
		Orca [91]	$\mathcal{O}(\Lambda(\ell+k))$	$2(\ell-k+1)$	3	Stochastic	FSS
		Shark [75]	$(\Lambda+\kappa)(\ell+k)$	$\ell+s+2$	2	Faithful	FSS
		Guo et al. [74]	-	$\mathcal{O}(\Lambda k)$	$\log k+2$	Faithful	OT
		Guo et al. [74]	-	$\Lambda+k$	2	Stochastic	OT
3	SH	Dalskov et al. [49]	$4\ell-2k$	4ℓ	2	Stochastic	SS
		ABY3 [124]	-	ℓ	1	Probabilistic	SS
		AdaminPrivate [12]	-	$5\ell+5$	4	Stochastic	SS
	A	ABY3 [124]	$20(2\ell-k)$	3ℓ	1	Probabilistic	SS
		MaSTer [185]	-	3ℓ	1	Probabilistic	SS
		AdaminPrivate [12]	-	$30\ell+21\kappa+15$	5	Stochastic	SS
	R	SWIFT [98]	12ℓ	ℓ	1	Probabilistic	SS
		Fantastic Four [51]	$76(\ell+s)+54k+12$	3ℓ	1	Stochastic	SS
4	A	FLASH [34]	-	7ℓ	5	Probabilistic	SS
	F	Trident [40]	3ℓ	ℓ	1	Probabilistic	SS
		Tetrad [100]	ℓ	3ℓ	1	Probabilistic	SS
	R	Fantastic Four [51]	2ℓ	16ℓ	1	Stochastic	SS
		Fantastic Four [51]	ℓ	3ℓ	1	Probabilistic	SS
		FLASH [34]	-	14ℓ	5	Probabilistic	SS
		SWIFT [98]	ℓ	ℓ	1	Probabilistic	SS
N	SH (HM)	Baccarini et al. [16]	$\mathcal{O}(N)$	$Nt(2\ell+1)$	2	Stochastic	SS
		Liu et al. [110]	$3N\ell^2$	$2N\ell$	1	Stochastic	SS
	A (DM)	Escudero et al. [66]	$\mathcal{O}(t^2\log\ell)$	$\mathcal{O}(Nt^2\log\ell)$	$\mathcal{O}(\log t\log\ell)$	Stochastic	SS
		MD-ML [183]	$\mathcal{O}(t^2\log\ell)$	$N(\ell+\sigma)$	1	Probabilistic	SS
		MD-SONIC [190]	$\mathcal{O}(t^2\log\ell)$	$N(\ell+\sigma)$	1	Probabilistic	SS

convert secret shares to the boolean domain to locally extract the most significant bit (MSB). This technique is later refined by ABY2.0 [130] and generalized to the 3-party and 4-party settings by ABY3 [124] and Tetrad [100], as well as Trident [40], respectively. These constructions typically rely on costly ripple carry adder (RCA) circuits or more efficient parallel prefix adders (PPA). While some works integrate such constructions into custom secret sharing schemes [34], [98], [131], others adopt more general-purpose primitives such as edaBits [66] or Rabbit [120], as seen in [48], [51], [156].

Another line of work focuses on evaluation of non-linear comparison using GCs [38], [44], [92], [103], [111], [113], [114], [123], [146], [149], [174].

AriaNN [150] employs a probabilistic comparison based on FSS construction of Boyle et al. [27]. Their failure probability of the comparison protocol depends on the input magnitude relative to the ring size. LLAMA [77] subsequently improves this by designing a deterministic variant based on the FSS construction of Boyle et al. [25]. FssNN [177] advances this line of work by reducing distributed comparison function (DCF) key sizes, and stands out as the only framework to propose dedicated protocols for FSS key generation, thereby eliminating the need for a trusted dealer. SecureCART [164] leverages a setting with a privileged party in combination with FSS based on the construction of Boyle et al. [25].

All of the above techniques suffer from high communication overhead. To address this, several frameworks design protocols within the realm of arithmetic secret sharing, aiming to reduce communication at the expense of increased round complexity. CrypTFlow2 [142] builds on the idea of Garay et al. [72], reducing comparison to several fractional comparisons on input segments evaluated via OTs. Cheetah [88] further optimizes this approach significantly improving communication efficiency.

In contrast, SecureNN [170] simplifies MSB computation by reducing it to the evaluation of the LSB of a doubled value within an odd modulus field, requiring a switch from standard ring-based secret sharing to an odd ring structure.

TABLE 18: Theoretical cost analysis in total bits for ReLU functionality. Notation and abbreviations are defined in Table 2.

N	Sec.	Protocol	Offline Comm.	Online Comm.	Online Rounds	Tech.
2	SH	SecureML [125]	-	$\mathcal{O}(\Lambda\ell)$	2	SS
		Cryptflow2 [142]	-	$\Lambda\ell + 18\ell$	$\log\ell + 2$	OT
		Delphi [123]	$3\mathsf{CT}_{\mathsf{DGK}}$	4ℓ	1	HE
		MediSC [113]	$\mathcal{O}(\ell)$	$\mathcal{O}(\ell)$	$\mathcal{O}(\log\ell)$	SS
		COINN [89]	$\mathcal{O}(\ell^2)$	$\mathcal{O}(\Lambda\ell)$	$\mathcal{O}(1)$	GC
		SONIC [114]	-	$20\ell - 16$	$\log\ell + 2$	SS
		AriaNN [150]	$\mathcal{O}(\Lambda\ell)$	3ℓ	2	FSS
		ABY2 [130]	$\mathcal{O}(\Lambda)$	$230 + 3\ell$	$2 + \log_4\ell$	SS
		Boyle et al. [25]	$(\ell+1)(\Lambda+2\ell)$	2ℓ	1	FSS
		pMPL [156]	-	$8\ell + 2\ell\log\ell$	$\log\ell + 4$	SS
		FastSecNet [79]	$\ell(4\Lambda+\ell)$	2ℓ	1	FSS
		Orca [91]	$\ell(\Lambda+7)+\Lambda+1$	$2\ell + 2$	2	FSS
		FssNN [177]	$\mathcal{O}(\ell\Lambda)$	$2\ell + 1$	2	FSS
		Shark [75]	$\mathcal{O}(\ell(\Lambda+\kappa))$	$\ell + s$	2	FSS
3	SH	SecureNN [170]	-	$8\ell\log p + 24\ell$	10	SS
		Cryptflow [101]	-	$6\ell\log p + 19\ell$	7	SS
		pMPL [156]	-	$18\ell + 4\ell\log\ell$	$\log\ell + 5$	SS
		AdaminPrivate [12]	-	$5\ell_x + 6\ell + 5$	$\ell + 5$	SS
		Bicoptor [197]	-	$(\ell_x+2)\ell + 2$	2	SS
		Bicoptor2 [196]	-	$(\ell_x+1)^2 + 2\ell + 2$	2	SS
		Meteor [62]	$\mathcal{O}(\ell)$	$(4\ell+1)\log p + 3\ell$	$\log_4\ell + 3$	SS
		Falcon [171]	$\mathcal{O}(\ell)$	$3\ell\log p + 3\ell$	$3 + \log\ell$	SS
		CECILIA [167]	$\mathcal{O}(\ell)$	$\mathcal{O}(\ell)$	$\mathcal{O}(1)$	SS
	A	Falcon [171]	$\mathcal{O}(\ell\log\ell)$	$3\ell\log p + 18\ell$	$4 + \log\ell$	SS
		ABY3 [124]	60ℓ	45ℓ	$\log\ell + 3$	SS
		AdaminPrivate [12]	-	$\mathcal{O}(\ell_x\kappa)$	$\ell + 7$	SS
	F	BLAZE [131]	21ℓ	16ℓ	$\log\ell + 3$	SS
	R	SWIFT [98]	$21\ell - 6$	$16\ell - 6$	$\log\ell + 3$	SS
		Fantastic Four [51]	$14(\ell + s)$	$114\ell + 6s + 1$	$\mathcal{O}(\ell)$	SS
4	A	FLASH [34]	6ℓ	17ℓ	$\log\ell + 6$	SS
	F	Trident [40]	$14\ell + 2$	$10\ell + 2$	$\log\ell + 2$	SS
		Tetrad [100]	$13\ell + 2$	$6\ell - 2$	$\log\ell + 2$	SS
	R	FLASH [34]	22ℓ	24ℓ	$\log\ell + 2$	SS
		SWIFT [98]	$13\ell - 2$	$10\ell - 6$	$\log\ell + 1$	SS
		Fantastic Four [51]	-	$44\ell + 1$	$\mathcal{O}(\ell)$	SS
N	SH (HM)	Liu et al. [110]	$N\ell(10\ell + 2)$	$N\ell(2\ell + 6)$	3	SS
	A (DM)	Escudero et al. [66]	$7\ell\kappa$	$7\ell + 3s$	$\mathcal{O}(\log\ell)$	SS
		MD-ML [183]	$\mathcal{O}(N^2(\ell+s)^2)$	$\mathcal{O}(\ell s)$	$\log\ell + 2$	SS
		MD-SONIC [190]	$\mathcal{O}(N^2(\ell+s)^2)$	$\mathcal{O}(N(\ell+s))$	$\log(\ell-1) + 1$	SS

This technique is also leveraged in CrypTFlow [101], Falcon [171], Meteor [62], or AdamInPrivate [12].

