Curl: Private LLMs through Wavelet-Encoded Look-Up Tables

Manuel B. Santos[1], Dimitris Mouris[1], Mehmet Ugurbil[1], Stanislaw Jarecki[2], José Reis[1], Shubho Sengupta[3], and Miguel de Vega[1]

[1] Nillion
{ manuel.santos, dimitris, memo, jose.reis, miguel}@nillion.com

[2] University of California, Irvine
stasio@ics.uci.edu

[3] Meta Inc.
ssengupta@meta.com

Abstract. Recent advancements in transformers have revolutionized machine learning, forming the core of Large language models (LLMs). However, integrating these systems into everyday applications raises privacy concerns as client queries are exposed to model owners. Secure multiparty computation (MPC) allows parties to evaluate machine learning applications while keeping sensitive user inputs and proprietary models private. Due to inherent MPC costs, recent works introduce model-specific optimizations that hinder widespread adoption by machine learning researchers. CrypTen (NeurIPS'21) aimed to solve this problem by exposing MPC primitives via common machine learning abstractions such as tensors and modular neural networks. Unfortunately, CrypTen and many other MPC frameworks rely on polynomial approximations of the non-linear functions, resulting in high errors and communication complexity.

This paper introduces Curl, an easy-to-use MPC framework that evaluates non-linear functions as lookup tables, resulting in better approximations and significant round and communication reduction. Curl exposes a similar programming model as CrypTen and is highly parallelizable through tensors. At its core, Curl relies on discrete wavelet transformations to reduce the lookup table size without sacrificing accuracy, which results in up to 19× round and communication reduction compared to CrypTen for non-linear functions such as logarithms and reciprocals. We evaluate Curl on a diverse set of LLMs, including BERT, GPT-2, and GPT Neo, and compare against state-of-the-art related works such as Iron (NeurIPS'22) and Bolt (S&P'24) achieving at least 1.9× less communication and latency.

Finally, we resolve a long-standing debate regarding the security of widely used probabilistic truncation protocols by proving their security in the stand-alone model. This is of independent interest as many related works rely on this truncation style.

Keywords: large language models · privacy-enhancing technologies · secure multiparty computation

1 Introduction

Large language models (LLMs) like GPT-2, GPT-4 [1], BERT [19], and LLaMA [67] have emerged as prime examples showcasing the capabilities of artificial intelligence. LLMs assist individuals and businesses in everyday tasks; from machine translation [5], to text generation [32], and question answering [57], among others. To generate human-like responses, LLMs have been trained in vast amounts of data and continue learning through their interactions with users.

However, as LLMs become increasingly integrated into human lives, privacy becomes a critical concern as individuals frequently share sensitive information, including names, addresses, and credit card numbers, or even financial and healthcare information [61]. On top of that, there is a shift towards personalized AI, with OpenAI enabling a memory feature to ChatGPT [29]. For an LLM-powered assistant to be able to understand individual preferences, habits, and workflows and provide tailored assistance, it would require access to an immense amount of personal data. As this personalized data is retained and used to continuously improve the LLMs, the possibility of a data breach or unauthorized access poses huge risks to individuals' privacy.

Privacy-enhancing technologies (PETs) such as multiparty computation (MPC) [28,76] enable a plethora of privacy-preserving machine learning (PPML) use-cases. In the most prominent setting, a server possesses a proprietary model and aims to provide it as a service to clients for use with their private data [13,46,47,38,33]. The objective is for clients to receive only the inference results without gaining any knowledge about the model, while the model provider does not learn anything about clients' input. Another popular PPML use case involves multiple distrusting parties to securely train a model over their joint sensitive data without revealing anything about their data to each other [53,66,45,74,40].

Several MPC systems leverage linear secret sharing schemes as they allow linear operations to be performed with minimal communication overhead [17,24]. However, non-linear operations (e.g., square roots, logarithms) often present greater challenges and require specialized techniques such as polynomial approximations [46], look-up table evaluations [70], or other protocol-specific approaches [8,16]. Furthermore, incorporating domain knowledge can significantly enhance the efficiency and effectiveness of these protocols by tailoring the parameters to better fit specific use cases [36,48,58].

1.1 Related Work

1.1.1 Focusing on Function Secret Sharing

Recently, several secure computation works have emerged leveraging function secret sharing (FSS) [3,70,65,56,55]. Pika [70] extends the prior work of [3] by showcasing a novel approach to securely evaluate look-up tables (LUTs) through FSS. While it demonstrates efficacy in benchmarking against popular datasets like MNIST and CIFAR-10, its scalability poses challenges. As noted by Grotto [65], for large LUTs, the computational cost may render the protocol infeasible due to the extensive number of distributed point functions (DPF) evaluations required. In contrast, Curl circumvents this challenge by eliminating the need for DPF evaluations, thanks to the integration of the discrete wavelet transform (DWT) technique. On top of that, our technique can also benefit FSS-based frameworks by reducing the size of the LUTs, resulting in less computation and communication. On the other hand, Grotto [65] introduces innovative protocols leveraging custom splines and DPFs to handle a subset of functions efficiently. Yet, it faces challenges in terms of computational and communication overhead compared to alternatives like Sigma [33]. Orca [40] showcases the potential of GPU acceleration in FSS protocols, particularly tailored for convolutional neural networks (CNNs). However, its suitability for other architectures like transformers is questioned due to its reliance on heavy non-linearities [40].

Sigma [33] builds on top of Orca and distinguishes itself by relying on minimal-sized LUTs through protocols tailored for small ranges. Despite its efficiency from custom protocols with small-sized LUTs, it demands significant computing and communication resources (e.g., 1 TB RAM and around 9Gbps communication link). Furthermore, its use of deterministic truncation, although claiming improved security, falls short in speed compared to probabilistic truncation methods. Unfortunately, all FSS-based works focus on the two-party setting as FSS becomes impractical with more than two parties.

1.1.2 Focusing on Preprocessing

The line of work initiated by Pika [70] focuses on the two-party setting with an additional dealer party. In this scenario, the preprocessing phase necessitates the dealer to prepare and distribute an $\mathcal{O}(n)$-sized element. Given the assumption that the dealer will not collude with any party, reducing dependency on such assumptions is desirable. This has led to the development of protocols that eliminate the need for a dealer. Notably, some previous works have aimed to achieve this in the two-party setting, enabling both parties to independently generate the required preprocessing material [39,18,6].

The One-Time Truth Table (OTTT) protocol [39] employs a boolean circuit to represent the table for every possible input, resulting in an exponential complexity relative to the bit size. The OP-LUT protocol [18] attempts to enhance the OTTT preprocessing phase but only achieves improvements for small bit sizes, with the communication cost remaining exponential. The SP-LUT protocol [18] significantly enhances the preprocessing phase but modifies the online phase, leading to an exponential communication cost in the

bit size during the online phase. FLUTE [6] offers advancements over previous works but still requires $\mathcal{O}(2^n)$ communication in the setup phase. Therefore, finding a preprocessing phase with sub-exponential communication complexity while maintaining the efficiency of the online phase remains an open challenge.

1.1.3 Focusing on Fully-Homomorphic Encryption

Fully homomorphic encryption (FHE) schemes have benefited significantly from the use of LUTs to enhance performance and enable new applications. This concept was initially proposed by Ducas and Micciancio [22], who devised a method to evaluate arbitrary binary gates in FHE using LUTs. Building on this foundation, Chillotti et al. [10] implemented the evaluation of arbitrary functions through a tree of leveled multiplexers, leading to the development of the Torus FHE (TFHE) scheme. Despite their development of a fast bootstrapping mechanism, capable of execution in approximately 10 milliseconds on a CPU, its adoption has been limited. This is primarily because the control inputs for the multiplexers required fresh ciphertexts – meaning prior computation on them was not possible – and the approach necessitated expressing programs as deterministic automata. Further improvement over TFHE was presented in [11] with the introduction of the programmable bootstrapping (PBS) technique, allowing for efficient and general-purpose LUT evaluation. To enhance adoptability, HELM [30] expanded on the PBS technique and introduced a framework for automatically converting Verilog hardware description language (HDL) into encrypted circuits. HELM employs three modes of operation. The first mode exclusively processes binary gates. The second mode operates on integers using secure LUT evaluations. The third, mixed mode, works with binary circuits and "bridges" over to integers for secure LUT evaluations before returning to the binary domain. However, HELM is limited to very low-precision LUTs. This limitation arises because converting an integer to multiple bits requires multiple n-to-1 LUTs. Recently, Ripple [31] proposed using compression techniques based on discrete wavelet transform (DWT) theory to decrease the size of LUT, thereby accelerating the PBS technique for general smooth functions.

1.1.4 Focusing on Secure LLM Inference

Secure frameworks for LLM inference have only recently gained traction due to the inherent complexity of transformers compared to traditional neural networks. Techniques to ensure secure inference include notable implementations like THE-X [9] and Iron [36], which were among the first to use homomorphic encryption for matrix multiplication and non-linear functions.

Subsequently, several studies have explored MPC techniques for private transformer inference system [48,21]. MPCFormer [48] leverages the CrypTen [46] MPC engine, while Puma [21] utilizes the SecretFlow-SPU [52] MPC engine to assess MPC's suitability for LLM inference. MPCFormer introduces a distillation process where stronger models train weaker models to improve accuracy, compensating for the limitations of using small (2-degree) approximations of non-linear functions. Nevertheless, MPCFormer faces challenges due to the loss of approximation accuracy, necessitating fine-tuning the models. On the other hand, Puma uses a piece-wise approximation of the GeLU activation function, requiring comparisons and polynomials up to degree 6.

Recent research has increasingly focused on hybrid solutions, combining the best techniques for specific operations [58,40]. Bolt [58] developed a solution integrating both fully homomorphic encryption and MPC. Like Puma [21], Bolt uses a piece-wise approximation that requires comparison, which increases round complexity and communication. To boost efficiency, Bolt employs word elimination while maintaining accuracy. Additionally, it enhances MPCFormer's polynomial approximations by increasing the polynomial degree and mitigates the corresponding efficiency loss through Motzkin's polynomial pre-processing procedure. In contrast, Curl avoids both comparisons and polynomial approximations by using an optimized lookup table protocol that reduces the round and communication complexity typically added by polynomial approximations while maintaining accuracy.

1.1.5 Focusing on Secure Truncation

Machine learning tasks employ floating-point numbers to construct precise models [35]. However, it is widely recognized that the use of floating-point numbers renders MPC impractical from an efficiency standpoint. For this reason, most works on machine learning under MPC employ fixed-point representation instead, requiring a truncation protocol to maintain constant precision.

Among the available truncation methodologies – deterministic, nearest integer, and probabilistic – the deterministic and probabilistic approaches (see Section 3) are the most widely used in the secure computation setting [8,49]. Moreover, a substantial body of research favors probabilistic truncation due to its lower communication costs and resultant enhanced performance, e.g., [54,53,60,13,14,47,45,64]. Despite these advantages, some recent works express concerns about probabilistic truncation, and opt for more resource-intensive protocols to achieve deterministic truncation [49,34,33]. The concerns raised with regard to probabilistic truncation include *security* and *rounding errors*.

Correctness error. Before we address the above two concerns we recall that a widely used and inexpensive probabilistic truncation protocol over fields, due to Catrina and Hoogh [8], has a correctness errors with probability $2^{-(n-|x|)}$ where $|x|$ is the maximum bitsize of truncation input x. This necessitated using significantly larger rings, setting $n \geq |x| + \kappa$ where κ is a statistical security parameter. Damgård et al. [15] presented a probabilistic protocol with zero probability of correctness error, but it required a non-constant-round bit-decomposition protocol. Subsequent works [13,25] optimized [8,15] and showed a constant-round probabilistic truncation protocol with communication and computation costs matching Catrina and Hoogh [8], but with no correctness error and allowing $n \geq |x| + 1$. CrypTen [46] proposed a protocol inspired by [71] for division by public value, which can be reduced to a probabilistic truncation with non-zero correctness error.

Security. Recently, Li et al. [49] identified a security flaw in the security proof of the probabilistic truncation of [8], revealing a discrepancy between the information leaked by that protocol and by the ideal probabilistic truncation functionality. Note that the same issue pertains to the n-optimal probabilistic truncation of [25]. This led several recent papers, e.g., [33,33], to move away from the inexpensive probabilistic truncation protocols because of concerns regarding their security. In this paper, we show that the n-optimal probabilistic truncation protocol of [25] (as well as the original probabilistic truncation of [8]) securely realizes a natural ideal probabilistic truncation functionality which we define. This enables us to capitalize on the efficiency benefits of probabilistic truncation without compromising security (Section 3).

Rounding Errors. The LLAMA paper [34] recently suggested the use of deterministic truncation, claiming it improves inference accuracy. However, this assertion has not been backed by experimental evidence, and earlier studies indicating a possible decrease in accuracy with probabilistic truncation have not been confirmed through empirical testing [64,13]. Intriguingly, Gupta et al. [35] demonstrated that computation using 16-bit fixed-point with 12-bit precision and probabilistic truncation achieves nearly equivalent accuracy to that of a 32-bit floating-point. Conversely, truncation to the nearest integer fails to train under such conditions adequately. Additionally, although not the primary focus of the work, Keller and Sun [44] provided experimental evidence indicating that in neural networks, there is minimal disparity between probabilistic truncation and truncation to the nearest integer. Indeed, out of 28 distinct experiments, 13 exhibited superior accuracy with probabilistic truncation, 13 with truncation to the nearest integer, and 2 yielded identical accuracy.

1.2 Our Contributions

In this paper, we introduce Curl, a user-friendly MPC framework designed for efficiently evaluating non-linear functions using lookup tables. Curl addresses three critical aspects essential to secure protocols: efficiency, accuracy, and security.