Finally, the Bicoptor family of works [196], [197] evaluate comparison using multiple (local) truncations with the help of a "helper" party present in the online phase.

D.3.2. ReLU. The rectified linear unit (ReLU), defined as

$$\mathrm{ReLU}(x) = \max(0, x)\,,$$

is the predominant activation in contemporary deep learning, used extensively across neural network and transformer architectures.

In the following we outline the main approaches for evaluation of ReLU in MPC. We present the theoretical complexity costs of different frameworks for the ReLU functionality in Table 18. Only frameworks that thoroughly describe their ReLU protocol and allow for precise cost analysis are included.

Comparison + Multiplication. The ReLU function can be written as $\mathrm{ReLU}(x) = \mathbf{1}(x > 0) \cdot x$, where $\mathbf{1}(X)$ is the indicator function that returns 1 if the statement X is true and 0 otherwise. As a result, the most common approach for evaluating ReLU comprises evaluation of the comparison function followed by multiplication. This approach is widely adopted across multiple frameworks [77], [98], [124], [125],

TABLE 19: Theoretical cost analysis in total bits for sigmoid functionality. Notation and abbreviations are defined in Table 2.

N	Sec.	Protocol	Offline Comm.	Online Comm.	Online Rounds	Tech.
2	SH	SecureML [125]	-	$\mathcal{O}(\Lambda\ell)$	4	SS
		ABY2 [130]	$\mathcal{O}\Lambda$	$460+4\ell$	$2+\log_4\ell$	SS
		Zheng et al. [194]	10ℓ	$2(k+5)$	1	STP
		pMPL [156]	-	$18\ell+4\ell\log\ell$	$\log\ell+5$	SS
3	SH	pMPL [156]	-	$38\ell+8\ell\log\ell$	$\log\ell+6$	SS
	A	ABY3 [124]	108ℓ	81ℓ	$\log\ell+4$	SS
	F	BLAZE [131]	$(5\Lambda+23)\ell$	$(\Lambda+11)\ell$	5	SS
	R	SWIFT [98]	$39\ell-9$	$29\ell-9$	$\log\ell+4$	SS
4	F	Trident [40]	$15\ell+7$	$16\ell+7$	5	SS
		Tetrad [100]	$20\ell+3$	$9\ell-4$	$\log\ell+2$	SS
	R	FLASH [34]	26ℓ	113ℓ	$3\log\ell+20$	SS
		SWIFT [98]	$23\ell-1$	$20\ell-9$	$\log\ell+2$	SS

[130], [131], [150], [170], [171], [196], [197].

Garbled Circuits. Several works, mainly in the client-server setting, evaluate the ReLU function entirely in GCs [44], [92], [103], [111], [146], [149]

Approximation. Delphi [123] approximates ReLU with the square function $\mathrm{ReLU}(x) \approx x^2$ and combines it with precise GC evaluation to minimize communication overhead maximizing accuracy, thanks to their strategic planner choosing which activation can be cheaply approximated not to affect accuracy. MPCFormer [104] avoids comparisons entirely and approximates ReLU with a polynomial

$$\mathrm{ReLU}(x) \approx 0.125x^2 + 0.25x + 0.5\,.$$

D.3.3. Sigmoid. The *sigmoid function*, also known as the *logistic function*, is widely used in logistic regression and as the activation function in the final layer of binary classification neural networks.

The logistic function is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}}\,.$$

Below we describe the three main approaches to evaluating the *sigmoid* function. In Table 19, we report the theoretical costs of Sigmoid across different frameworks, restricting attention to those that provide sufficient detail for precise cost analysis.

Piecewise Polynomials. The first attempt to evaluate the sigmoid function was taken by SecureML [125], who approximated sigmoid with a concrete piecewise polynomial:

$$\sigma(x) \approx \begin{cases} 0 & x < -\frac{1}{2} \\ x + \frac{1}{2} & -\frac{1}{2} \le x < \frac{1}{2} \\ 1 & x > \frac{1}{2} \end{cases}. \tag{1}$$

This crypto-friendly approximation requiring two comparison primitives and an evaluation of a linear function was later adopted by majority of PPML works [12], [34], [39], [40], [98], [100], [105], [124], [130], [154], [156], [170].

Fourier Series Approximation. Zheng et al. [194] follow the approach of Squirrel [118], approximating sigmoid with a degree-M Fourier series expansion on a closed interval $[-a, a]$ around 0.

$$\sigma(x) \approx \omega_0 + \sum_{i=1}^{M} \omega_i \sin \frac{2\pi i x}{2^{\lceil \log_2 a \rceil + 1}}\,.$$

They find empirically $a = 16$ and $M = 5$ suffices for high accuracy. The coefficients ω are public constants computed using an integral over a closed domain $[-a, a]$, and the computation of $\sin$ can be reduced to local computations using trigonometric identities.

LUT-based. SirNN [141] and SecFloat [140] propose evaluating sigmoid precisely with a LUT-based approach based on the construction of Dessouky et al. [59]. While more precise, these are inherently more expensive than polynomial approximations.

D.3.4. Softmax. Transformers introduce complex nonlinearities such as Softmax, GELU, and Layer Normalization (LayerNorm), whose secure evaluation incurs significant cost [78], [104], [129] due to the necessity of computing complex math functions (exponentiation, square root or division).

In attention mechanisms, Softmax performs a normalized exponential transformation:

$$\mathrm{Softmax}(x_i) = \frac{\exp(x_i - \bar{x})}{\sum_{j=1}^{d} \exp(x_j - \bar{x})}, \quad \bar{x} = \max_j x_j. \tag{2}$$

The two main challenges in secure Softmax are computing the exponential function and performing division.

MPC-Friendly Approximations. Several works replace the exponential function with low-degree, MPC-

TABLE 20: High-level protocol approach analysis to the Softmax functionality. The Max+Exp+Rec method refers to a partitioned approach for each primitive (Maximum, Exponential, Reciprocal) separately. (●) indicates use of a particular method whereas (○) indicates the method is not employed. Additional abbreviations are defined in Table 2.

N	Sec.	Protocol	Method							Tech.
			ReLU	Square	Iterative	LUT	Max+Exp+Rec	Piecewise Pol.	Exact	
2	SH	SecureML [125]	●	○	○	○	○	○	○	SS,GC
		ABY2 [130]	●	○	○	○	○	○	○	SS,GC
		DeepSecure [149]	○	○	○	○	○	○	●	GC
		Pika [169]	○	○	○	●	○	○	○	FSS
		MPCFormer [104]	○	●	○	○	○	○	○	SS,GC
		East [60]	○	○	●	○	●	○	○	SS
		Sigma [76]	○	○	○	●	●	○	○	FSS
		Iron [78]	○	○	○	●	●	○	○	FSS
		CipherGPT [86]	○	○	○	●	●	○	○	FSS
		MPCViT [186]	●	○	○	○	○	○	○	SS,GC
		Primer [193]	○	○	○	○	○	○	●	GC
		PriFFT [182]	○	○	●	○	○	○	○	SS
		SecFormer [119]	○	●	●	○	●	○	○	SS
		SirNN [141]	○	○	○	●	●	○	○	OT
		Nimbus [107]	○	○	●	○	●	●	○	SS
		Bolt [129]	○	○	○	●	●	●	○	FSS
		BumbleBee [117]	○	○	●	○	●	●	○	SS
		Zhang et al. [194]	○	○	●	○	○	○	○	SS
		SHAFT [93]	○	○	●	○	○	○	○	SS
		Shark [75]	●	○	○	●	○	○	○	FSS
3	SH	PUMA [63]	○	○	●	○	●	●	○	SS
		CryptGPU [159]	○	○	●	○	●	○	○	SS
	A	AdamInPrivate [12]	○	○	●	●	●	○	○	SS,OT
		Mosformer [43]	○	○	○	●	●	○	○	FSS
4	F	Trident [40]	●	○	○	○	○	○	○	SS,GC
		Tetrad [100]	●	○	○	○	○	○	○	SS,GC

friendly functions:

$$\mathrm{Softmax}(x_i) \approx \frac{F(x_i)}{\sum_{j=1}^{d} F(x_j)}, \tag{3}$$

$$F(x) = \begin{cases} (x+c)^2 \text{ [104], [119]} \\ \mathrm{ReLU}(x) \text{ [40], [125], [172]} . \end{cases} \tag{4}$$

These approximations reduce overhead but compromise accuracy. The division operation is often realised using a division circuit in GCs [40], [100], [125], [186], iterative methods [119] or LUTs [75]. Piranha [172] approximates the exponential with a similar non-linear function as ReLU, however, for efficiency, they compute the inverse in clear, compromising the privacy of the data.

Polynomial Approximations. Softmax exponentials can be approximated using Taylor, Maclaurin, or input-clipped exponentials:

- BOLT [129] decomposes inputs into $x_i - \bar{x} = -\ln 2 \cdot z + p$ and approximates $\exp(p)$ with a 2-degree polynomial; the division is handled via reciprocal protocols.
- PUMA [63], East [60], CipherGPT [86], and BumbleBee [117] use clipped exponentials:

$$\exp(x) \approx \begin{cases} 0 & x \leq T_{exp} \\ (1 + x/2^r)^{2^r} & T_{exp} < x \leq 0 \end{cases} \tag{5}$$

with varying choices for T_{exp} and r.

Ordinary Differential Equation. Zheng et al. [194], Shaft [93] and PriFFT [182] approximate Softmax directly using an ODE updated iteratively for t iterations. However, to converge, this method requires

$$max(\bar{x}) - min(\bar{x}) \leq t .$$

As a result t needs to be sufficiently large (in practice 128 or 256) to achieve reasonable accuracy [93]. Note that the computation and communication overhead grows linearly in t.

Lookup Table (LUT) Techniques. SirNN [141], Iron [78] and Sigma [76] precompute exponentials and reciprocals using LUTs. While fast, these incur higher communication. Sigma optimizes bit-widths to improve LUT performance. Shark [75] takes on a hybrid approach by approximating the exponential with ReLUs and computing division using optimised LUTs from LLAMA [77].

Garbled Circuits. DeepSecure [149] and Primer [193] skip the optimization step of MPC-friendly approximations and use GCs to evaluate Softmax precisely.

. Reporting exact theoretical costs for Softmax is inherently ambiguous: some works optimize a specific protocol for a fixed approximation method, whereas others propose approximation techniques that are protocol-agnostic. Accordingly, Table 20 summarizes the frameworks and their respective approaches, including only those that provide sufficient detail to enable meaningful comparison.

TABLE 21: Approximation of the GELU functionality expressed in terms of required MPC building block protocols. For LUTs, the number in brackets indicates the table size in the number of entries. Abbreviations are defined in Table 2.

N	Sec.	Protocol	Primitives				
			LUT	Multiplication	Truncation	Comparison	Multiplexer
2	SH	MPCFormer [104]	-	1	1	-	-
		SirNN [141]	6 (2^8)	7	6	5	2
		Sigma [76]	1 (2^8)	-	1	2	3
		Iron [78]	6 (2^8)	6	5	5	2
		Bolt [129]	-	3	4	2	2
		CipherGPT [86]	1 (2^6)	1	2	2	2
		Nimbus [107]	-	1	1	2	1
		SecFormer [119]	-	5	6	2	1
		BumbleBee [117]	-	4	6	3	3
		SHAFT [93]	-	2	2	2	2
3	SH	PUMA [63]	-	4	4	3	3
	A	Mosformer [43]	1 (2^8)	-	1	3	3

D.3.5. GELU. Transformers favor the GELU (Gaussian Error Linear Unit) activation:

$$\mathrm{GELU}(x) = \frac{x}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right). \tag{6}$$

This is approximated as [85]:

$$\mathrm{GELU}(x) \approx \frac{x}{2}\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right). \tag{7}$$

In the following we describe the different approximations of the GELU function, with concrete costs in terms of MPC primitives presented in Table 21. The table includes only frameworks whose protocol descriptions allow for precise cost analysis.

Quadratic Approximation. MPCFormer [104] simply take the quadratics designed approximation:

$$\mathrm{GELU}(x) \approx 0.125x^2 + 0.25x + 0.5, \tag{8}$$

which is efficient but requires knowledge distillation to recover accuracy.

Piecewise Polynomials. BumbleBee [117] and PUMA [63] identify the linearity of GELU at input extremes and apply:

$$\mathrm{GELU}(x) \approx \begin{cases} 0 & x < T_{exp} \\ P^3(x) & T_{exp} \leq x < T'_{exp} \\ P^6(x) & T'_{exp} \leq x < 3 \\ x & x > 3 \end{cases} \tag{9}$$

BOLT [129] uses a symmetric polynomial over $|x|$ and adds $0.5x$.

LUT and Fourier-Based Methods. SecFormer [119] approximates $\mathrm{erf}(x)$ via Fourier series over $[-1.7, 1.7]$. Iron [78], Mosformer [43] and Sigma [76] use LUTs to approximate $\tanh$. Sigma defines a correction term $\delta(x)$, calculated by LUT, for:

$$\mathrm{GELU}(x) \approx \mathrm{ReLU}(x) - \delta(x), \quad x \in [-4, 4]. \tag{10}$$

D.3.6. Layer Normalization. LayerNorm [14] stabilizes model activations by normalizing feature-wise statistics:

$$\mathrm{LayerNorm}(x_i) = \frac{\gamma(x_i - \mu)}{\sigma} + \beta, \tag{11}$$

where $\mu = \frac{1}{d}\sum_i x_i$, and $\sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$. While μ is easy to compute securely, σ and especially its reciprocal require square root and division approximations.

Iterative Methods. Several works [7], [12], [141], [150], [171] approximate the inverse square root with numerical methods such as Newton-Raphson or Goldschmidt's algorithm. These approximations require evaluation of linear operations based off an initial guess. SirNN [141] and SecFloat [140] employ LUT-based approach, while other alternatives involve expensive Taylor series expansion. Comet [176] proposed new plug-in method for approximating the initial guess without communication. SecureNN [170] takes iterative approach to division by a secret computing bit-by-bit long division algorithm. Falcon [171] relies on standard Newton-Raphson iterations, however, in order to achieve better performance their algorithm leaks α, such that $2^\alpha \leq x < 2^{\alpha+1}$. PrivFormer [7] builds upon the idea of Falcon and preserves privacy by proposing a protocol for computing the α in the secret-shared form using repeated comparisons.

LUT-Based. CipherGPT [86], Mosformer [43] and Sigma [76] use LUTs for reciprocal square roots. Sigma optimizes representation via 13-bit floating-point encoding.

Structural Rewrites. Iron [78] simplifies LN to $x_i - \mu$ and fuses it with the next linear layer, though BOLT [129] notes this degrades accuracy.

. Similar to the Softmax functionality, reporting exact theoretical complexities for the normalization operation is inherently ambiguous. As a result, in Table 22 we present an overview of the approaches adopted by different frameworks, focusing only on those described in sufficient detail and novelty to allow for meaningful analysis.

TABLE 22: High-level protocol approach analysis to the normalization functionality. (●) indicates use of a particular method whereas (○) indicates the method is not employed. Abbreviations are defined in Table 2.