Curl employs a novel LUT compression technique using discrete wavelet transformation (DWT) achieving efficient approximations with minimal errors. This effectively reduces the size of original LUTs while preserving the accuracy of non-linear functions, thus surpassing traditional polynomial piece-wise approximation methods. By minimizing communication costs, Curl significantly enhances end-to-end runtime performance and supports

a wide range of activation functions including GeLU, SiLU, Gaussian error function, Sigmoid, and Hyperbolic tangent. Additionally, Curl generalizes the piece-wise approximation used within the GeLU protocol from [58,33] to encompass any bounded odd or even function that converges to a piecewise polynomial, further broadening the utility of our approach.

Curl's core technique extends to complex non-linear functions used by LLMs, ensuring both client input data privacy and safeguarding the intellectual property of service providers, such as their models. We evaluate Curl across a spectrum of LLMs including BERT (tiny, base, large), GPT-2, and GPT-Neo, including both CPU and GPU backends. Our evaluation demonstrates significant advancements over state-of-the-art frameworks, achieving at least 6.5 times fewer rounds and 1.9 times less communication overhead.

Finally, Curl addresses the security concerns surrounding efficient probabilistic truncation by introducing a natural ideal functionality, a corresponding protocol, and providing a simulation security proof. Many works that rely on this truncation style were proven to have a security flaw by [49] as they cannot be simulated in the real-ideal paradigm. Our result is of independent interest since our proof settles this debate and restores confidence in secure probabilistic truncation protocols.

We summarize our contributions as follows:

- A novel framework based on DWT compression that achieves high accuracy with low round and communication complexities. When applied to LLM inference, Curl demonstrates at least a 6.5× reduction in round complexity and a 1.9× reduction in communication when compared against existing methods.
- Curl builds on top of the user-friendly CrypTen [46] framework, offering high flexibility and low adoption friction for developers and researchers, thereby democratizing access to secure computation techniques. As a proof-of-concept, we implement a variety of LLM models and numerous non-linear functions, all operable on both CPU and GPU.
- We introduce a novel natural ideal functionality for efficient probabilistic truncation and prove Escudero et al. [25] to be secure. This result is of independent interest.

2 Preliminaries

2.1 Arithmetic & Binary Secret Sharing

We consider two main algebraic structures: the ring $\mathbb{Z}_Q$ where $Q = 2^n$ for some bit width n, and the binary field $\mathbb{Z}_2$. We consider additive secret sharing between a set of parties $\mathcal{P}$ in both $\mathbb{Z}_Q$ and $\mathbb{Z}_2$.

Arithmetic secret sharing. $[\![x]\!]$ denotes additive shares in the ring $\mathbb{Z}_Q$ for parties $\mathcal{P}_0, \ldots, \mathcal{P}_{\mathsf{N}-1}$. This means each party $\mathcal{P}_i$ owns a share $[\![x]\!]_i$, where the sum of all shares is equal to x. Private addition of secret shared values is executed by local share addition and private multiplication is implemented through Beaver triples [2]. Fixed-point representation of a floating-point number x_R is considered using the standard integer encoding with nearest-integer approximation: $x = \lfloor x_R \cdot 2^f \rceil$, for precision f [8]. To decode, we simply divide by 2^f.

Binary secret sharing. Some secure operations profit from a secret sharing representation in the binary domain. For this reason, we consider a binary secret sharing scheme for $x \in \mathbb{Z}_{2^n}$, denoted as $\langle\langle x \rangle\rangle$. A binary secret share $\langle\langle x \rangle\rangle$ is formed by secret sharing all the bits of x in $\mathbb{Z}_2$ and each party $\mathcal{P}_i$ owns a share $\langle\langle x \rangle\rangle_i$. To reconstruct the secret, the parties $\mathcal{P}_0, \ldots, \mathcal{P}_{\mathsf{N}-1}$ add their shares in $\mathbb{Z}_{2^n}$ (i.e., equivalent to bitwise XOR).

Conversions. Arithmetic shares suit best arithmetic operations ($+$, $\times$) and binary shares perform best in evaluating logical expressions (XOR, AND, right-shift and left-shift operations). For this reason, following the line of research of mixed-mode secure computation [25,46,59], we make use of known conversions between the arithmetic and the binary domain. We follow the approach provided by Damgård et al. ([16], §3) to convert from arithmetic shares to binary shares (A2B) and the procedure using algorithm 2 in [46] to convert binary shares into arithmetic shares (B2A). We denote these protocols by $\Pi^{\mathsf{A2B}}_{n,l}$ and $\Pi^{\mathsf{B2A}}_{n,l}$, respectively, where n represents the input bitwidth and l the output bitwidth.

We note that the preprocessing procedure used for B2A conversion can be optimized using the techniques proposed by Escudero et. al [25]. In particular, CrypTen's approach requires n tuples of shares $([\![r]\!], \langle\langle r \rangle\rangle)$ for a bit r, whereas [25] employs just 1 tuple of share $([\![s]\!], \langle\langle s \rangle\rangle)$ for a ring element s, with the former being comparatively more costly than the latter.

2.2 Large Language Models (LLMs)

In 2017, Vaswani et al. [69] introduced the Transformer architecture, a type of neural network that utilizes attention mechanisms, enabling AI models to prioritize different segments of the input sequence when generating an output. A transformer comprises an encoder-only architecture (like in BERT [19,75]), a decoder-only (e.g., GPT), or both encoder and decoder [69]. The encoder takes the input sequence and maps it to a lower-dimensional representation to generate a sequence of hidden states. The decoder generates an output sequence by iteratively generating tokens from the previously generated hidden states. In practice, multiple encoder and decoder blocks are used together to achieve higher accuracy.

Attention. A core component of transformers is *self-attention*, a mechanism that weighs the importance of different parts of an input sequence when making predictions. Self-attention takes a query matrix Q and a set of key-value matrix pairs K, V, which are all produced by linear layers, and produces an output as follows, where d is the dimension of the keys:

$$\mathsf{Attention}(Q, K, V) = \mathsf{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d}}\right) \cdot V.$$

Usually, the inputs to the softmax are masked so that each token is only influenced by previous tokens, thereby creating a causal relationship.

Layers. In the context of secure LLM inference, all the components of transformers can be seen as *linear* and *non-linear* layers. A linear layer consists of matrix multiplications and feed-forward neural networks (FNN) which can be evaluated using fixed-point arithmetic. A crucial part of evaluating the linear layers is to maintain the fixed-point precision using truncation.[4] For instance, the matrix multiplication $Q \cdot K^T$ in self-attention can be computed as multiple inner products, each of which needs to be followed by a truncation. Next comes the FNN which relies on an inner product followed by an activation function or a non-linear layer.

A non-linear layer, like the rectified linear unit (ReLU) [27], Gaussian error linear unit (GeLU) [37], sigmoid linear unit (SiLU), softmax, etc., is not straightforward to evaluate securely. ReLU can be implemented with a secure comparison (i.e., $\mathsf{ReLU}(x) = \mathtt{max}(0, x)$), while GeLU and other activation functions are not as trivial. Several works like CrypTen [46] resort to polynomial approximations which has a significant impact on accuracy. More recent works like Sigma [33] approximate GeLU using ReLU and then using lookup table methods to evaluate the difference. Lastly, layer normalization (`LayerNorm`) and root mean squared normalization (`RMSNorm`) require non-linear functions such as reciprocals or reciprocal square roots.

Transformer Block. A transformer block comprises several linear and non-linear layers. This usually includes attention, linear layers, non-linear activation functions, and layer normalization.

Embedding Layers. Like any language model, LLMs start off with a word embedding layer and a positional embedding layer. These layers are LUT protocols based on the token value and position of the token respectively. As LUT protocols, they can also be thought of as multiplications with a one-hot vector. To implement these under MPC, we have several options.

Firstly, even though the token value is a hidden input, its position is public. While we keep the weights hidden under MPC, with the public input we can do a plaintext lookup. With the hidden token value, however, the lookup is not so straightforward. The client either has to secret share the token as a one-hot vector, which can be quite large or we can use a random one-hot vector generated by the provider. The alternative, which is to run equality with every possible value is also quite costly. To relieve the client from sharing a one-hot vector for every token and to avoid running the costly equality multiple times, we use the provider to generate the one-hot vector. This protocol then works just like the LUT protocols, with the exception that the LUT is not public. Since the LUT, which is the embedding weights matrix, is also private, we need to rely on Beaver triple multiplication.

[4] The precision on fixed-point values doubles after multiplication so truncation is used to bring it back down.

2.3 The Discrete Wavelet Transform (DWT)

A Discrete Wavelet Transformation (DWT) is a transformation that decomposes a discretely sampled signal into approximation and detail coefficients [68]. The detail coefficients contain the information about the error incurred by the approximation coefficients. Given the approximation and detail coefficients, one can invert the application of a DWT to obtain the original signal. We convey this idea in Fig. 1. Starting with the whole signal we can apply a DWT to obtain a first level of approximation and detail coefficients. This first-level approximation and detail coefficients are enough to recover the original signal. We can repeat this process on the approximation coefficient to obtain a second level of approximation and detail coefficients. The second-level approximation and detail coefficients together with the first-level detail coefficients allow us to recover the original signal. This process can be repeated as necessary to reduce the size of the approximation coefficients.

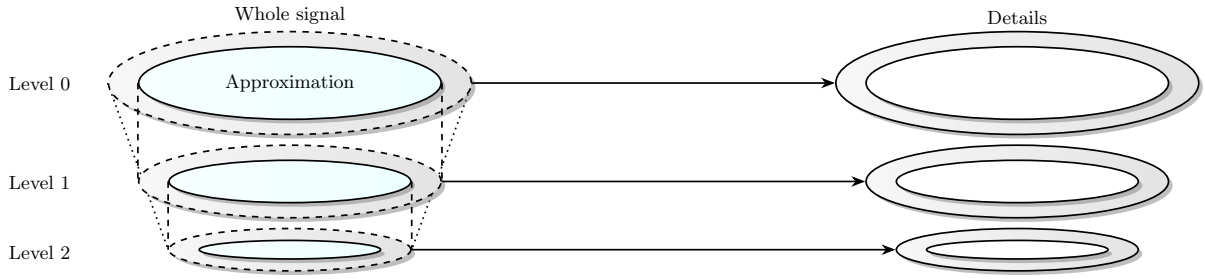


Fig. 1: Overview of two DWT iterations. The signal is represented by the approximation (blue) and the details (gray).

There are many different wavelet families and each uses linear transformations (e.g., matrix multiplication): Daubechy (Db) wavelets, Biorthogonal wavelets, and others. Let us explore in more detail two types of wavelets: the Haar and the $(5, 3)$ Biorthogonal transformations.

2.3.1 Haar DWT

A special case of the Db family is the Haar wavelet, also called Db1. In Haar, the approximation coefficients are obtained by averaging every two consecutive samples of the original signal. One of the properties of the Db wavelet of degree m (Dbm) is that if a signal encodes data of polynomials of degree strictly below m, then the corresponding detail coefficients for those polynomial parts will be zero.

More specifically, the Haar DWT is a linear transformation that can be represented by a matrix W. This linear transformation when applied to a one-dimensional signal $\boldsymbol{v}$ of length $2n$ (where $n \in \mathbb{Z}$), is defined as:

$$W \cdot \boldsymbol{v} = W \cdot \begin{bmatrix} v_0 \\ \vdots \\ v_{n-1} \\ v_n \\ \vdots \\ v_{2n-1} \end{bmatrix} = \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \\ d_0 \\ \vdots \\ d_{n-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{d} \end{bmatrix},$$

where $\boldsymbol{a}$ and $\boldsymbol{d}$ correspond to the approximation and detail coefficients respectively and are computed as:

$$a_k = \frac{v_{2k} + v_{2k+1}}{2}, \quad d_k = \frac{v_{2k} - v_{2k+1}}{2},$$

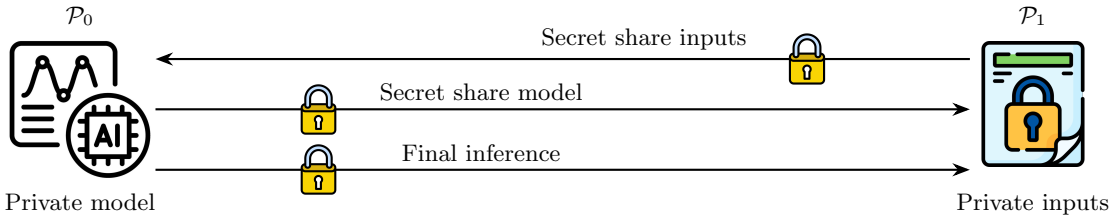


Fig. 2: Overview of Curl for privacy-preserving inference.

for $k = 0, 1, \ldots, n-1$. We note that the Haar DWT operation W can be applied multiple j times. For a vector v with size n divisible by 2^j,

$$W^j \cdot v = \begin{bmatrix} \overset{\frown}{\boldsymbol{a}}^j \\ \overset{\frown}{\boldsymbol{d}}^j \end{bmatrix},$$

where the approximation vector $\overset{\frown}{\boldsymbol{a}}^j$ has size $n/2^j$ and the detail vector $\overset{\frown}{\boldsymbol{d}}^j$ amounts to the rest (i.e., $n - n/2^j$). Note that, since W is an orthogonal matrix, its inverse is obtained by its transpose. One can see that, for $k = 0, 1, \ldots, n-1$, the inverse is given by $v_{2k+1} = a_k + d_k$ and $v_{2k} = a_k - d_k$.