N	Sec.	Protocol	Method				Tech.
			Iterative	LUT	Piecewise Pol.	Exact	
2	SH	DeepSecure [149]	○	○	○	●	GC
		SirNN [141]	●	○	○	○	SS
		Llama [77]	○	○	●	○	FSS
		AriaNN [150]	●	○	○	○	FSS
		Sigma [76]	○	●	○	○	FSS
		SecFormer [119]	●	○	○	○	SS
	A	Pika [169]	○	●	○	○	FSS
3	SH	PrivFormer [7]	●	○	○	○	SS
		SecureNN [170]	●	○	○	○	SS
	A	Falcon [171]	●	○	○	○	SS
		AdamInPrivate [12]	●	○	○	○	SS
		Mosformer [43]	○	●	○	○	FSS
4	SH	PrivPy [105]	●	○	○	○	SS
	F	Trident [40]	○	○	○	●	GC
		Tetrad [100]	○	○	○	●	GC

D.4. Transformers

Transformers are composed almost entirely of linear operations and simple element-wise non-linearities, making them a natural fit for the secure linear-layer protocols discussed above [187]. Here we focus on a decoder-only Transformer, as used in generative LLMs such as GPT-2 and LLaMA [165] to illustrate how linear layers work.

Attention Mechanism. Self-attention is essentially a sequence of dot-products and matrix multiplications. Given query Q, key K, and value V matrices (which are themselves outputs of linear projections of the input tokens), scaled dot-product attention computes:

$$\text{Attention}(Q,K,V) = \text{Softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)\cdot V,$$

where d_k is the dimensionality of the key vectors. In practice, Transformers use multi-head attention (MHA): the matrices Q, K, V are each split into H heads (along the feature dimension), and the attention formula is applied to each head in parallel, then concatenated. This entails computing many smaller matrix multiplies ($Q_i K_i^\top$ for each head i) and is followed by a final projection (another linear layer) to mix head outputs (see [168] for more details.).

Feed-Forward Network (FFN). After the attention sub-layer, each Transformer block has an internal feed-forward network, which is two linear transformations with an element-wise non-linearity in between. In GPT style models [138], the FFN is:

$$\text{FFN}(x) = W_2(\text{GELU}(W_1 x + b_1)) + b_2.$$

Newer models like LLaMA-2 use a gated activation (SwiGLU [153]):

$$\text{FFN}_{\text{LLaMA}}(x) = W_2((\mathrm{W_g x}\cdot\text{Sigmoid}(\mathrm{W_g x}))\odot(W_1 x)),$$

which involves three matrices W_1, W_2, W_g and a element-wise product ($\odot$) combining two paths (see [187] for more details.). The challenge with Transformers is their scale: the matrices (e.g., the projection matrices W_1, W_2 or the Q,K,V projections) are extremely large (millions of elements), and attention introduces unique data access patterns (e.g., one query dot-product with many keys). Recent works have applied and adapted MPC techniques to cope with these challenges:

In two-party settings, CipherGPT [86] specifically targets GPT-like models by using VOLE (Vector-OLE) to handle the massive matrix multiplications in attention and FFN. These approaches typically treat each Transformer layer's linear operations as large dot-products or GEMMs and apply improved OT protocols to each. LLAMA [77] introduced using a dealer to pre-share keys for all linear transformations; this means the dealer provides correlated secrets for things like the W_1, W_2 matrices and even the attention weight computation. Building on this, Orca [91], SIGMA [76], and Shark [75] show that even at the scale of tens of millions of parameters, preprocessed FSS keys allow two servers to obtain results of these linear layers essentially with one round of local computation. The downside (aside from requiring a dealer) is the storage and setup complexity of the keys. In summary, securely evaluating Transformers combines all the challenges of linear layers (massive matrix multiplications) with the need to manage state across sequential steps.

Appendix E. MPC Puzzle

This section provides additional details for the MPC Puzzle introduced in Section 5. Section E.1 presents a unified representation of secret-sharing schemes and specifies the mappings required to reconcile the different notations employed across prior works. Section E.2 outlines the remaining conversion protocols among the standard secret-sharing schemes listed in Table 7.

E.1. Unifying Secret Sharing Schemes

As discussed in Section 5, Table 7 adopts a unified notation for the secret-sharing schemes under consideration. In Table 23, we provide a correspondence between the notation used in prior works and the unified notation adopted here. The table focuses only on schemes where the notation diverges from that in Table 7. Specifically, it lists the mapping of parties, shares, and sharing semantics across these schemes.

TABLE 23: Mapping of sharing semantics from existing works to the unified semantics presented in Table 7.

Notation	Work	Parties	Shares	Semantics
$[x]^{\mathsf{a},1}$	[183]	$P_1 \to P_1$ $P_2 \to P_2$	$x^1 \to x_1 \quad m_x^1 \to \Delta_1$ $x^2 \to x_2 \quad m_x^2 \to \Delta_2$	$x = x^1 + x^2 \quad m_x = x \cdot \alpha$ $m_x = m_x^1 + m_x^2$
$[x]^{\mathsf{m}}$	[130]	$P_0 \to P_1$ $P_1 \to P_2$	$\Delta_v \to m_x \quad \delta_{v_1} \to \lambda_x^1$ $\delta_{v_2} \to \lambda_x^2$	$\Delta_v = v + \delta_v$ $\delta_v = \delta_{v_1} + \delta_{v_2}$
$[\![x]\!]^{\mathsf{a},1}$	[51]	$P_0 \to P_1$ $P_1 \to P_2$ $P_2 \to P_3$	$r \to \alpha \quad [x \cdot r]_1 \to \Delta_2$ $[x \cdot r]_0 \to \Delta_1 \quad [x \cdot r]_2 \to \Delta_3$	$[x \cdot r] = \sum_{i=0}^{2} [x \cdot r]_i$
$[\![x]\!]^{\mathsf{m}}$	[62]	$P_0 \to P_1$ $P_1 \to P_2$ $P_2 \to P_3$	$m_x \to m_x \quad \langle \psi \rangle_1 \to -\lambda_x^2$ $\langle \psi \rangle_0 \to -\lambda_x^1 \quad \langle \psi \rangle_2 \to -\lambda_x^3$	$m_x = x - \psi_x \quad \psi_x = \sum_{i=0}^{2} \langle \psi_x \rangle_i$
$[\![x]\!]^{\mathsf{m}}$	[98]	$P_0 \to P_1$ $P_1 \to P_3$ $P_2 \to P_2$	$[\alpha_v]_1 \to \lambda_x^1 \quad \beta_v + \gamma_v \to m_x$ $[\alpha_v]_2 \to \lambda_x^2 \quad \gamma_v \to \lambda_x^3$	$v = \beta_v - \alpha_v$ $\alpha_v = [\alpha_v]_1 + [\alpha_v]_2$
$[\![x]\!]^{\mathsf{m}}$	[131]	$P_0 \to P_3$ $P_1 \to P_1$ $P_2 \to P_2$	$[\alpha_v]_1 \to \lambda_x^1 \quad \beta_v + \gamma_v \to m_x$ $[\alpha_v]_2 \to \lambda_x^3 \quad \gamma_v \to \lambda_x^2$	$v = \beta_v - \alpha_v$ $\alpha_v = [\alpha_v]_1 + [\alpha_v]_2$
$[\![x]\!]^{\mathsf{m},1}$	[39]	$P_0 \to P_3$ $P_1 \to P_1$ $P_2 \to P_2$	$m_v \to m_x \quad \lambda_{v,1} \to \lambda_x^1$ $\lambda_{v,2} \to \lambda_x^2$	$v = m_v - \lambda_v$ $\lambda_v = \lambda_{v,1} + \lambda_{v,2}$
$[\![x]\!]^{\mathsf{m},2}$	[81]	$P_0 \to P_3$ $P_1 \to P_1$ $P_2 \to P_2$	$\equiv$	$\equiv$
$\langle\!\langle x \rangle\!\rangle^{\mathsf{a},1}$	[48]	$P_A \to P_1 \quad P_C \to P_3$ $P_B \to P_2 \quad P_D \to P_4$	$\equiv$	$\equiv$
$\langle\!\langle x \rangle\!\rangle^{\mathsf{r}}$	[51]	$P_0 \to P_1 \quad P_2 \to P_3$ $P_1 \to P_2 \quad P_3 \to P_4$	$x_0 \to x_4 \quad x_2 \to x_2$ $x_1 \to x_1 \quad x_3 \to x_3$	$x = \sum_{i=0}^{3} x_i$
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m},1}$	[40]	$P_0 \to P_4 \quad P_2 \to P_3$ $P_1 \to P_2 \quad P_3 \to P_1$	$m_v \to m_x \quad \lambda_{v,2} \to \lambda_x^2$ $\lambda_{v,1} \to \lambda_x^1 \quad \lambda_{v,3} \to \lambda_x^3$	$v = m_v - \lambda_v \quad \lambda_v = \sum_{i=1}^{3} \lambda_{v,i}$
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m},1}$	[81]	$P_0 \to P_1 \quad P_2 \to P_3$ $P_1 \to P_2 \quad P_3 \to P_4$	$m_x^* + \lambda_x \to m_x \quad \lambda_x^2 \to \lambda_x^2$ $\lambda_x^1 \to \lambda_x^1 \quad \lambda_x^* \to \lambda_x^3$	$x = m_x^* - \lambda_x^* \quad \lambda_x = \lambda_x^1 + \lambda_x^2$ $x = m_x - \lambda_x$
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m},1}$	[98]	$P_0 \to P_1 \quad P_2 \to P_2$ $P_1 \to P_3 \quad P_3 \to P_4$	$[\alpha_v]_1 \to \lambda_x^1 \quad \beta_v + \gamma_v \to m_x$ $[\alpha_v]_2 \to \lambda_x^2 \quad \gamma_v \to \lambda_x^3$	$v = \beta_v - \alpha_v$ $\alpha_v = [\alpha_v]_1 + [\alpha_v]_2$
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m},1}$	[100]	$P_0 \to P_4 \quad P_2 \to P_2$ $P_1 \to P_3 \quad P_3 \to P_1$	$m_v \to m_x \quad \lambda_{v,2} \to \lambda_x^2$ $\lambda_{v,1} \to \lambda_x^1 \quad \lambda_{v,3} \to \lambda_x^3$	$v = m_v - \lambda_v \quad \lambda_v = \sum_{i=1}^{3} \lambda_{v,i}$
$\langle\!\langle x \rangle\!\rangle^{\mathsf{m},2}$	[34]	$E_1 \to P_1 \quad V_1 \to P_3$ $E_2 \to P_2 \quad V_2 \to P_4$	$\mu_x^1 \to m_{x,1} \quad \sigma_x^1 \to \lambda_x^1$ $\mu_x^2 \to m_{x,2} \quad \sigma_x^2 \to \lambda_x^2$	$\mu_x = \mu_x^1 + \mu_x^2 \quad x = \mu_x - \sigma_x$ $\sigma_x = \sigma_x^1 + \sigma_x^2$

E.2. Conversion protocols

In this section, we present the remaining standard conversions, highlighting the main approaches for conversions between additive (a), replicated (r), and masked (m) secret sharing schemes.

Security. For the conversion protocols, we assume a static semi-honest adversary that corrupts parties prior to protocol execution. We distinguish between two types of conversions: (i) conversions within the same set of parties $\mathcal{P}$, and (ii) conversions across different sets $\mathcal{P} \to \mathcal{P}'$.

In the first case, the adversarial threshold is determined by the secret-sharing scheme from which the conversion originates. In the second case, we consider a single adversary with separate corruption thresholds for each party set, again dependent on the respective secret-sharing schemes.

We omit a detailed discussion of the malicious setting, but note that extending security requires leveraging the underlying mechanisms of each scheme. For example, replicated secret-sharing schemes typically exploit the fact that each share is held by more than the corruption threshold, ensuring that at least one honest party can verify correctness. Likewise, protocols designed for dishonest-majority settings are commonly augmented with techniques such as Message Authentication Codes (MACs) to guarantee correctness.

E.2.1. Local Conversions. Table 24 summarizes the standard local conversions within the same set of parties. We denote these as conversions from a source of N_S parties to a destination of N_D parties, where both belong to the same party set.

In the 2PC setting, the only local conversion is from masked to additive sharing. In the 3PC setting, additional conversions become possible, including replicated and masked to additive, as well as masked to replicated sharing within the same party set. Furthermore, 3PC admits additional local conversions where either the source or destination involves only a subset of the three parties. The 4PC setting can be viewed as a natural generalization of 3PC, with additional set of conversions, particularly in cases where the source or destination involves only two or three of the parties.

E.2.2. Conversions Across Different Party Sets. Motivated by the examples in Section 5.1, we also consider conversions between different sets of parties, i.e., from $\mathcal{P}$ to $\mathcal{P}'$. In practice, these conversions typically follow the naive approach outlined in Section 5.1.4. Our observations suggest that this general strategy is difficult to optimize further. The main challenges arise from the absence of shared Pseudo-Random Function (PRF) keys—which enable non-interactive sampling of random values—across different party sets, and from the differing corruption thresholds imposed by a single adversary. We leave the exploration of potential optimizations in this setting as an interesting direction for future work.

E.2.3. 2PC Secret-Share Conversions. We begin by considering conversions within the same party set in the 2PC setting.

$\underline{[x]^{\mathsf{a}} \to [y]^{\mathsf{m}}}$. In the preprocessing phase, each party P_i locally generates its share λ_y^i. In the online phase, the parties reconstruct $x + \lambda_y$ by exchanging $v_i = x_i + \lambda_y^i$ and computing $v_1 + v_2$. Each party P_i then sets $m_y = v_1 + v_2$ locally. This conversion requires a total communication of 2ℓ bits over one round.

$\underline{[x]^{\mathsf{m}} \to [y]^{\mathsf{a}}}$. Local (cf. Table 24).

E.2.4. 3PC Secret-Share Conversions. These conversions are discussed in Section 5.2.

E.2.5. 3PC–2PC Secret-Share Conversions. We now consider conversions between 3PC and 2PC secret-sharing schemes within the same party set. Without loss of generality, we assume that P_3 is the omitted party in 3PC-to-2PC conversions, and the added party in 2PC-to-3PC conversions. The corresponding cases for omitting or adding other parties are analogous and therefore omitted for brevity.

$\underline{[\![x]\!]^{\mathsf{a}} \to [y]^{\mathsf{a}}}$. Party P_3 generates an additive share $[\cdot]^{\mathsf{a}}$ of its value $u = x_3$ by splitting it as $u = u_1 + u_2$ and distributing u_1 to P_1 and u_2 to P_2. This requires 2ℓ bits of communication in one round, though the cost can be reduced by ℓ bits if the parties share PRF-based keys for non-interactive sampling. To complete the conversion, P_1 locally sets $y_1 = x_1 + u_1$ and P_2 sets $y_2 = x_2 + u_2$, obtaining the additive share $[y]^{\mathsf{a}}$.

$\underline{[\![x]\!]^{\mathsf{a}} \to [y]^{\mathsf{m}}}$. This conversion can be achieved by first performing $[\![x]\!]^{\mathsf{a}} \to [x']^{\mathsf{a}}$ followed by $[x']^{\mathsf{a}} \to [y]^{\mathsf{m}}$. This will result in 3ℓ bits of communication over two rounds.

$\underline{[\![x]\!]^{\mathsf{r}} \to [y]^{\mathsf{a}}}$. Local (cf. Table 24).

$\underline{[\![x]\!]^{\mathsf{r}} \to [y]^{\mathsf{m}}}$. This conversion can be achieved by first performing $[\![x]\!]^{\mathsf{r}} \to [x']^{\mathsf{a}}$ followed by $[x']^{\mathsf{a}} \to [y]^{\mathsf{m}}$. This will result in 2ℓ bits of communication over one round.

$\underline{[\![x]\!]^{\mathsf{m}} \to [y]^{\mathsf{a}}/[y]^{\mathsf{m}}}$. Local (cf. Table 24).

$\underline{[x]^{\mathsf{a}} \to [\![y]\!]^{\mathsf{a}}}$. Local (cf. Table 24).

$\underline{[x]^{\mathsf{a}} \to [\![y]\!]^{\mathsf{r}}}$. Parties P_1 and P_2 first generate additive shares of zero, $[\zeta]^{\mathsf{a}}$, using the Π_{zero} protocol. They also jointly sample a random value $r \in \mathbb{Z}_{2^\ell}$ non-interactively via the shared key setup. Then, P_1 sends $x_1 + \zeta_1 - r$ to P_3, while P_2 sends $x_2 + \zeta_2$ to P_3. Finally, the shares are set as $y_1 = x_1 + \zeta_1 - r$, $y_2 = r$, and $y_3 = x_2 + \zeta_2$. The total communication cost is 2ℓ bits in one round.