2.3.2 Biorthogonal DWT

While the Dbm wavelets use an orthogonal matrix, the Biorthogonal wavelets require two different matrices where the transpose of one matrix is the inverse of the other. One matrix, B, is used for the decomposition of the signal and the other, $\widetilde{B}$, is used for the reconstruction.

In this work, we consider the $(5, 3)$ Biorthogonal DWT defined by B where the corresponding approximation and detail coefficients are given by:

$$a_k = \frac{1}{8}\left(-v_{2k-2} + 2 \cdot v_{2k-1} + 6 \cdot v_{2k} + 2 \cdot v_{2k+1} - v_{2k+2}\right) \quad d_k = \frac{1}{4}\left(v_{2k} - 2 \cdot v_{2k+1} + v_{2k+2}\right),$$

for $k = 0, \ldots, n-1$. Note that the $(5, 3)$ Biorthogonal DWT computes the approximation and detail coefficients with a weighted average of 5 and 3 points, respectively. The inverse transformation B^{-1} is given by $\widetilde{B}$ where $B^{-1} = \widetilde{B}^\top$.

2.4 Threat & Computation Model

Curl allows two or more computing parties $(\mathcal{P}_0, \mathcal{P}_1, \ldots)$ to perform private inference and training of machine-learning models. In the most prominent use case, one party (e.g., $\mathcal{P}_0$) owns a trained model, while another party (e.g., $\mathcal{P}_1$) aims to utilize the model for inference on their private data. Both parties wish to keep both the data and the model's architecture private. For instance, imagine a scenario where $\mathcal{P}_0$ is a cybersecurity firm developing a malware detection machine learning model that they offer as a service (MLaaS) to client organizations. The clients ($\mathcal{P}_1$) would like to use the model in their internal codebase while keeping sensitive implementation details private. In MLaaS, both the model owner and the clients could benefit from outsourcing the model and their inputs respectively to an MPC infrastructure and not participate in the computation.

Curl is flexible and can facilitate multiple other use cases such as: (a) having different parties that possess distinct sets of features jointly train a model without exchanging raw data, (b) unveiling correlations where one party retains feature data and another party holds corresponding labels without revealing anything else, among others. For the sake of presentation, we assume a two-party protocol and we focus on the inference setting (i.e., $\mathcal{P}_0$ holds the private model and $\mathcal{P}_1$ holds the sensitive inputs), as demonstrated in Fig. 2.

We assume the parties are *semi-honest* (i.e., follow the protocol specification) in the pre-processing model. This threat model enables a wide range of realistic use cases of secure machine learning [41,46,51,54,63], while

pre-processing is used to generate correlated randomness which the parties consume in the online phase. There is a plethora of ways to generate this randomness; one can use general-purpose MPC [77,43,42], specialized protocols [20,18,6], or more commonly a trusted dealer [3,4,34,40,46,64,65]. In Curl, we resort to the latter. Lastly, as long as not all the parties collude, no information is leaked about the private inputs (or private models depending on the use case).

3 Secure Truncation

In this work, we use truncation over the ring $\mathbb{Z}_Q$ and we define it as follows. For input $[\![x]\!]$, it outputs $[\![x']\!]$ where $x' = \lfloor x/2^f \rfloor$ and f is the precision of the fixed-point representation.

The truncation operation is required whenever we work with fixed-point numbers. Multiplication of two fixed-point rationals with precision f obtains a fixed-point rational with precision $2f$. Therefore, we require truncation to go back to a fixed-point rational with precision f. As we analyze in Section 3.3, the number of truncations required in a GPT-2 model is at least 2^{20}. Hence, it is paramount to use an efficient truncation protocol. As mentioned in Section 1.1.5, works on secure computation over fixed-point rationals use two types of truncation protocols: a deterministic truncation, where output x' is set exactly to $\lfloor x/2^f \rfloor$; or a probabilistic truncation, where $x' = \lfloor x/2f \rfloor + u$ where $u = 1$ with probability $(x \mod 2^f)/2^f$ and $u = 0$ otherwise. Although implementing probabilistic truncations incurs minimal costs, two concerns have been raised about it: security and the impact of rounding errors in ML applications. In this section, we summarize the security issue raised by Li et al. [49] and present a solution in the stand-alone secure computation framework [7,50].

3.1 Revisiting Li et al. [49]

Li et al. [49] pointed out that some efficient probabilistic truncation protocols fail to meet the criteria for standard simulation-based security [50], rendering them insecure in this framework. This includes Catrina and Hoogh's [8], CrypTen's [46], and other proposals including [15,13,25].

Li et al. use a variant of the protocol of [8] to exemplify that the above protocols are not simulatable as realizations of a standard ideal probabilistic truncation functionality, because in these protocols the rounding error u is a deterministic function of input x and an information which the protocol reveals. Specifically, given the protocol execution, the rounding error u is a deterministic function of $x_{(f)} = (x \mod 2^f)$, i.e., the last f bits of x. This violates an ideal functionality for probabilistic truncation, where $u = 1$ with probability $x_{(f)}/2^f$ (and $u = 0$ with probability $1 - x_{(f)}/2^f$), but whether or not u becomes 1 or 0 is not revealed.

We show that the probabilistic rounding protocols cited above realize a *modified* probabilistic truncation functionality, which we define in Fig. 3. The modification is natural: The functionality picks *and reveals* a "cutoff" point $s_{(f)}$ chosen at random in $\mathbb{Z}_{2^f} = [0, ..., 2^f - 1]$, and sets the rounding error as $u = 1$ if $x_{(f)} \geq s_{(f)}$, and as $u = 0$ otherwise. In Appendix A we prove that the n-optimal probabilistic truncation protocol of Escudero et al. [25], included in Fig. 9, realizes this modified truncation functionality. We note that several other probabilistic truncation protocols, e.g., [8], also realize this functionality, after adjustments related to correctness error.

Technically, we show an efficient simulator Sim s.t. for every input x and every set C of corrupted parties,

$$\left([\![\tilde{x}']\!], \tilde{s}_{(f)}, \mathsf{View}_{\mathsf{C}}([\![x]\!])\right) \approx_{\epsilon} \left([\![x]\!]', s_{(f)}, \mathsf{Sim}([\![x]\!]_{\mathsf{C}}, [\![x']\!]_{\mathsf{C}}, s_{(f)})\right)$$

where $[\![\tilde{x}']\!]$ and $\tilde{s}_{(f)}$ are the output and the cutoff point set by a protocol execution, $\mathsf{View}_{\mathsf{C}}([\![x]\!])$ is the adversarial view of that protocol execution, $[\![x']\!]$ and $s_{(f)}$ are the output and the cutoff point set by the ideal functionality, and $[\![x]\!]_{\mathsf{C}}$ and $[\![x']\!]_{\mathsf{C}}$ are the shares of resp. $[\![x]\!]$ and $[\![x']\!]$ held by parties C.

3.2 Modified Probabilistic Truncation Functionality

If x is an integer in $\mathbb{Z}_{2^n}$, we define $x_{(f)} = (x \mod 2^f)$ as the f least significant bits of x, and $x^{(f)} = \lfloor x/2^f \rfloor$ as the truncation of the last f bits of x. In particular it holds that $x = x^{(f)} \cdot 2^f + x_{(f)}$. We present the modified

probabilistic truncation functionality $\mathcal{F}_{n,f}^{\mathsf{TruncPr}}([\![x]\!])$ in Fig. 3. Note that the proposed functionality reveals the cutoff point $s_{(f)}$ whereas the functionality considered by Li et al. ([49, Functionality 2]) can be stated as following the same procedure except that it hides $s_{(f)}$. (In particular, if u is based on the random cutoff $s_{(f)}$ as in Fig. 3 then $u = 1$ with probability $x_{(f)}/2^f$, as expected.)

Note that this extra element conveys as much information as the correctness definition of the probabilistic truncation itself, which has seen wide acceptance and use in the literature. By definition, probabilistic truncation incurs a rounding error with probability $p_x = x_{(f)}/2^f$. Therefore, for a fixed input value x, one can estimate probability p_x given outputs x' from multiple runs of this functionality, and then approximate the f least significant bits of x as $x_{(f)} \approx p_x \cdot 2^f$. Each run of functionality $\mathcal{F}_{n,f}^{\mathsf{TruncPr}}([\![x]\!])$ reveals in addition the cutoff points $s_{(f)}$, which can also be used to approximate $x_{(f)}$ because each $s_{(f)}$ partitions range $[0, ..., 2^f - 1]$, and bit $u = x' - x^{(f)}$ reveals to which side of this partition $x_{(f)}$ belongs. However, the information revealed by these two versions of probabilistic truncation functionality appear equivalent. Indeed, for any $x_{(f)}$ and $x^*_{(f)}$ in $\mathbb{Z}_{2^f}$, the statistical difference between the sequence of rounding errors $u^{(1)}, ..., u^{(i)}$ "leaked" by i runs of the standard probabilistic truncation functionality on these two points is $1 - (1 - \delta)^i$ for $\delta = |x_{(f)} - x^*_{(f)}|/2^f$, which is *identical* to the difference between the corresponding views made by rounding errors and cutoff points leaked by the modified probabilistic truncation functionality $\mathcal{F}_{n,f}^{\mathsf{TruncPr}}([\![x]\!])$.

$\mathcal{F}_{n,f}^{\mathsf{TruncPr}}([\![x]\!])$

Input: Assume parties hold $[\![x]\!] \in \mathbb{Z}_{2^n}$ for $x < 2^{n-1}$.

1. Reconstruct x from sharing $[\![x]\!]$.
2. Set $x^{(f)} = \lfloor x/2^f \rfloor$ and $x_{(f)} = (x \bmod 2^f)$.
3. Pick cutoff point $s_{(f)}$ at random in $\mathbb{Z}_{2^f}$.
4. Set $x' = x^{(f)} + u$ where
$$u = \begin{cases} 0 \text{ if } x_{(f)} \leq s_{(f)}, \\ 1 \quad \text{otherwise.} \end{cases}$$
5. Generate a random sharing $[\![x']\!]$ of x'.
6. Output $[\![x']\!]$ and leak $s_{(f)}$ to the adversary.

Fig. 3: Ideal modified probabilistic truncation functionality

We prove the following theorem in Appendix A:

Theorem 1. *Protocol $\Pi_{n,f}^{\mathsf{TruncPr}}([\![x]\!])$ of Escudero et al. [25], shown in Fig. 9 in Appendix A, securely realizes functionality $\mathcal{F}_{n,f}^{\mathsf{TruncPr}}([\![x]\!])$ shown in Fig. 3 with semi-honest security in the stand-alone security framework [50].*

3.3 Comparison to CryptTen's Truncation

We chose to implement truncation using the protocol of Escudero et al. [25] instead of CrypTen's truncation [46] because of correctness and rounding errors in the latter protocol. First, as in the case of the probabilistic truncation of [8] discussed in Section 1.1.5, CrypTen's truncation incurs a correctness error with probability $2^{-(n-|x|)}$ where $|x|$ is the maximum bit size of the truncation input x.

Consider the implications of the above for LLM inference. Assuming the maximum bit size of fixed-point elements is 24 (8-bit integer part and 16-bit precision), after multiplication, the input bit size doubles. Since n is fixed at 64 bits, the correctness error occurs with a probability of $2^{-(64-48)} = 2^{-16}$. For instance, the

GPT-2 model requires at least 2^{20} inner products.[5] Assuming a truncation is executed after each inner product, we expect 16 random errors, undermining inference correctness.

We verified this experimentally by running 2^{20} CrypTen truncations of 48-bit elements 30 times, resulting in an average error of 15.83, which aligns with the expected value of 2^4. By contrast, the probabilistic truncation protocol of Escudero et al. [25] is always correct.

Furthermore, CrypTen's truncation mechanism exhibits worse rounding errors compared to functionality in Fig. 3 realized by the protocol of [25], even for $N = 2$ parties, increasing further if the number of parties increases. CrypTen's rounding error is linear in N because each party truncates their share of the input value x locally, i.e., $[x']_i = \lfloor [x]_i / 2^f \rfloor$. Since each $[x']_i$ can be 1 away from the rational value, summing N of them results in the output x' which can be up to N away from $x^{(f)}$. In contrast, value x' output by the truncation protocol of [25] is at most 1 away from $x^{(f)}$. For these reasons we opt for the truncation protocol from Escudero et al. [25] instead of CrypTen's truncation mechanism after each multiplication.

4 Curl Overview

As we mentioned in Section 2.3, DWT is used to decompose signals into approximation and detail coefficients. DWT is particularly effective for signals that represent smooth functions (e.g., logarithm, square root) since the detail coefficients are relatively small compared to the approximation coefficients. This unique observation allows us to set the detail coefficients to zero and invert the application of the DWT to obtain an approximation of the original signal. The main benefit of this process is that for smooth functions we acquire detailed approximations while requiring significantly fewer samples than the original signals. We leverage this to develop a secure protocol with compressed LUTs achieving high accuracy with low round and communication complexities.

We begin by analyzing several DWT techniques, specifically Haar and Biorthogonal DWT, for plaintext evaluation of LUTs and we translate the plaintext procedures into the secure setting. Next, we apply our DWT LUT techniques and create Curl, an easy-to-use framework for privacy-preserving machine learning that exposes a tensor-based programming model for both CPU and GPU backends. Finally, we evaluate Curl on non-linear functions and LLMs securely, improving accuracy and reducing communication compared to related works.

4.1 Improving Approximations with DWT-Encoded LUTs

Our key contribution lies in utilizing discrete wavelet transform (DWT) to securely evaluate a table. We start by explaining the secure evaluation of a look-up table (LUT) and demonstrate the application of both Haar and Biorthogonal DWT compression techniques for function evaluation using plaintext values. Next, we extend those two techniques to the secure setting. Our protocols use some techniques introduced in [12,39,70] but contrary to previous works, Curl can be easily extended to any number of parties. For simplicity, we present our protocols in the two-party setting.