$\underline{[x]^{\mathsf{a}} \to [\![y]\!]^{\mathsf{m}}}$. In the preprocessing phase, parties non-interactively generate their λ_y shares using the shared-key setup as follows:

$$P_1, P_3 : \lambda_y^1, \quad P_1, P_2 : \lambda_y^2, \quad P_2, P_3 : \lambda_y^3.$$

During the online phase, P_1 sends $x_1 + \lambda_y^1$ to P_2, while P_2 sends $x_2 + \lambda_y^3$ to P_1. This allows both P_1 and P_2 to compute

$$m_y = x + \lambda_y^1 + \lambda_y^2 + \lambda_y^3.$$

In parallel, P_3 receives $x_1 + \lambda_y^1 + \zeta_1$ and $x_2 + \lambda_y^2 + \zeta_2$ from P_1 and P_2, respectively, where ζ_i denotes an additive share of zero generated via the Π_{zero} protocol. Using these, P_3 can also compute m_y. The total communication cost is 4ℓ bits in one round.

This conversion can be optimized to reduce communication at the cost of an additional round: P_2 sends $x_2 + \lambda_y^3$ to P_1, who then computes m_y and sends it to both P_2 and P_3. This variant requires 3ℓ bits of communication over two rounds.

$\underline{[x]^{\mathsf{m}} \rightarrow [\![y]\!]^{\mathsf{a}}}$. Local (cf. Table 24).

$\underline{[x]^{\mathsf{m}} \rightarrow [\![y]\!]^{\mathsf{r}}/[\![y]\!]^{\mathsf{m}}}$. The conversion proceeds by first applying the local conversion $[x]^{\mathsf{m}} \rightarrow [x']^{\mathsf{a}}$ from Table 24, followed by the corresponding conversion from $[x']^{\mathsf{a}}$ to $[\![y]\!]^{\mathsf{r}}$, or $[\![y]\!]^{\mathsf{m}}$ as defined earlier.

E.2.6. 4PC Secret-Share Conversions. In the following, we describe the 4PC conversions within the same set of parties.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \rightarrow \langle\!\langle y\rangle\!\rangle^{\mathsf{r}}}$. Each party P_i receives $x_{i+1} + \zeta_{i+1}$ from P_{i+1} and $x_{i+2}+\zeta_{i+2}$ from P_{i+2}, and sets its replicated share as $(y_i, y_{i+1}, y_{i+2}) = (x_i+\zeta_i,\ x_{i+1}+\zeta_{i+1},\ x_{i+2}+\zeta_{i+2})$. The total communication cost is 8ℓ bits in one round. Here, ζ_i denotes the additive share of zero generated by P_i via the Π_{zero} protocol.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \rightarrow \langle\!\langle y\rangle\!\rangle^{\mathsf{m}}}$. First, during the preprocessing phase, each party generates its λ_y^i share using the shared-key setup as follows:

$$P_1, P_3, P_4 : \lambda_y^1, \quad P_1, P_2, P_4 : \lambda_y^2$$
$$P_1, P_2, P_3 : \lambda_y^3, \quad P_2, P_3, P_4 : \lambda_y^4$$

In the online phase, each P_i receives $x_j + \lambda_y^j + \zeta_j$ from every $P_{j \neq i}$, costing 12ℓ bits sent over 1 round. Here, ζ_j denotes the additive share of zero generated for P_j through the Π_{zero} protocol. All parties then locally compute $m_y = x + \lambda_y$. Alternatively, in the online phase, P_1 can obtain $x_j + \lambda_y^j + \zeta_j$ from every $P_{j\neq 1}$ and reconstruct m_y. P_1 can then distribute m_y to all $P_{j\neq 1}$, reducing communication to 6ℓ bits at the cost of an extra round.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \rightarrow \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}}$. Local (cf. Table 24).

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \rightarrow \langle\!\langle y\rangle\!\rangle^{\mathsf{m}}}$. In the preprocessing phase, each party generates its λ_y^i share using the shared-key setup as follows:

$$P_1, P_3, P_4 : \lambda_y^1, \quad P_1, P_2, P_4 : \lambda_y^2$$
$$P_1, P_2, P_3 : \lambda_y^3, \quad P_2, P_3, P_4 : \lambda_y^4$$

In the online phase, each party P_i receives $x_{i-1} + \lambda_y^{i-1}$ from P_{i-1} and then locally computes $m_y = x + \lambda_y$. The total communication cost is 4ℓ bits in one round.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \rightarrow \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}/\langle\!\langle y\rangle\!\rangle^{\mathsf{r}}}$. Local (cf. Table 24).

E.2.7. 4PC-3PC Secret-Share Conversions. We now consider conversions between 4PC and 3PC secret-sharing schemes within the same party set. Without loss of generality, we assume that P_4 is the omitted party in 4PC-to-3PC conversions, and the added party in 3PC-to-4PC conversions. The corresponding cases for omitting or adding other parties are analogous and therefore omitted for brevity.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{a}}}$. Parties P_1 and P_4 non-interactively sample a random value $r \in \mathbb{Z}_{2^\ell}$ using the shared-key setup. Then, P_4 sends $x_4 - r$ to P_3. The shares are set as $y_1 = x_1 + r$, $y_2 = x_2$, and $y_3 = x_3 + x_4 - r$. The total communication cost is ℓ bits in one round.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{r}}}$. This conversion can be realized by first performing $\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \rightarrow [\![x']\!]^{\mathsf{a}}$ and then applying $[\![x']\!]^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{r}}$, which requires 4ℓ bits of communication over two rounds.

Alternatively, the two rounds can be merged into one at the cost of higher communication. Parties invoke the Π_{zero} protocol to generate an additive share of zero $\langle\!\langle \zeta\rangle\!\rangle^{\mathsf{a}}$. Each $P_i \in \{P_1, P_2, P_3\}$ sends $x_i + \zeta_i$ to P_{i-1} (indices taken circularly, excluding P_4), resulting in 3ℓ bits of communication. In parallel, P_4 sends $x_4 + \zeta_4$ to both P_2 and P_3. The replicated shares are then set as

$$P_1, P_3 : y_1 = x_1 + \zeta_1,$$
$$P_1, P_2 : y_2 = x_2 + \zeta_2,$$
$$P_2, P_3 : y_3 = x_3 + \zeta_3 + x_4 + \zeta_4.$$

This conversion requires 5ℓ bits of communication in a single round.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \rightarrow [\![y]\!]^{\mathsf{m}}}$. In the preprocessing phase, parties non-interactively generate their λ_y shares using the shared-key setup as follows:

$$P_1, P_3 : \lambda_y^1, \quad P_1, P_2 : \lambda_y^2, \quad P_2, P_3 : \lambda_y^3$$

In the online phase, parties invoke the Π_{zero} protocol to generate additive shares of zero $\langle\!\langle \zeta\rangle\!\rangle^{\mathsf{a}}$ and then perform a public reconstruction of $m_y = x + \lambda_y + \zeta$ among P_1, P_2, and P_3. This requires each party to receive the following values:

$$P_1 \leftarrow P_2 : x_2 + \zeta_2, \quad P_3 : x_3 + \lambda_y^3 + \zeta_3, \quad P_4 : x_4 + \zeta_4$$
$$P_2 \leftarrow P_3 : x_3 + \zeta_3, \quad P_1 : x_1 + \lambda_y^1 + \zeta_1, \quad P_4 : x_4 + \zeta_4$$
$$P_3 \leftarrow P_1 : x_1 + \zeta_1, \quad P_2 : x_2 + \lambda_y^2 + \zeta_2, \quad P_4 : x_4 + \zeta_4$$

The cost for this conversion is 9ℓ bits communicated over 1 round. Alternatively, reconstruction of m_y can instead be carried out only by P_1, who then forwards the result to P_2 and P_3. This reduces the total cost to 5ℓ bits over two rounds.

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \rightarrow [\![y]\!]^{\mathsf{a}}/[\![y]\!]^{\mathsf{r}}}$. Local (cf. Table 24).

$\underline{\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \rightarrow [\![y]\!]^{\mathsf{m}}}$. The conversion proceeds by first applying the local conversion $\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \rightarrow [\![x']\!]^{\mathsf{r}}$ from Table 24, followed by the conversion from $[\![x']\!]^{\mathsf{r}}$ to $[\![y]\!]^{\mathsf{m}}$. This conversion has a total cost of 3ℓ bits in one round.

$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to [\![y]\!]^{\mathsf{a}}/[\![y]\!]^{\mathsf{r}}/[\![y]\!]^{\mathsf{m}}$. Local (cf. Table 24).

$[\![x]\!]^{\mathsf{a}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$. Local (cf. Table 24).

$[\![x]\!]^{\mathsf{a}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{r}}$. Parties invoke the Π_{zero} protocol to generate additive shares of zero $\langle\!\langle\zeta\rangle\!\rangle^{\mathsf{a}}$. Additionally, P_2, P_3, and P_4 jointly sample random $r \in \mathbb{Z}_{2^\ell}$ using the shared-key setup. P_1 sends $x_1+\zeta_1$ to P_3 and P_4, P_2 sends $x_2+\zeta_2$ to P_1 and P_4, and P_3 sends $x_3+\zeta_3-r$ to P_1 and P_2. Finally, the replicated shares are set as

$$y_1 = x_1+\zeta_1, \quad y_2 = x_2+\zeta_2, \quad y_3 = x_3+\zeta_3-r, \quad y_4 = r.$$

The total communication cost of this conversion is 6ℓ bits in one round.

$[\![x]\!]^{\mathsf{a}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{m}}$. The conversion proceeds by first applying the local conversion $[\![x]\!]^{\mathsf{a}} \to \langle\!\langle x'\rangle\!\rangle^{\mathsf{a}}$ from Table 24, followed by the conversion from $\langle\!\langle x'\rangle\!\rangle^{\mathsf{a}}$ to $\langle\!\langle y\rangle\!\rangle^{\mathsf{m}}$. This conversion has a total cost of 12ℓ bits in one round or 6ℓ bits over two rounds.

$[\![x]\!]^{\mathsf{r}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$. Local (cf. Table 24).

$[\![x]\!]^{\mathsf{r}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{r}}$. Parties non-interactively sample random values as follows:

$$P_1, P_2, P_3 : r_1, r_2, \quad P_2, P_3, P_4 : r3$$

They then locally compute the following shares:

$$\begin{aligned} P_1, P_3 &: y_1 = x_1 + r_1 \\ P_1, P_2 &: y_2 = x_2 + r_2 \\ P_2, P_3 &: y_3 = x_3 - r_1 - r_2 - r_3 \\ P_2, P_3, P4 &: y_4 = r_3 \end{aligned}$$

To complete the sharing, P_1 sends y_1 to P_4, P_2 sends y_2 to P_4 and P_3 sends y_3 to P_1. This results in a total cost of 3ℓ bits in one round.

$[\![x]\!]^{\mathsf{r}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{m}}$. During the preprocessing phase, each party generates its λ_y^i share using the shared-key setup as follows:

$$P_1, P_3, P_4 : \lambda_y^1, \quad P_1, P_2, P_4 : \lambda_y^2$$
$$P_1, P_2, P_3 : \lambda_y^3, \quad P_2, P_3, P_4 : \lambda_y^4$$

In the online phase, parties invoke the Π_{zero} protocol to generate additive shares of zero $\langle\!\langle\zeta\rangle\!\rangle^{\mathsf{a}}$ and then perform a public reconstruction of $m_y = x + \lambda_y + \zeta$. This requires each party to receive the following values:

$$\begin{aligned} &P_1 \leftarrow P_2 : x_3 + \lambda_y^4 \\ &P_2 \leftarrow P_3 : x_1 + \lambda_y^1 \\ &P_3 \leftarrow P_1 : x_2 + \lambda_y^2 \\ &P_4 \leftarrow P_1 : x_1 + \lambda_y^1 + \zeta_1, \quad P_2 : x_2 + \lambda_y^2 + \zeta_2, \quad P_3 : x_3 + \lambda_y^3 + \zeta_3 \end{aligned}$$

The cost for this conversion is 6ℓ bits communicated over one round. Alternatively, reconstruction of m_y can instead be carried out only by P_1, who then forwards the result to P_2, P_3 and P_4. This reduces the total cost to 4ℓ bits over two rounds.

$[\![x]\!]^{\mathsf{m}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$. Local (cf. Table 24).

$[\![x]\!]^{\mathsf{m}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{r}}$. The conversion proceeds by first applying the local conversion $[\![x]\!]^{\mathsf{m}} \to [\![x']\!]^{\mathsf{r}}$ from Table 24, followed by the conversion from $[\![x']\!]^{\mathsf{r}}$ to $\langle\!\langle y\rangle\!\rangle^{\mathsf{r}}$. This conversion has a total cost of 3ℓ bits in one round.

$[\![x]\!]^{\mathsf{m}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{m}}$. The conversion proceeds by first applying the local conversion $[\![x]\!]^{\mathsf{m}} \to [\![x']\!]^{\mathsf{r}}$ from Table 24, followed by the conversion from $[\![x']\!]^{\mathsf{r}}$ to $\langle\!\langle y\rangle\!\rangle^{\mathsf{m}}$. This conversion has a total cost of 6ℓ bits in one round or 4ℓ bits over two rounds.

E.2.8. 4PC-2PC Secret-Share Conversions. Here we consider conversions between 4PC and 2PC secret-sharing schemes within the same party set. Without loss of generality, we assume that P_3 and P_4 are the omitted parties in 4PC-to-2PC conversions, and the added parties in 2PC-to-4PC conversions. The corresponding cases for omitting or adding other pair of parties are analogous and therefore omitted for brevity.

$\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \to [y]^{\mathsf{a}}$. Parties jointly sample random values non-interactively as follows:

$$P_2, P_3 : r_1, \quad P_1, P_4 : r_2.$$

Then, P_3 sends $x_3 - r_1$ to P_1, and P_4 sends $x_4 - r_2$ to P_2. The final 2PC additive shares are computed as

$$y_1 = x_1 + r_2 + (x_3 - r_1), \quad y_2 = x_2 + r_1 + (x_4 - r_2).$$

This conversion requires 2ℓ bits of communication in one round.

$\langle\!\langle x\rangle\!\rangle^{\mathsf{a}} \to [y]^{\mathsf{m}}$. During preprocessing, parties non-interactively generate their λ_y shares as

$$P_1 : \lambda_y^1, \quad P_2 : \lambda_y^2.$$

In the online phase, P_1 and P_2 receives

$$\begin{aligned} P_1 \leftarrow P_2 : x_2 + \lambda_y^2 + \zeta_2, \quad P_3 : x_3 + \zeta_3, \quad P_4 : x_4 + \zeta_4 \\ P_2 \leftarrow P_1 : x_1 + \lambda_y^1 + \zeta_1, \quad P_3 : x_3 + \zeta_3, \quad P_4 : x_4 + \zeta_4 \end{aligned}$$

Here, ζ_i denotes the additive share of zero generated via the Π_{zero} protocol. Finally, P_1 and P_2 locally compute $m_y = x + \lambda_y$. This conversion costs 6ℓ bits in one round. The communication can be reduced to 4ℓ bits over two rounds by first reconstructing m_y only at P_1, who then forwards the result to P_2.

$\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \to [y]^{\mathsf{a}}$. Local (cf. Table 24).

$\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \to [y]^{\mathsf{m}}$. The conversion proceeds by first applying the local conversion $\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \to [x']^{\mathsf{a}}$ from Table 24, followed by the conversion from $[x']^{\mathsf{a}}$ to $[y]^{\mathsf{m}}$. This conversion has a total cost of 2ℓ bits in one round.

$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to [y]^{\mathsf{a}}/[y]^{\mathsf{m}}$. Local (cf. Table 24).

$\underline{[x]^{\mathsf{a}} \rightarrow \langle\langle y \rangle\rangle^{\mathsf{a}}}$. Local (cf. Table 24).

$\underline{[x]^{\mathsf{a}} \rightarrow \langle\langle y \rangle\rangle^{\mathsf{r}}}$. Parties P_1, P_2, P_3 jointly sample $r_1 \in \mathbb{Z}_{2^\ell}$ and set $y_3 = r_1$, while P_2, P_3, P_4 jointly sample $r_2 \in \mathbb{Z}_{2^\ell}$ and set $y_4 = r_2$. P_1 and P_2 jointly samples random $r_3 \in \mathbb{Z}_{2^\ell}$. Then,

$$y_1 = x_1 - r_1 - r_3, \qquad y_2 = x_2 - r_2 + r_3.$$

P_1 sends y_1 to P_3 and P_4, and P_2 sends y_2 to P_1 and P_4.

The total communication cost of this conversion is 4ℓ bits in one round.