4.1.1 Secure LUT Evaluation

Let $\mathsf{F} : \mathbb{R} \to \mathbb{R}$ be a function and consider its fixed-point encoded counterpart $\mathrm{T}_{\mathsf{F}} : \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n}$, with precision f. The function T_{F} can be represented as a LUT in $(\mathbb{Z}_{2^n})^{2^n}$, where $\mathrm{T}_{\mathsf{F}}(x)$ corresponds the value at the x-th index. We briefly explain the method for securely computing $\mathrm{T}_{\mathsf{F}}(\llbracket x \rrbracket) = \llbracket y \rrbracket$, which is depicted in Fig. 4.

Before initiating the protocol, both parties $\mathcal{P}_0$ and $\mathcal{P}_1$ can locally produce a LUT T_{F} representation of the function F. This evaluation needs to be performed only once and can be reused indefinitely. The secure

[5] For GPT-2, $d_{\text{model}} = 768$, $d_{ff} = 3072$, $n_{\text{heads}} = 12$, and $d_{\text{k}} = 64$. Considering an input of size $L = 64$, the token embedding requires $d_{\text{model}} \times L$ inner products, each attention mechanism requires $(d_{\text{k}} + d_{\text{model}}) \times L$, and each feed-forward layer requires at least $(d_{\text{ff}} + d_{\text{model}}) \times L$ inner products. This amounts to at least $3,637,248 > 2^{20}$ inner products.

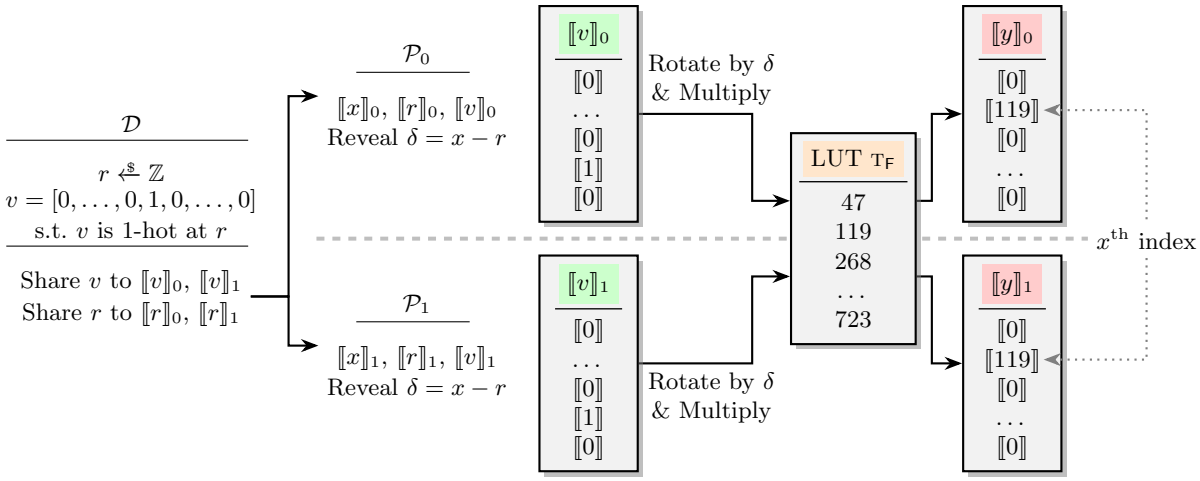


Fig. 4: Secure evaluation $\mathsf{T_F}(x) = y$ of a public LUT ($\mathsf{T_F}$) on secret-shared x ($\mathcal{P}_i$ holds $[\![x]\!]_i$) with the help of dealer $\mathcal{D}$.

evaluation of the LUT begins with the dealer generating a random element $r \in \mathbb{Z}_{2^n}$. Then, it generates a 1-hot vector v with elements equal to 0 everywhere except in the r-th entry. Then, the dealer secret-shares to both $\mathcal{P}_0$ and $\mathcal{P}_1$ the random value r along with the 1-hot vector v. These steps make up the preprocessing phase as they do not depend on the function input x. During the online phase, the parties locally compute the offset value δ given by $\delta = x - r$. This offset value is revealed to both parties $\mathcal{P}_0$ and $\mathcal{P}_1$ as it is used to locally rotate the secret-shared vector v. Note that δ cannot be revealed to the dealer, otherwise $\mathcal{D}$ would be able to recover the underlying value of the input x. After the parties rotate their shares of v by δ, the underlying value 1 is placed at $r + (x - r) = x$-th index. Finally, the parties compute the inner product between the shares of the rotated vector and the public LUT $\mathsf{T_F}$. Indeed, this yields the share of $\mathsf{T_F}$ at index x. Note that all operations are executed in the ring $\mathbb{Z}_{2^n}$.

4.2 Look-Up Table Approximation

As previously discussed, DWT provides a way to approximate signals. We observe that it is specifically well-suited for smooth functions, such as exponential, sigmoid, logarithm, and square root, to name a few. Additionally, it provides a compression mechanism that allows us to reduce bandwidth usage. Next, we delve into two approaches for evaluating a DWT-approximated LUT in the plaintext setting. This exploration will help inform the secure protocols for both Haar and $(5, 3)$ Biorthogonal DWTs.

Approach 1. Recall from Section 2.3 that when DWT is applied to a vector $\boldsymbol{v}$ j times, we end up with two vectors $\overset{\curvearrowleft j}{\boldsymbol{a}}$ and $\overset{\curvearrowleft j}{\boldsymbol{d}}$, where $\overset{\curvearrowleft j}{\boldsymbol{a}}$ is 2^j times smaller than the original vector. By setting the detail coefficients to zero, we define the approximation of $\boldsymbol{v}$ to be the approximation coefficients. This effectively reduces the size of the original vector by 2^j.

Similar to how we apply the DWT to a general vector $v \in (\mathbb{Z}_{2^n})^{2^n}$, we apply it to the LUT representation $\mathsf{T_F}$. We define $\overset{\curvearrowleft j}{\mathsf{T_F}} \in (\mathbb{Z}_{2^n})^{2^{n-j}}$ to be the LUT containing only the approximation coefficients of the original LUT after j DWT operations, $W^j \cdot \mathsf{T_F}$. Note that, in this process, the original LUT is compressed by a factor of 2^j, while the original input value x remains within $\mathbb{Z}_{2^n}$. This leads to a natural question: *how do we find the original input x in the j-times compressed table $\overset{\curvearrowleft j}{\mathsf{T_F}}$?* To locate x in $\overset{\curvearrowleft j}{\mathsf{T_F}}$, we truncate x by j bits and find the corresponding index in $\overset{\curvearrowleft j}{\mathsf{T_F}}$. This process yields the following approximation:

$$\mathsf{T_F}(x) \approx \overset{\curvearrowleft j}{\mathsf{T_F}}(x \gg j),$$

which is depicted in Fig. 5. Observe that the output bitwidth of the compressed LUT $\overset{\curvearrowleft j}{\mathrm{T}_\mathsf{F}}$ is the same as the output bitwidth of the original table T_F.

x
000
001
010
011
100
101
110
111

T_F
0000
0010
0100
0110
1000
1010
1100
1110

(a) Evaluation of the original LUT, i.e., $\mathrm{T}_\mathsf{F}(x)$.

$x \gg 1$
00
01
10
11

$\overset{\curvearrowleft 1}{\mathrm{T}_\mathsf{F}}$
0001
0101
1001
1101

(b) Approach 1 for LUT approximation. Evaluation of the Haar-compressed LUT over the truncated input, i.e., $\overset{\curvearrowleft 1}{\mathrm{T}_\mathsf{F}}$.

Fig. 5: Execution of the original and Haar-compressed LUT (T_F and $\overset{\curvearrowleft 1}{\mathrm{T}_\mathsf{F}}$) on inputs x and $x \gg 1$, respectively.

Approach 2. From DWT theory, we can retrieve the original signal from the approximation and detail coefficients by evaluating the inverse transform over them. Moreover, it is known that the inverse transform provides a method to amplify a signal, by considering the original signal as the approximation coefficients [68]. We can use this method to amplify the approximated LUT $\overset{\curvearrowleft j}{\mathrm{T}_\mathsf{F}}$ and get a better approximation than approach 1. We execute the following chain of operations:

$$\mathrm{T}_\mathsf{F} \xrightarrow{W^j} W^j \cdot \mathrm{T}_\mathsf{F} = \begin{bmatrix} \overset{\curvearrowleft j}{\mathrm{T}_\mathsf{F}} \\ \overset{\curvearrowleft j}{\boldsymbol{d}} \end{bmatrix} \longrightarrow \begin{bmatrix} \overset{\curvearrowleft j}{\mathrm{T}_\mathsf{F}} \\ \mathbf{0} \end{bmatrix} =: \overset{\curvearrowleft j}{\mathrm{E}_\mathsf{F}} \xrightarrow{W^{-j}} W^{-j} \cdot \overset{\curvearrowleft j}{\mathrm{E}_\mathsf{F}} =: \widetilde{\mathrm{T}}_\mathsf{F}, \tag{1}$$

and we obtain $\mathrm{T}_\mathsf{F}(x) \approx \widetilde{\mathrm{T}}_\mathsf{F}(x)$ for any DWT type.

Next, we explain both approaches for the Haar and $(5,3)$ Biorthogonal transformations. We discuss how the structure of the DWT matrices affects the approximation, as this provides insight into translating a plaintext approximation to a secure protocol. Finally, we extend these approaches to the secure setting and present the corresponding protocols.

4.2.1 Haar-LUT Evaluation

The two aforementioned approaches presented in Section 4.2 can be applied to the Haar DWT. Considering vectors of size 2^n, recall from Section 2.3.1 that the Haar DWT compresses data by computing the average of two elements: $a_k = (v_{2k} + v_{2k+1})/2$, for $k = 0, 1, \ldots, 2^{n-1} - 1$. In other words, as k represents the $n-1$ most significant bits (MSB) of the original index, the index elements with the same $n-1$ MSBs are averaged. Applying consecutively DWT j times, the DWT averages the index elements with the same $n-j$ MSBs.

We observe that for the Haar DWT, both approaches 1 and 2 yield the same approximation. As described in Section 4.2, the inverse Haar transformation is given by $v_{2k+1} = a_k + d_k$ and $v_{2k} = a_k - d_k$. Since the detail coefficients are set to 0, we observe that $v_{2k} = v_{2k+1} = a_k$, i.e., the elements from indexes with the same $n-1$ MSBs are reconstructed to the same value. In other words, the inverse transformation duplicates the entries of the indexes having the same $n-1$ MSBs. This yields the same result as taking the $n-1$ MSB of x and looking up the compressed table $\overset{\curvearrowleft 1}{\mathrm{T}_\mathsf{F}}$, i.e., more generally $\overset{\curvearrowleft j}{\mathrm{T}_\mathsf{F}}(x \gg j) = \widetilde{\mathrm{T}}_\mathsf{F}(x)$.

Secure Haar LUT Evaluation. The techniques described above can be used together to securely compute an approximated function F through a look-up table. The full protocol is formally described in Fig. 6.

$\Pi^{\mathsf{Haar\text{-}LUT}}_{n,l,j,\mathsf{F}}(\llbracket x \rrbracket)$

Parameters: n, l for input bit-width and output bit-width. F is the function to be computed and represented by the corresponding $j-$times compressed LUT $\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}$.

Dealer setup:

1. $\mathcal{D}$ randomly generates $r \xleftarrow{\$} \mathbb{Z}_{2^{n-j}}$.
2. $\mathcal{D}$ builds a 1-hot vector v of size 2^{n-j} with value 1 at position r.
3. $\mathcal{D}$ shares r and v to $\mathcal{P}_0$ and $\mathcal{P}_1$.

LUT evaluation for parties $\mathcal{P}_i$ for $i \in \{0,1\}$:

4. $\mathcal{P}_i$ truncates the input value $\llbracket x \rrbracket$ by j: $\llbracket x^* \rrbracket = \llbracket x \gg j \rrbracket$. ▷ *Truncation protocol [25, §5.1]*
5. $\mathcal{P}_i$ reduces its share $\llbracket x^* \rrbracket_i$ by locally computing $\llbracket x^* \rrbracket_i \mod 2^{n-j}$.
6. $\mathcal{P}_i$ compute $\llbracket \delta \rrbracket = \llbracket x^* \rrbracket - \llbracket r \rrbracket$ and reveal δ.
7. $\mathcal{P}_i$ rotate $\llbracket v \rrbracket_i$ by δ.
8. $\mathcal{P}_i$ apply the inner product between $\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}$ and the rotated vector. We denote the resulting share by $\llbracket y \rrbracket$.
9. Output: $\llbracket y \rrbracket$.

Fig. 6: Secure Haar protocol.

The secure LUT procedure of Fig. 4 can be adapted to be compatible with the DWT compression technique. Assume input bitwidth n, output bitwidth l, and the number of DWT transformations to be j. The secure Haar LUT protocol, $\Pi^{\mathsf{Haar\text{-}LUT}}_{n,l,j,\mathsf{F}}$ (Fig. 6) is given as follows. During the preprocessing phase, similar to the secure LUT evaluation, the dealer produces a secret-shared random 1-hot vector. However, to profit from DWT compression, the 1-hot vector has reduced size, i.e., 2^{n-j}, and $r \in \mathbb{Z}_{2^{n-j}}$. Following approach 1, we have that the look-up index is given by the $n-j$ MSBs of the input x. Therefore, the shift δ to be applied to the compressed LUT $\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}$ is computed using the $n-j$ MSBs of x and r. The online phase starts with the parties truncating the shares of x by j and computing its modulo in $\mathbb{Z}_{2^{n-j}}$. As we saw, the truncation from the DWT-LUT evaluation approach 1 can be executed with the Escudero et al. [25, §5.1] probabilistic truncation protocol over rings. The modulo operation is required because the shares of r belong to $\mathbb{Z}_{2^{n-j}}$ and the shares of truncated x belong to $\mathbb{Z}_{2^n}$. We note that modulo can be executed locally as described below. The rest of the protocol is similar to the online phase of the secure LUT evaluation (Fig. 4).

Local modulo. In the literature, several protocols evaluate the modulo operation over shares in a ring or field [8,26]. Those protocols are used whenever the space of the modulo operation is the same for both inputs and outputs. However, we observe that the modulo operation required for our protocol needs to reduce the ring size of the operand. More specifically, the shares $\llbracket x^* \rrbracket$ in $\mathbb{Z}_{2^n}$ have to be reduced to shares in $\mathbb{Z}_{2^{n-j}}$. This can be achieved locally by letting the parties compute the modulo operation over their shares. For correctness, consider $x = (\llbracket x \rrbracket_1 + \llbracket x \rrbracket_2) \mod 2^n$ and $x' = x \mod 2^{n-j}$. We have,

$$
\begin{aligned}
x' &= x \mod 2^{n-j} \\
&= ((\llbracket x \rrbracket_1 + \llbracket x \rrbracket_2) \mod 2^n) \mod 2^{n-j} \\
&= (\llbracket x \rrbracket_1 + \llbracket x \rrbracket_2) \mod 2^{n-j} \quad \triangleright\ 2^{n-j} \textit{ is a factor of } 2^n. \\
&= \left(\llbracket x \rrbracket_1 \mod 2^{n-j} + \llbracket x \rrbracket_2 \mod 2^{n-j}\right) \mod 2^{n-j} \\
&= (\llbracket x' \rrbracket_1 + \llbracket x' \rrbracket_2) \mod 2^{n-j}.
\end{aligned}
$$

Thus, the shares of x' can be computed locally from $\llbracket x \rrbracket$.

4.2.2 Biorthogonal-LUT Evaluation

Now, we explore the LUT DWT approximation for the $(5,3)$ Biorthogonal DWT. As described in Section 2.3.2, the approximation coefficients are given by a weighted average of 5 elements and, similar to Haar, approach 1 for the Biorthogonal compresses two indexes with the same $n-1$ MSBs to the same index. On the other hand, approach 2 requires computing $B^{-j} \cdot \overset{\curvearrowleft j}{\mathrm{E}_{\mathsf{F}}} =: \widetilde{\mathrm{T}}_{\mathsf{F}}$ as shown in Eq. (1). For simplicity, we set $j=1$ and examine the structure of the inverted Biorthogonal DWT matrix B^{-1}:

$$\begin{array}{c}\text{Index}\\ \begin{matrix}0\,0\,0\,0\\0\,0\,0\,1\\0\,0\,1\,0\\0\,0\,1\,1\end{matrix}\end{array}\begin{pmatrix}2 & 0 & 0 & \\ 1 & 1 & 0 & \\ 0 & 2 & 0 & \cdots\\ 0 & 1 & 1 & \\ \vdots & & & \ddots\end{pmatrix}\cdot\frac{1}{4}.$$

To simplify the exposition, we present matrix B^{-1} only with the elements that interact with the approximation coefficients, i.e., $\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}$. Thus, the detail coefficients are effectively set to zero. Observe that the line indexes with the same $n-1$ MSBs (green) constitute a 2×2 submatrix that combines two elements from $\overset{\curvearrowleft 1}{\mathrm{T}_{\mathsf{F}}}$, i.e., its corresponding index element and the subsequent element. For instance, the 001 -th block of $\widetilde{\mathrm{T}}_{\mathsf{F}}$, combines the 001 -th and the (001 $+1$)-th index element of $\overset{\curvearrowleft 1}{\mathrm{T}_{\mathsf{F}}}$. Note that the LSB (red) of the line index plays a role in the weights of the sub-matrices. Indeed, we have:

$$\begin{pmatrix}2 & 0\\ 1 & 1\end{pmatrix} = \begin{pmatrix}2-0 & 0\\ 2-1 & 1\end{pmatrix}.$$

Finally, consider the case $j=2$. We omit the $(1/4)^j$ constant for readability. The 2 LSBs of the indices define the elements of 4×2 sub-matrices and the $n-2$ MSBs of the indices define the elements from the approximation LUT $\overset{\curvearrowleft 2}{\mathrm{T}_{\mathsf{F}}}$ to be used.

$$\begin{array}{cccc}\text{Index} & & B^{-2} & \overset{\curvearrowleft 2}{T_{\mathsf{F}}^{\mathrm{ext}}}\end{array}\quad \widetilde{\mathrm{T}}_{\mathsf{F}}$$

$$\begin{array}{c}\begin{matrix}0\,0\,0\,0\\0\,0\,0\,1\\0\,0\,1\,0\\0\,0\,1\,1\\0\,1\,0\,0\\0\,1\,0\,1\\0\,1\,1\,0\\0\,1\,1\,1\end{matrix}\end{array}\begin{pmatrix}4 & 0 & 0 & \\ 3 & 1 & 0 & \\ 2 & 2 & 0 & \\ 1 & 3 & 0 & \\ 0 & 4 & 0 & \cdots\\ 0 & 3 & 1 & \\ 0 & 2 & 2 & \\ 0 & 1 & 3 & \\ \vdots & & & \ddots\end{pmatrix}\cdot\begin{pmatrix}a_0\\ \vdots\\ a_{2^{n-2}}\\ \hline 0\\ \vdots\\ 0\end{pmatrix} = \begin{pmatrix}(2^2-0)\cdot a_{0} + 0\cdot a_{0+1}\\ (2^2-1)\cdot a_{0} + 1\cdot a_{0+1}\\ (2^2-2)\cdot a_{0} + 2\cdot a_{0+1}\\ (2^2-3)\cdot a_{0} + 3\cdot a_{0+1}\\ \hline (2^2-0)\cdot a_{1} + 0\cdot a_{1+1}\\ (2^2-1)\cdot a_{1} + 1\cdot a_{1+1}\\ (2^2-2)\cdot a_{1} + 2\cdot a_{1+1}\\ (2^2-3)\cdot a_{1} + 3\cdot a_{1+1}\\ \vdots\end{pmatrix}$$

We can generalize the above pattern for any j, which provides an approximation formula suited for the secure setting:

$$\widetilde{\mathrm{T}}_{\mathsf{F}}(x) = \frac{1}{4^j}\left(\left(2^j - x_{(j)}\right)\cdot \overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}(x^{(j)}) + x_{(j)}\cdot \overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}(x^{(j)}+1)\right), \tag{2}$$

where $x^{(j)}$ and $x_{(j)}$ represent the $n-j$ MSBs and j LSBs of x, respectively. We recall that the $n-j$ MSBs can be obtained by truncation and the j LSB by the $\bmod 2^j$ operation.

Secure Biorthogonal LUT evaluation. Eq. (2) can be used to securely compute an approximated function F through a look-up table. The full protocol is formally described in Fig. 7. Similarly, the secure LUT

$\Pi^{\mathsf{Bior\text{-}LUT}}_{n,l,j,\mathsf{F}}([\![x]\!])$

Parameters: n, l for input bit-width and output bit-width. F is the function to be computed and represented by the corresponding $j-$times compressed LUT $\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}$.

Dealer setup:

1. $\mathcal{D}$ randomly generates $r \xleftarrow{\$} \mathbb{Z}_{2^{n-j}}$.
2. $\mathcal{D}$ builds a 1-hot vector v of size 2^{n-j} with value 1 at position r.
3. $\mathcal{D}$ shares r and v to $\mathcal{P}_0$ and $\mathcal{P}_1$.

LUT evaluation for parties $\mathcal{P}_i$ for $i \in \{0,1\}$:

4. $\mathcal{P}_i$ truncates the input value $[\![x]\!]$ by j, $[\![x \gg j]\!] := [\![x^{(j)}]\!]$. Then, $\mathcal{P}_i$ reduces its share $[\![x^{(j)}]\!]_i$ by locally computing $[\![x^{(j)}]\!]_i \mod 2^{n-j}$.
5. $\mathcal{P}_i$ computes the j LSB of x by locally computing $[\![x_{(j)}]\!] := [\![x]\!] - 2^j \cdot [\![x^{(j)}]\!]$.
6. $\mathcal{P}_i$ compute $[\![\delta]\!] = [\![x^{(j)}]\!] - [\![r]\!]$ and reveal δ.
7. $\mathcal{P}_i$ rotate $[\![v]\!]_i$ by δ and $\delta + 1$.
8. $\mathcal{P}_i$ apply the inner product between $\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}$ and the rotated vectors. We denote the resulting shares by $[\![\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}(x^{(j)})]\!]$ and $[\![\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}(x^{(j)}+1)]\!]$.
9. $\mathcal{P}_i$ computes the expression

$$[\![y]\!] := \frac{1}{4^j}\left(\left(2^j - [\![x_{(j)}]\!]\right) \cdot [\![\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}(x^{(j)})]\!] + [\![x_{(j)}]\!] \cdot [\![\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}(x^{(j)}+1)]\!]\right).$$

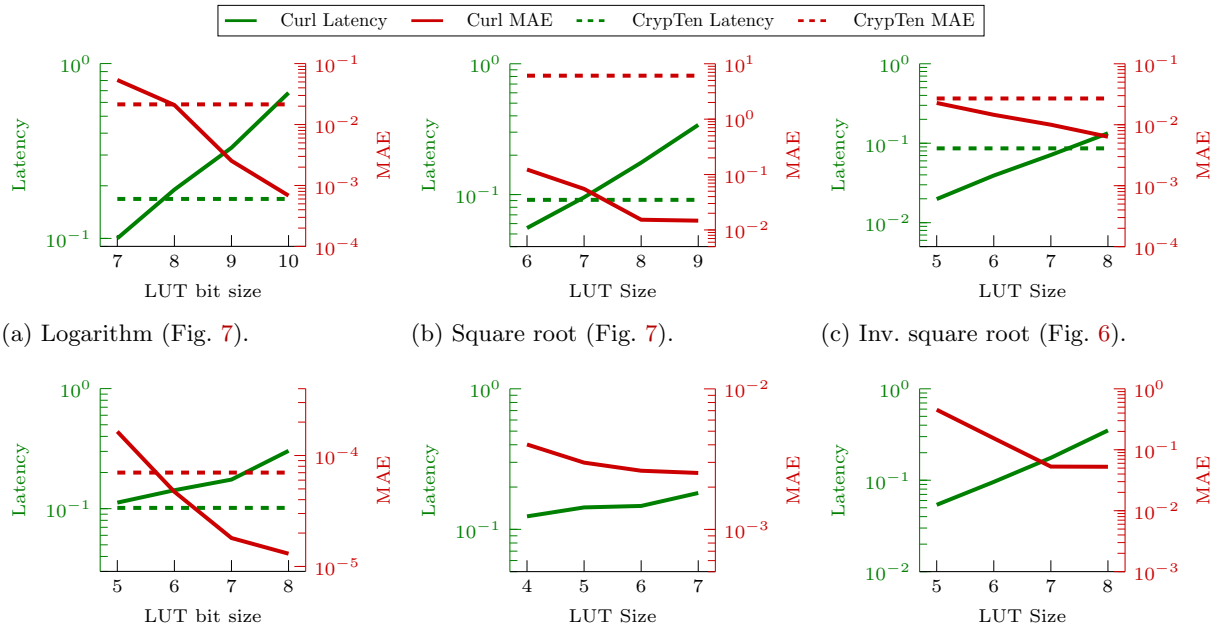
10. Output: $[\![y]\!]$.

Fig. 7: Secure Biorthogonal protocol.

procedure described before (Fig. 4) can be adapted to be compatible with the Biorthogonal DWT compression technique. Assume input bitwidth n, output bitwidth l, and the number of DWT transformations to be j.

The secure Biorthogonal LUT protocol, $\Pi^{\mathsf{Bior\text{-}LUT}}_{n,l,j,\mathsf{F}}$ (Fig. 7), is similar to $\Pi^{\mathsf{Haar\text{-}LUT}}_{n,l,j,\mathsf{F}}$ with the difference that it requires two LUT evaluations of $\overset{\curvearrowleft j}{\mathrm{T}_{\mathsf{F}}}$. As an optimization, note that these two LUT evaluations do not require two different one-hot vectors. Since the LUT evaluation is executed in two consecutive indices, we can use the same vector $\boldsymbol{v}$ to generate two shifted vectors: one shifted by δ and the other by $\delta + 1$. This allows us to use the same dealer setup phase as in secure Haar LUT evaluation. The online phase starts with the parties computing $x^{(j)}$ as the $n - j$ MSBs and $x_{(j)}$ as the j LSBs of x. More specifically, $x^{(j)}$ is computed by truncating $[\![x]\!]$ by j (i.e., [25, §5.1]) and locally computing its modulo in $\mathbb{Z}_{2^{n-j}}$, while $x_{(j)}$ is computed locally based on $x^{(j)}$ as $x = 2^j \cdot x^{(j)} + x_{(j)}$. The rest of the protocol is similar to the online phase of the secure LUT evaluation (Fig. 4) on δ and $\delta + 1$. In the final step 10, the parties compute the expression (2) as the output.