$\underline{[x]^{\mathsf{a}} \rightarrow \langle\langle y \rangle\rangle^{\mathsf{m}}}$. During the preprocessing phase, each party generates its λ_y^i share using the shared-key setup as follows:

$$P_1, P_3, P_4 : \lambda_y^1, \quad P_1, P_2, P_4 : \lambda_y^2$$
$$P_1, P_2, P_3 : \lambda_y^3, \quad P_2, P_3, P_4 : \lambda_y^4$$

In the online phase, parties invoke the Π_{zero} protocol to generate additive shares of zero $\langle\langle \zeta \rangle\rangle^{\mathsf{a}}$ and then perform a public reconstruction of $m_y = x + \lambda_y + \zeta$. This requires each party to receive the following values:

$$P_1 \leftarrow P_2 : x_2 + \lambda_y^4$$
$$P_2 \leftarrow P_1 : x_1 + \lambda_y^1$$
$$P_3 \leftarrow P_1 : x_1 + \lambda_y^1 + \zeta_1, \quad P_2 : x_2 + \lambda_y^2 + \zeta_2, \quad P_4 : \lambda_y^4 + \zeta_4$$
$$P_4 \leftarrow P_1 : x_1 + \lambda_y^1 + \zeta_1, \quad P_2 : x_2 + \lambda_y^2 + \zeta_2, \quad P_3 : \lambda_y^3 + \zeta_3$$

The cost for this conversion is 8ℓ bits communicated over one round. Alternatively, reconstruction of m_y can instead be carried out only by P_1, who then forwards the result to P_2, P_3 and P_4. This reduces the total cost to 4ℓ bits over two rounds.

$\underline{[x]^{\mathsf{m}} \rightarrow \langle\langle y \rangle\rangle^{\mathsf{a}}}$. Local (cf. Table 24).

$\underline{[x]^{\mathsf{m}} \rightarrow \langle\langle y \rangle\rangle^{\mathsf{r}} / \langle\langle y \rangle\rangle^{\mathsf{m}}}$. The conversion proceeds by first applying the local conversion $[x]^{\mathsf{m}} \rightarrow [x']^{\mathsf{a}}$ from Table 24, followed by the corresponding conversion from $[x']^{\mathsf{a}}$ to $\langle\langle y \rangle\rangle^{\mathsf{r}}$, or $\langle\langle y \rangle\rangle^{\mathsf{m}}$ as defined earlier.

TABLE 24: Local conversions from source N_S to destination N_D number of parties within the same set.

Setting	$N_S \to N_D$	Conversions	Parties			
			P_1	P_2	P_3	P_4
2PC	2→2	$[x]^{\mathsf{m}} \to [y]^{\mathsf{a}}$	$y_1 = m_x - \lambda_x^1$	$y_2 = -\lambda_x^2$	-	-
3PC	3→3	$[\![x]\!]^{\mathsf{r}} \to [\![y]\!]^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2$	$y_3 = x_3$	-
		$[\![x]\!]^{\mathsf{m}} \to [\![y]\!]^{\mathsf{a}}$	$y_1 = m_x - \lambda_x^1$	$y_2 = -\lambda_x^2$	$y_3 = -\lambda_x^3$	-
		$[\![x]\!]^{\mathsf{m}} \to [\![y]\!]^{\mathsf{r}}$	$y_1 = m_x - \lambda_x^1$ $y_2 = -\lambda_x^2$	$y_2 = -\lambda_x^2$ $y_3 = -\lambda_x^3$	$y_3 = -\lambda_x^3$ $y_1 = m_x - \lambda_x^1$	-
	3→2	$[\![x]\!]^{\mathsf{r}} \to [y]^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2 + x_3$	-	-
		$[\![x]\!]^{\mathsf{m}} \to [y]^{\mathsf{a}}$	$y_1 = m_x - \lambda_y^1$	$y_2 = -\lambda_x^2 - \lambda_x^3$	-	-
		$[\![x]\!]^{\mathsf{m}} \to [y]^{\mathsf{m}}$	$m_y = m_x$ $\lambda_y^1 = \lambda_x^1$	$m_y = m_x$ $\lambda_y^2 = \lambda_x^2 + \lambda_x^3$	-	-
	2→3	$[x]^{\mathsf{a}} \to [\![y]\!]^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2$	$y_3 = 0$	-
		$[x]^{\mathsf{m}} \to [\![y]\!]^{\mathsf{a}}$	$y_1 = m_x - \lambda_x^1$	$y_2 = -\lambda_x^2$	$y_3 = 0$	-
4PC	4→4	$\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2$	$y_3 = x_3$	$y_4 = x_4$
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$	$y_1 = m_x - \lambda_x^1$	$y_2 = -\lambda_x^2$	$y_3 = -\lambda_x^3$	$y_4 = -\lambda_x^4$
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{r}}$	$y_1 = m_x - \lambda_x^1$ $y_2 = -\lambda_x^2$ $y_3 = -\lambda_x^3$	$y_2 = -\lambda_x^2$ $y_3 = -\lambda_x^3$ $y_4 = -\lambda_x^4$	$y_3 = -\lambda_x^3$ $y_4 = -\lambda_x^4$ $y_1 = m_x - \lambda_x^1$	$y_4 = -\lambda_x^4$ $y_1 = m_x - \lambda_x^1$ $y_2 = -\lambda_x^2$
	4→3	$\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \to [\![y]\!]^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2$	$y_3 = x_3 + x_4$	-
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \to [\![y]\!]^{\mathsf{r}}$	$y_1 = x_1$ $y_2 = x_2$	$y_2 = x_2$ $y_3 = x_3 + x_4$	$y_3 = x_3 + x_4$ $y_1 = x_1$	-
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to [\![y]\!]^{\mathsf{a}}$	$y_1 = m_x - \lambda_y^1$	$y_2 = -\lambda_y^2$	$y_3 = -\lambda_x^3 - \lambda_x^4$	-
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to [\![y]\!]^{\mathsf{r}}$	$y_1 = m_x - \lambda_y^1$ $y_2 = -\lambda_x^2$	$y_2 = -\lambda_y^2$ $y_3 = -\lambda_x^3 - \lambda_x^4$	$y_3 = -\lambda_x^3 - \lambda_x^4$ $y_1 = m_x - \lambda_y^1$	-
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to [\![y]\!]^{\mathsf{m}}$	$m_y = m_x$ $\lambda_y^1 = \lambda_x^1$ $\lambda_y^2 = \lambda_x^2$	$m_y = m_x$ $\lambda_y^2 = \lambda_x^2$ $\lambda_y^3 = \lambda_x^3 + \lambda_x^4$	$m_y = m_x$ $\lambda_y^3 = \lambda_x^3 + \lambda_x^4$ $\lambda_y^1 = \lambda_x^1$	-
	3→4	$[\![x]\!]^{\mathsf{a}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2$	$y_3 = x_3$	$y_4 = 0$
		$[\![x]\!]^{\mathsf{r}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2$	$y_3 = x_3$	$y_4 = 0$
		$[\![x]\!]^{\mathsf{m}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$	$y_1 = m_x - \lambda_x^1$	$y_2 = -\lambda_x^2$	$y_3 = -\lambda_x^3$	$y_4 = 0$
	4→2	$\langle\!\langle x\rangle\!\rangle^{\mathsf{r}} \to [y]^{\mathsf{a}}$	$y_1 = x_1 + x_3$	$y_2 = x_2 + x_4$	-	-
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to [y]^{\mathsf{a}}$	$y_1 = m_x - \lambda_x^1 - \lambda_x^3$	$y_2 = -\lambda_x^2 - \lambda_x^4$	-	-
		$\langle\!\langle x\rangle\!\rangle^{\mathsf{m}} \to [y]^{\mathsf{m}}$	$m_y = m_x$ $\lambda_y^1 = \lambda_x^1 + \lambda_x^3$	$m_y = m_x$ $\lambda_y^2 = \lambda_x^2 + \lambda_x^4$	-	-
	2→4	$[x]^{\mathsf{a}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$	$y_1 = x_1$	$y_2 = x_2$	$y_3 = 0$	$y_4 = 0$
		$[x]^{\mathsf{m}} \to \langle\!\langle y\rangle\!\rangle^{\mathsf{a}}$	$y_1 = m_x - \lambda_x^1$	$y_2 = -\lambda_x^2$	$y_3 = 0$	$y_4 = 0$