Both Haar and Biorthogonal secure evaluation protocols can be used to directly evaluate a function over a fixed interval. Furthermore, we note that our protocols can be combined with a piece-wise approach commonly employed in prior works [58,33]. We extend this method in Appendix B to accommodate any bounded even or odd function (Fig 11). We demonstrate the applicability of this protocol to various functions, including s-shape functions such as tanh, erf and sigmoid, as well as activation functions like GeLU and SiLUq. Additionally, in Appendix B.3, we introduce a protocol for both sin and cos.



(a) Logarithm (Fig. 7). (b) Square root (Fig. 7). (c) Inv. square root (Fig. 6).

(d) Sigmoid (Fig. 11). (e) SiLU (App. B.2). (f) SiLU (Fig. 7).

Fig. 8: Mean absolute error (MAE) and latency trade-offs for different DWT compressions. We plot CrypTen as a baseline.

4.3 Complexity Analysis

4.3.1 Memory

The secure DWT-LUT approach, using compressed tables, significantly minimizes the memory required for LUT handling. For instance, consider the interval $(0, 4096)$ with a precision of $f = 16$. A conventional LUT protocol would typically demand a table size of approximately 2.15 GB. However, employing the DWT technique allows us to reduce the original table size by a factor of approximately 2^{20} while maintaining high accuracy levels as discussed in Section 5.1.2. The corresponding DWT-reduced LUT occupies only 1 KB of memory, demonstrating a substantial reduction in storage requirements without compromising accuracy.

4.3.2 Round & Communication

Both Haar and Biorthogonal protocols have the same preprocessing phase and both start the online phase by truncating and opening a single field element. However, the Biorthogonal requires an additional truncation and two share multiplications (see Fig. 7 Step 9). Observe that Escudero et al. [25, §5.1] probabilistic truncation protocol over rings involves one round to open a ring element. Consequently, while the Haar protocol operates within 2 rounds and transmits 2 ring elements, the Biorthogonal is more resource-intensive, requiring 4 rounds and transmitting 5 elements.

5 Experimental Results

We implemented Curl by building on top of CrypTen and open-source it in https://github.com/jimouris/curl. Curl natively supports both CPU and GPU backends, however, we focus on CPU experiments as this is the most widely used setting in MPC. For completeness, we conduct some GPU experiments using an NVIDIA GeForce RTX 4080. All parties run in separate processes on a `c5n.9xlarge` AWS instance with 36

vCPUs and 96 GB of memory. As communication and round complexity form crucial metrics for assessing MPC end-to-end runtime, we extensively report these numbers in all our benchmarks. This can be used to glean WAN performance.

Furthermore, as Curl's novelty lies in improving approximations through DWT-encoded lookup tables, we focus on reporting the mean absolute errors (MAE) and mean relative errors (MRE) over the expected result. Finally, we focus on private LLM inference using secret-shared models on secret-shared data. Note that Curl can be used for private training in a similar way, which we omit due to space constraints. We use 16 bits of precision, following related works that typically use precision between 9 and 17 bits [54,33,72,70].

Table 1: Overview of Curl's improvement over CrypTen for selected tensors functions in terms of latency, mean absolute error (MAE), and mean relative error (MRE), for tensors of 256×256 elements.

Op.	Protocol	Domain	Curl						CrypTen [46]				
			LUT	Latency (sec.)†	Com.‡ Rounds	Com.‡ MB	Error§ MAE	Error§ MRE	Latency (sec.)	Com. Rounds	Com. MB	Error MAE	Error MRE
log	Fig. 7	$(0, 64)$	2^8	0.17	4	2.6	2.09e-2	5.48e-2	0.17	40	39.8	2.14e-2	6.36e-3
reciprocal	Fig. 7	$(1, 64)$	2^7	0.09	4	2.6	7.18e-4	1.43e-3	0.11	59	38.3	1.7e-4	7.05e-3
sqrt	Fig. 7	$(0, 256)$	2^6	0.06	4	2.6	1.23-1	1.11e-2	0.09	26	17.3	6.09e+0	4.04e-1
invsqrt	Fig. 6	$(0, 256)$	2^6	0.04	2	1.0	1.45e-2	1.14e-1	0.09	24	15.7	2.69e-2	0.405e-1
sin	App. B.3	$(-64, 64)$	2^5	0.08	16	20.4	4.55e-3	1.14e-2	0.11	37	24.1	8.52e-1	1.58e+0
cos	App. B.3	$(-64, 64)$	2^5	0.08	16	20.4	4.77e-3	9.85e-2	0.10	37	24.1	8.86e-1	1.45e+0
sigmoid	Fig. 11	$(-64, 64)$	2^6	0.10	22	33.6	4.70e-5	7.83e-2	0.11	26	26.2	7.00e-5	3.49e+0
	Fig. 7	$(-64, 64)$	2^6	0.10	4	2.6	1.11e-2	6.59e-2	0.11	26	26.2	7.00e-5	3.49e+0
tanh	Fig. 11	$(-64, 64)$	2^5	0.09	22	33.6	2.31e-4	3.96e-4	0.13	26	26.2	8.60e-5	1.19e-4
erf	Fig. 11	$(-64, 64)$	2^3	0.09	22	33.6	8.98e-4	1.83e-3	0.21	56	36.2	3.39e+7	3.40e+7
GeLU	App. B.2	$(-64, 64)$	2^4	0.10	30	47.7	5.95e-3	2.79e+0	N/A	N/A	N/A	N/A	N/A
	Fig. 7	$(-4, 4)$	2^4	0.11	4	2.6	2.60e-3	5.02e-2	N/A	N/A	N/A	N/A	N/A
SiLU	App. B.2	$(-64, 64)$	2^6	0.14	30	47.7	2.61e-3	5.48e-3	N/A	N/A	N/A	N/A	N/A
	Fig. 7	$(-64, 64)$	2^6	0.09	4	2.6	1.54e-1	1.18e-1	N/A	N/A	N/A	N/A	N/A

† Green background indicates the fastest runtime, ‡ orange indicates the lowest communication, and § blue indicates the lowest error (MAE or MRE).

5.1 Non-linear Functions

We experimentally verified that the Biorthogonal protocol achieves better approximations than Haar with almost identical latency and communication costs (Appendix C). Therefore, we use the Biorthogonal protocol for almost all our experiments. We choose constrained ranges of input values to enable a meaningful comparison with CrypTen. For wider ranges, such as $(0, 2^{12})$, CrypTen's approximations exhibit significantly higher errors. Since previous MPC works [58,48,21] rely on similar polynomial approximations, we can argue that they all have similar error levels. For example, for the log function, CrypTen's approximation yields a MAE of 1.09e+10, whereas the Biorthogonal with a compression $j = 21$ achieves a substantially lower MAE of 2.66e-1. We note that the latency time reported is dominated by computation time as the protocol is executed in the same machine.

5.1.1 Choosing LUT Sizes

We evaluate our secure DWT protocols to compute a set of smooth functions commonly used in machine learning tasks such as logarithm (used for perplexity and cross-entropy that measure LLM's performance), square root and inverse square root (used for layer normalization), sigmoid, and SiLU. In Fig. 8 we investigate the trade-offs between latency on the left y-axes (green) and MAE on the right y-axes (red) for the

aforementioned non-linear functions. As a baseline, we show CrypTen's latency and MAE with dashed green and red lines, respectively. In the case of SiLU, which we evaluate with two different protocols, CrypTen does not support this so we only show Curl's latency and MAE. Of course, bigger LUTs in Curl achieve better approximations but require more computation and thus higher latency. However, DWT reduces the size of the LUTs which therefore decreases the computation complexity of Curl while still achieving good approximations. For a LAN setting, the total runtime is expected to be dominated by the computation time. However, for WAN, the total runtime is mostly dominated by communication. Therefore, we focus on achieving better errors than CrypTen while keeping a similar latency level. As an example, consider running 265×265 log functions in a LAN setting with 0.8ms ping-time and 3 Gbps bandwidth and in a WAN setting with 80ms ping-time and 200 Mbps bandwidth. We estimate[6] Curl to take 180ms and CrypTen to take 313ms ($\times1.8$ more) in LAN and Curl to take 599ms and CrypTen to take $5,039$ms ($\times10.1$ more) in WAN.

For general and unbounded functions, we apply the secure Biorthogonal protocol (Fig. 7). Exceptionally, we use Haar-encoded protocol (Fig. 6) for the invsqrt function as it achieves better errors whenever the $(0, 1)$ interval is included. For sqrt and invsqrt we consider the interval $(0, 256)$. However, for log we consider the interval to be $(0, 64)$ as CrypTen's MAE and MRE are of the order of 1e+6 for $(0, 256)$. This corresponds to an original LUT of size 2^{24} for sqrt and invsqrt and of size 2^{22} for log. Observe the logarithm approximation in Fig. 8a in which both latency and MAE of Curl matches that of CrypTen's for LUT sizes of 8 bits. In Figs. 8b and 8c, Curl's MAE is always below CrypTen's. For these two functions, we use LUT sizes with 6 bits as these provide lower latency for Curl. We set the compression parameter to $j = 18$ for both sqrt and invsqrt and 14 for log. For sigmoid, we use the protocol presented in Fig. 11 and set the interval to be $(-64, 64)$. As analyzed in Appendix B.1, sigmoid is not an odd function but the last step of the protocol can be adapted to be applied to any s-shape function, including sigmoid. We observe that Curl achieves a better approximation than CrypTen for LUT sizes with 6 or more bits with runtime being marginally slower. Finally, for SiLU, we set the interval to be $(-64, 64)$ and we use two different protocols: the approach described in Appendix B.2 which is a derivation of Fig 11 protocol (Fig. 8e); and the Biorthogonal protocol (Fig. 8f). We observe that Appendix B.2 approach achieves better MAE errors when compared to the Biorthogonal protocol. The latency from Fig. 8e is less affected by an increase in the LUT when compared with Fig. 8f. The same behavior happens with GeLU and SiLU as we can observe in Table 1. This is justified by the fact that the approach from Appendix B.2 requires comparisons, whether the Biorthogonal protocol does not. Indeed, note that in all functions where a DWT-LUT protocol is applied, the latency decreases by half when we half the LUT size.

5.1.2 Evaluations

Having found the optimal tradeoffs for LUT sizes, we compare the Curl approximations with the DWT technique to CrypTen's polynomial approximations in terms of latency, communication, and errors. In Table 1, we evaluate a set of commonly used non-linear functions including logarithm, inverse square root, the sigmoid activation function, as well as GeLU and SiLU. For unbounded functions, we use the protocol from Fig. 7, while for bounded functions (e.g., sin) we use our optimized protocols which are referenced in the table. We experimentally verified that, in most cases, the Biorthogonal protocol outperforms the Haar DWT in terms of both MAE and MRE, with the only exception being invsqrt. Observe that for the same LUT size, the latency for Haar is 2/3 lower than the Biorthogonal approach (compare sqrt 0.06s latency using Biorthogonal with invsqrt 0.04s latency using Haar). For each function, we specify the evaluation domain for which we report the errors (i.e., MAE and MRE).

Observe that all functions that utilize the protocol from Fig. 7 and Fig. 6 have constant rounds and communication, regardless of the LUT size, as the one-hot vectors (which depend on the size) are generated during preprocessing from the dealer. Latency is affected by the LUT size (e.g., sqrt vs log) since the LUT output value is accumulated with shares of zero, which are as many as the LUT size (see last step of Fig. 4). However, as discussed before, the end-to-end latency will be dominated by the number of rounds and total

[6] We add the computation latency, the ping-time multiplied by the number of rounds and the amount of information sent divided by the bandwidth. We use the numbers reported in Table 1.

communication size, for which Curl is almost always significantly better. Regarding trigonometric functions (sin and cos) we use the approach proposed in Appendix B.3. We note that CrypTen's approximation of both functions already has a 1.2e+2 error for the range $(-128, 128)$. For this reason, we decrease the interval range to $(-64, 64)$. On the other hand, Curl's approximation keeps the same level of errors independent of the input range. Moreover, Curl achieves that while having less latency, rounds and communication when compared with CrypTen.

5.2 LLM Inference

We evaluate Curl with BERT Tiny (4 million parameters), BERT Base (110 million parameters), BERT Large (340 million parameters), GPT-2 (124 million parameters), and GPT Neo (1.3 billion parameters). In Table 2, we report the latency, number of rounds, and total communication for each LLM for 64 inputs. For completeness, we also evaluate BERT Tiny and Base on GPU for 64 elements and report between $2-3\times$ speedup compared to CPU (1.1 and 6.5 seconds, respectively). Finally, we use the QNLI classification task from the GLUE benchmark [73] to evaluate the accuracy of our models. Curl achieves an accuracy of 80.3% for encrypted BERT Tiny, which almost matches the plaintext accuracy (i.e., 81.4%). This further verifies our claim regarding the probabilistic truncation protocol that is sufficiently accurate to evaluate LLMs. However, we observed that in LLMs the LUT protocols are not great for inverse square root as the input range varies and the output of the function changes rapidly in $(0, 1]$. Thus, to achieve high accuracy we resort to a comparison and two LUTs: a dense one between $(0, 1]$ and a sparse one in $(1, 256)$, getting the best of both worlds.

Table 2: Runtime and communication of Curl with a sequence length of 64 items.

Model	Latency (s)	Rounds	Com. (GB)
BERT Tiny	3.55	409	1.34
BERT Base	13.63	1,629	2.8
BERT Large	33.93	3,093	5.66
GPT-2	16.61	1,630	3.77
GPT-Neo	103.4	3,118	14.9

Next, we compare with Iron, MPCFormer, Puma, and Bolt in Table 3 for BERT Base with 128 items. In terms of end-to-end latency, Curl outperforms all other frameworks by at least $1.5\times$, with as much as $20\times$ in the case of Iron. More importantly, Curl achieves this with significantly less amount of rounds and total communication. Specifically, Iron and Bolt require $8\times$ and $6\times$ more rounds than Curl, respectively, while in terms of total communication, all other frameworks need more than 10 GBs (with Iron needing 281 GBs) while Curl only needs 5.7 GBs. Although the end-to-end runtime in a real-world instantiation would increase due to communication, Curl would not be affected as much by that since it exhibits the lowest communication and number of rounds.

Finally, in Table 4 we compare Curl with CrypTen for five LLMs with 64 inputs. This differs from Tables 2 and 3 as CrypTen lacks a protocol to evaluate embeddings (Section 2.2). Thus, in Table 4, we skip embedding for a fair comparison. Curl needs less than half the amount of rounds CrypTen does for all five models and almost half the total communication. Although Curl's runtimes are already faster than CrypTen's, the difference in end-to-end latency will increase further in a realistic instantiation due to communication.

6 Concluding Remarks

We have introduced Curl, a framework for privacy-preserving machine learning that evaluates non-linear functions as lookup tables, resulting in better approximations and significant round and communication

Table 3: Runtime and communication comparisons with a sequence length of 128 items for the full BERT Base LLM.

Framework	Latency (s)	Rounds	Com. (GB)
Iron [36]	475	13,663	281
MPCFormer [48]	55.3	–	12.1
Puma [21]	33.9	–	10.8
Bolt [58]	185	10,509	59.6
Bolt (WE) [58]†	91	10,901	25.7
Curl	22.5	1,629	5.7

† In Bolt, WE stands for word elimination.

Table 4: Runtime and communication comparisons with CrypTen with a sequence length of 64 items. These models skip the embeddings as CrypTen does not support lookups.

Model	Curl			CrypTen [46]		
	Lat. (s)	Rounds	Com. (GB)	Lat. (s)	Rounds	Com. (GB)
BERT Tiny	0.51	252	0.02	0.86	539	0.05
BERT Base	9.01	1,472	1.31	13.23	3,089	2.62
BERT Large	28.79	2,936	4.11	38.59	6,149	7.6
GPT-2	8.94	1,464	1.31	12.93	3,060	2.62
GPT-Neo	98.44	2,952	11.95	137.20†	6,144	18.91

† Approximated since CrypTen could not fit GPT Neo in RAM.

reduction. Curl achieves this by relying on discrete wavelet transformations to reduce the lookup table sizes without sacrificing accuracy, which has resulted in up to 19× round and communication reduction compared to related works. We have evaluated Curl on five large language models such as BERT, GPT-2, and GPT Neo (1.3 billion parameters), and have compared it against state-of-the-art related works, significantly reducing latency, the number of communication rounds, and the total communication. On top of that, Curl is easy to use by exposing a tensor-based programming model that machine learning researchers and developers are familiar with which allows for both CPU and GPU backends. Lastly, we have introduced a new ideal functionality for probabilistic truncation protocols and proved their security in the real-ideal paradigm. This is of independent interest, as truncation is a core component of encrypted machine-learning models, and probabilistic truncation was previously considered insecure.

References

1. Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
2. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.
3. 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 Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 871–900, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.
4. Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume

11891 of *Lecture Notes in Computer Science*, pages 341–371, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

5. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
6. Andreas Brüggemann, Robin Hundt, Thomas Schneider, Ajith Suresh, and Hossein Yalame. FLUTE: Fast and secure lookup table evaluations. In *2023 IEEE Symposium on Security and Privacy*, pages 515–533, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.
7. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13:143–202, 2000.
8. Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199, Amalfi, Italy, September 13–15, 2010. Springer, Heidelberg, Germany.
9. Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. THE-X: Privacy-preserving transformer inference with homomorphic encryption. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3510–3520, Dublin, Ireland, May 2022. Association for Computational Linguistics.
10. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.
11. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning*, pages 1–19, Cham, 2021. Springer International Publishing.
12. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press.
13. Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, October 2020.
14. Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 2183–2200. USENIX Association, August 11–13, 2021.
15. 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 *2019 IEEE Symposium on Security and Privacy*, pages 1102–1120, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.
16. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.
17. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.
18. 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 *ISOC Network and Distributed System Security Symposium – NDSS 2017*, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society.
19. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
20. Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 523–535, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

21. Ye Dong, Wen jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Cheng. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*, 2023.
22. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.
23. Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, November 2018.
24. Daniel Escudero. An introduction to secret-sharing-based secure multiparty computation. Cryptology ePrint Archive, Report 2022/062, 2022. https://eprint.iacr.org/2022/062.
25. Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 823–852, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.
26. Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. Cryptology ePrint Archive, Report 2020/338, 2020. https://eprint.iacr.org/2020/338.
27. Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3):121–136, 1975.
28. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.
29. Dongyu Gong, Xingchen Wan, and Dingmin Wang. Working memory capacity of ChatGPT: An empirical study. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10048–10056, 2024.
30. Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. HELM: Navigating Homomorphic Encryption through Gates and Lookup Tables. Cryptology ePrint Archive, Paper 2023/1382, 2023. https://eprint.iacr.org/2023/1382.
31. Charles Gouert, Mehmet Ugurbil, Dimitris Mouris, Miguel de Vega, and Nektarios Georgios Tsoutsos. Ripple: Enhancing Homomorphic Lookup Tables with Wavelets, 2024.
32. Tanya Goyal, Junyi Jessy Li, and Greg Durrett. News summarization and evaluation in the era of GPT-3. *arXiv preprint arXiv:2209.12356*, 2022.
33. Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT Inference with Function Secret Sharing. Cryptology ePrint Archive, Report 2023/1269, 2023. https://eprint.iacr.org/2023/1269.
34. Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. LLAMA: A low latency math library for secure inference. *Proceedings on Privacy Enhancing Technologies*, 2022(4):274–294, October 2022.
35. Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1737–1746. JMLR.org, 2015.
36. Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 15718–15731. Curran Associates, Inc., 2022.
37. Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.
38. Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 809–826, Boston, MA, USA, August 10–12, 2022. USENIX Association.
39. Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 600–620, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.
40. Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: FSS-based secure training with GPUs. Cryptology ePrint Archive, Report 2023/206, 2023. https://eprint.iacr.org/2023/206.
41. Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 1651–1669, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

42. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1575–1590, Virtual Event, USA, November 9–13, 2020. ACM Press.
43. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.
44. Marcel Keller and Ke Sun. Effectiveness of mpc-friendly softmax replacement, 2020.
45. Marcel Keller and Ke Sun. Secure quantized training for deep learning. In *International Conference on Machine Learning*, pages 10912–10938. PMLR, 2022.
46. Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021.
47. Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: Super-fast and robust privacy-preserving machine learning. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 2651–2668. USENIX Association, August 11–13, 2021.
48. Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. MPCFormer: Fast, performant and private transformer inference with MPC. In *International Conference on Learning Representations (ICLR)*, 2023.
49. 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 *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5377–5394, Anaheim, CA, August 2023. USENIX Association.
50. Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. https://eprint.iacr.org/2016/046.
51. Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 619–631, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.
52. Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 17–33, Boston, MA, July 2023. USENIX Association.
53. Payman Mohassel and Peter Rindal. ABY^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 35–52, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
54. Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.
55. Dimitris Mouris, Christopher Patton, Hannah Davis, Pratik Sarkar, and Nektarios Georgios Tsoutsos. Mastic: Private Weighted Heavy-Hitters and Attribute-Based Metrics. Cryptology ePrint Archive, Report 2024/221, 2024. https://eprint.iacr.org/2024/221.
56. Dimitris Mouris, Pratik Sarkar, and Nektarios Georgios Tsoutsos. PLASMA: Private, Lightweight Aggregated Statistics against Malicious Adversaries. *Proceedings on Privacy Enhancing Technologies*, 2024(3):1–19, July 2024.
57. Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
58. Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 133–133, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.
59. Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. Cryptology ePrint Archive, Report 2020/1225, 2020. https://eprint.iacr.org/2020/1225.
60. Arpita Patra and Ajith Suresh. BLAZE: Blazing fast privacy-preserving machine learning. In *ISOC Network and Distributed System Security Symposium – NDSS 2020*, San Diego, CA, USA, February 23–26, 2020. The Internet Society.
61. Charith Peris, Christophe Dupuy, Jimit Majmudar, Rahil Parikh, Sami Smaili, Richard Zemel, and Rahul Gupta. Privacy in the time of language models. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 1291–1292, 2023.

62. Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.
63. M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: XNOR-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019: 28th USENIX Security Symposium*, pages 1501–1518, Santa Clara, CA, USA, August 14–16, 2019. USENIX Association.
64. Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis R. Bach. AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2022(1):291–316, January 2022.
65. Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. Grotto: Screaming fast $(2+1)$-PC for $\mathbb{Z}_{2^n}$ via (2, 2)-DPFs. Cryptology ePrint Archive, Report 2023/108, 2023. https://eprint.iacr.org/2023/108.
66. Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *2021 IEEE Symposium on Security and Privacy*, pages 1021–1038, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
67. Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
68. P.J. Van Fleet. *Discrete Wavelet Transformations: An Elementary Approach with Applications*. Wiley, 2019.
69. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
70. Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022(4):351–377, October 2022.
71. Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, July 2019.
72. Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021(1):188–208, January 2021.
73. Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.
74. Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU platform for secure computation. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 827–844, Boston, MA, USA, August 10–12, 2022. USENIX Association.
75. Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. Harnessing the power of LLMs in practice: A survey on ChatGPT and beyond. *ACM Transactions on Knowledge Discovery from Data*, 2023.
76. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.
77. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Proof of Theorem 1

We consider the probabilistic truncation protocol of Escudero et al. [25], denoted $\Pi_{n,f}^{\mathsf{TruncPr}}(\llbracket x \rrbracket)$ and shown in Fig. 9. Below we prove Theorem 1, i.e. we show that protocol $\Pi_{n,f}^{\mathsf{TruncPr}}(\llbracket x \rrbracket)$ securely realizes the modified probabilistic truncation functionality $\mathcal{F}_{n,f}^{\mathsf{TruncPr}}(\llbracket x \rrbracket)$ shown in Fig. 3 in Section 3. To simplify notation, we assume below that $\ell = n-1$, in which case $\tilde{s} = s = (x + \tilde{r}) \bmod 2^n$ for $\tilde{r} = 2^{n-1}b + 2^f r + r'$. In the general case of $\ell < n$ all arguments are the same except that computation is done over $\mathbb{Z}_{2^{\ell+1}}$ instead of $\mathbb{Z}_{2^n}$.

$\Pi^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$

1. Assume parties hold $[\![x]\!] \in \mathbb{Z}_{2^n}$ for $x < 2^\ell$, $\ell < n$.
2. Assume parties hold $([\![r]\!], [\![r']\!], [\![b]\!])$ from pre-processing where r, r', b are random in resp. $\mathbb{Z}_{2^{\ell-f}}$, $\mathbb{Z}_{2^f}$, and $\mathbb{Z}_2$.
3. Parties set $[\![s]\!] = 2^{n-(\ell+1)} \cdot ([\![x]\!] + 2^\ell [\![b]\!] + 2^f [\![r]\!] + [\![r']\!])$ and publicly reconstruct s from $[\![s]\!]$.
4. Denote $s = 2^{n-(\ell+1)} \cdot \tilde{s}$ and note that $\tilde{s} = (x + \tilde{r}) \bmod 2^{\ell+1}$ where $\tilde{r} = 2^\ell b + 2^f r + r' < 2^{\ell+1}$.
5. Parties compute $[\![v]\!] = [\![b \oplus \tilde{s}_\ell]\!] = [\![b]\!] + \tilde{s}_\ell \cdot (1 - 2[\![b]\!])$, where $\tilde{s}_\ell$ is the ℓ-th bit of $\tilde{s}$.
6. Parties output $[\![x']\!] = 2^{\ell-f}[\![v]\!] + \lfloor (\tilde{s} \bmod 2^\ell)/2^f \rfloor - [\![r]\!]$.

Fig. 9: Probabilistic truncation protocol of [25].

Proof. The simulator receives the shares of corrupted parties in sharings $[\![x]\!], [\![r]\!], [\![r']\!], [\![b]\!]$ and a random value $s_{(f)} \in \mathbb{Z}_{2^f}$ from functionality $\mathcal{F}^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$. On these inputs the simulator picks a random value $s^{(f)}$ in $\mathbb{Z}_{2^{n-f}}$, sets $s = 2^f \cdot s^{(f)} + s_{(f)}$, and simulates the reconstruction of secret-sharing $[\![s]\!] = [\![x + \tilde{r}]\!] = [\![x]\!] + 2^{n-1}[\![b]\!] + 2^f[\![r]\!] + [\![r']\!]$ into value s. Note that all other computations in the protocol are local given the input shares and the revealed value s, so the simulator's work is done.

Note that $\Pi^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$ reveals only s reconstructed from secret-sharing $[\![s]\!] = [\![x+\tilde{r}]\!]$. Since $s = x+\tilde{r} \bmod 2^n$ and $\tilde{r}$ is random in $\mathbb{Z}_{2^n}$ it follows that s is uniform in $\mathbb{Z}_{2^n}$ for every x, hence the simulation above is perfect. Note also that since the simulator sets s as $2^f \cdot s^{(f)} + s_{(f)}$, output $s_{(f)}$ in the simulation agrees with value $s_{(f)}$ set by the functionality.

It remains to argue that distribution of x' output by the $\Pi^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$ is the same as in the ideal-world functionality $\mathcal{F}^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$. In the ideal-world functionality, value x' is a function of x and $s_{(f)}$, determined by the following formula:

$$x' = \begin{cases} x^{(f)} & \text{if } s_{(f)} \geq x_{(f)}, \\ x^{(f)} + 1 & \text{otherwise.} \end{cases} \tag{3}$$

We argue that x' is determined in the same way in protocol $\Pi^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$. Recall that for any v, we denote $v^{(f)} = \lfloor v/2^f \rfloor$ and $v_{(f)} = (v \bmod 2^f)$, hence $v = v^{(f)} \cdot 2^f + v_{(f)}$. Correctness of the protocol is based on the observation that if $s = x + \tilde{r} \bmod 2^n$ then $s = x + (2^f r + r') - 2^{n-1}v$ over integers, where $v = b \oplus s_{n-1}$, equivalently $s = 2^f(x^{(f)} + (r - \tilde{v})) + (x_{(f)} + r')$ where $\tilde{v} = (2^{(n-1)-f})v$. There are two cases for how $s^{(f)}$ and $s_{(f)}$ relate to $x^{(f)}, r, \tilde{v}$ and $x_{(f)}, r'$:

$$s^{(f)} = \begin{cases} x^{(f)} + (r - \tilde{v}) & \text{if } x_{(f)} + r' < 2^f, \\ x^{(f)} + (r - \tilde{v}) + 1 & \text{otherwise.} \end{cases} \tag{4}$$

$$s_{(f)} = \begin{cases} x_{(f)} + r' & \text{if } x_{(f)} + r' < 2^f, \\ x_{(f)} + r' - 2^f & \text{otherwise.} \end{cases} \tag{5}$$

Since $[\![x']\!]$ is set to $s^{(f)} - [\![r - \tilde{v}]\!]$, equation (4) implies that:

$$x' = s^{(f)} - (r - \tilde{v}) = \begin{cases} x^{(f)} & \text{if } x_{(f)} + r' < 2^f, \\ x^{(f)} + 1 & \text{otherwise.} \end{cases} \tag{6}$$

However, equation (5) implies that:

$$\begin{array}{l} x_{(f)} + r' < 2^f \text{ if and only if } \quad s_{(f)} = x_{(f)} + r' \geq x_{(f)} \\ x_{(f)} + r' \geq 2^f \text{ if and only if } s_{(f)} = x_{(f)} + (r' - 2^f) < x_{(f)}. \end{array} \tag{7}$$

Equations (6) and (7) together imply the same expression of x' according to $x_{(f)}$ and $s_{(f)}$ as in equation (3). We conclude that x' output by protocol $\Pi^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$ is set by the same formula as in the ideal functionality $\mathcal{F}^{\mathsf{TruncPr}}_{n,f}([\![x]\!])$. □

In the proof above, note that the randomness of the opened masked value is part of the view of the adversary, so the rounding error is fixed in both the real and ideal worlds by the adversary's view. This argument can be extended to all probabilistic truncation protocols that follow the same pattern regarding masking the input value with randomness. More specifically, the proof can be adapted for the protocols with non-zero correctness error, including [8] and [15], and for other protocols with perfect correctness, e.g., [13].

B Bounded functions

Recall that an odd function F obeys the following property $\mathsf{F}(-x) = -\mathsf{F}(x)$ and an even function F is such that $\mathsf{F}(-x) = \mathsf{F}(x)$. In this section, we generalize the approach taken by Sigma [33] for the GeLU function [33] and apply it to any bounded odd or even function that converges to some constant c for $|x| \to \infty$ (e.g., s-shape functions). This method can be applied to several important functions: s-shape functions, used as activation functions (hyperbolic tangent tanh and error function erf); shifted s-shape functions, like sigmoid; and more general functions such as GeLU and SiLU.

The odd/even property guarantees that we only need to compute the function for positive values. This effectively halves the size of the required LUT. To achieve this, we compute the sign of the input and multiply it by its value, then get its absolute value. The convergence property ensures that, given some acceptable error, we can clip the function for some interval $[-a, a]$ and assign a constant value c to all elements outside that interval. We note that [33] sets $c = \mathsf{F}(a)$ but we set the constant c to be the limit value of the function F. Outside $[-a, a]$, we claim that the error incurred by [33] is bigger for most of the inputs but for a short interval. Intuitively, Fig. 10 shows the errors incurred by Curl's and Sigma's approach to the Sigmoid function. Observe that the error following Sigma's approach (red area) is bigger than the error by Curl's approach (green area) outside the interval $[2, 2.76]$.

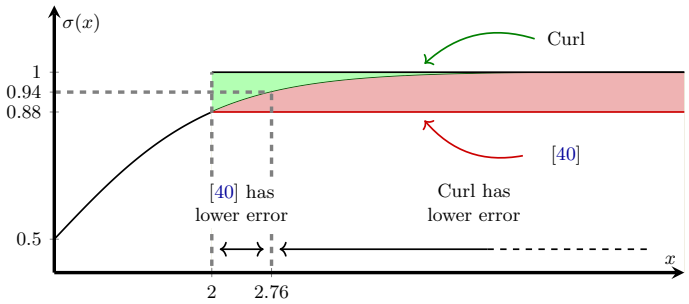


Fig. 10: Analysis of error incurred by Curl's (green) and Sigma's (red) approach on approximating Sigmoid function $\sigma(x)$ for $x > 2$. Curl's approach sets the approximation $\widetilde{\sigma}(x) = 1$ and Sigma's approach sets $\widetilde{\sigma}(x) = \sigma(2)$ for $x > 2$. Sigma's approach error is lower in $[2, 2.76)$ and Curl's approach error is lower in $(2.76, +\infty)$.

The $\Pi^{\mathsf{F}}_{n,a,c,p,j}(\llbracket x \rrbracket)$ protocol is formally shown in Fig. 11. Recall from Section 2.1, that $\Pi^{\mathsf{A2B}}_{n,l}$ and $\Pi^{\mathsf{B2A}}_{n,l}$ represent the arithmetic to binary and binary to arithmetic conversion protocols. The protocol goes as follows. For input value x, the parties start by computing the signal of x and its absolute value. Note that the signal value can be computed using the expression $\mathrm{sgn}(x) = 1 - 2 \cdot (x < 0)$ and the absolute value of x can be computed as $|x| = \mathrm{sgn}(x) \cdot x$. The absolute value is then used to check whether x lies within a predetermined interval $[-a, a]$ or not. Then, parties run the secure Biorthogonal protocol $\Pi^{\mathsf{Bior\text{-}LUT}}_{n,l,j,\mathsf{F}}(\llbracket x \rrbracket)$ to evaluate on $|x|$, returning y'. In case $x \in [-a, a]$, parties output y', otherwise they output the limit value c of the function F. This is effectively computed by the operation $i \cdot y' + (1 - i) \cdot c$, where i is the comparison bit for $|x| < a$. In the last step, the output value is corrected in case the function is odd: returning $\mathrm{sgn}(x) \cdot y$. This is set by the parity parameter p, which is $p = 0$ for even functions and $p = 1$ for odd functions.

$\Pi^{\mathsf{F}}_{n,a,c,p,j}(\llbracket x \rrbracket)$

Parameters: n is the full bit-width, a defines the interval in which the bior-LUT protocol is applied, c is the constant value for the function outside the interval, p is the parity of F ($p = 1$ for odd functions and $p = 0$ for even functions) and j is the compression variable for the bior-LUT protocol.

Compute sign and absolute value, $\llbracket \mathbf{sgn}(x) \rrbracket$ and $\llbracket |x| \rrbracket$:

1. $\langle\langle x \rangle\rangle \leftarrow \Pi^{\mathsf{A2B}}_{n,n}(\llbracket x \rrbracket) \in \mathbb{Z}_2^n$
2. $\langle\langle b \rangle\rangle \leftarrow \langle\langle y \rangle\rangle \gg (n-1) \in \mathbb{Z}_2$
3. $\llbracket b \rrbracket \leftarrow \Pi^{\mathsf{B2A}}_{1,n}(\langle\langle b \rangle\rangle) \in \mathbb{Z}_{2^n}$
4. $\llbracket \mathrm{sgn}(x) \rrbracket \leftarrow 1 - 2 \cdot \llbracket b \rrbracket \in \mathbb{Z}_{2^n}$
5. $\llbracket |x| \rrbracket \leftarrow \llbracket \mathrm{sgn}(x) \rrbracket \cdot \llbracket x \rrbracket \in \mathbb{Z}_{2^n}$

Compute comparison bit: $b = x < 0$.

Check $|x| < a$:

6. $\llbracket z \rrbracket \leftarrow \llbracket |x| \rrbracket - a \in \mathbb{Z}_{2^n}$
7. $\langle\langle z \rangle\rangle \leftarrow \Pi^{\mathsf{A2B}}_{n,n}(\llbracket z \rrbracket) \in \mathbb{Z}_2^n$
8. $\langle\langle i \rangle\rangle \leftarrow \langle\langle z \rangle\rangle \gg (n-1) \in \mathbb{Z}_2$
9. $\llbracket i \rrbracket \leftarrow \Pi^{\mathsf{B2A}}_{1,n}(\langle\langle i \rangle\rangle) \in \mathbb{Z}_{2^n}$

Compute comparison bit: $i = |x| < a$.

Compute F:

10. $\llbracket y' \rrbracket \leftarrow \Pi^{\mathsf{Bior\text{-}LUT}}_{n,n,j,\mathsf{F}}(\llbracket |x| \rrbracket)$
11. $\llbracket y \rrbracket \leftarrow (\llbracket i \rrbracket \cdot \llbracket y' \rrbracket + (1 - \llbracket i \rrbracket) \cdot c)$ ▷ *Computes share of:* $y = \begin{cases} y', & \text{if } |x| < a \\ c, & \text{otherwise.} \end{cases}$
12. Output: $\llbracket \mathrm{sgn}(x) \rrbracket^p \cdot \llbracket y \rrbracket \in \mathbb{Z}_{2^n}$ ▷ *Equivalent to $\llbracket y \rrbracket$ for even functions and $\llbracket -y \rrbracket$ otherwise.*

Fig. 11: Protocol for convergent bounded odd ($p = 1$) or even ($p = 0$) function F based on the secure biorthogonal-encoded LUT protocol.

B.1 S-shape functions

In the previous section, we presented a protocol that can be applied to any bounded odd or even function that is convergent. S-shape functions are bounded functions that converge to some value c for $|x| \to \infty$ but are not necessarily odd or even. For example, we have that tanh and erf are odd functions but the sigmoid function $\sigma(x)$ is not. However, we note that we can transform any s-shape function to an odd function by applying a shift. For example, the sigmoid function $\sigma(x)$ can be shifted by $1/2$, resulting in an odd function: $\sigma(x) - 1/2$.

Generally, observe that, for an s-shape function F, there exists a shift value s, such that $\mathsf{F}(-x) - s = -\mathsf{F}(x) + s \Leftrightarrow \mathsf{F}(-x) = 2 \cdot s - \mathsf{F}(x)$. So, setting for $b = x < 0$, we can change the last step (12) of protocol $\Pi^{\mathsf{F}}_{n,a,c,p,j}(\llbracket x \rrbracket)$ to evaluate the function F:

$$\begin{aligned} \mathsf{F}(x) &= b \cdot \mathsf{F}(-|x|) + (1-b) \cdot \mathsf{F}(|x|) \\ &= b \cdot (2 \cdot s - \mathsf{F}(|x|)) + (1-b) \cdot \mathsf{F}(|x|) \\ &= b \cdot 2 \cdot s + \mathrm{sgn}(x) \cdot \mathsf{F}(|x|), \end{aligned}$$

where $\mathrm{sgn}(x) = (1 - 2 \cdot b)$. Thus, for the sigmoid function where $s = 1/2$, the last step (12) from Fig. 11 is as follows: $\llbracket b \rrbracket + \llbracket \mathrm{sgn}(x) \rrbracket \cdot \llbracket y \rrbracket$. Note that for both tanh and erf $b = 0$, recovering the initial expression.

Parameters. Now, for each of the three s-shape functions, we need to define the following parameters: bitwidth n, the interval $[-a, a]$, the constant c, the parity p and the compression value j. We use the hardcoded bitwidth value from CrypTen $n = 64$, imposed by the PyTorch dependency. Also, $c, p = 1$ for all three s-shape

functions. We note that [33] clips the interval in GeLu and accepts a constant error on the order of $\sim 10^{-5}$. We follow a similar approach and accept errors $< 10^{-5}$. For the error function (erf), hyperbolic tangent (tanh) and sigmoid ($\mathsf{sigmoid}$) we set the interval to be $[-4, 4]$, $[-8, 8]$ and $[-16, 16]$, respectively. These intervals incurr a maximum error on the order of 10^{-8}, 10^{-7} and 10^{-7}, and the size of the original LUT is 2^{18}, 2^{19} and 2^{20}, respectively.

B.2 Activation Functions: GeLU And SiLU

In this section, we consider two popular activation functions: GeLU [37] and SiLU [23], given by

$$\mathsf{GeLU}(x) \coloneqq \frac{x}{2}\left(1 + \mathsf{erf}\left(\frac{x}{\sqrt{2}}\right)\right),$$

where $\mathsf{erf}(\cdot)$ denotes the Gauss error function and

$$\mathsf{SiLU}(x) \coloneqq x \cdot \mathsf{sigmoid}(x).$$

These activation functions are sometimes considered smooth approximations to the classical ReLU of [27]. We observe that the functions $\mathsf{D}_{\mathsf{GeLU}} \coloneqq \mathsf{ReLU}(x) - \mathsf{GeLU}(x)$ and $\mathsf{D}_{\mathsf{SiLU}} \coloneqq \mathsf{ReLU}(x) - \mathsf{SiLU}(x)$ are even functions. Regarding $\mathsf{D}_{\mathsf{GeLU}}$, note that

$$\mathsf{D}_{\mathsf{GeLU}} = \frac{1}{2}\left(|x| - x\,\mathsf{erf}\left(\frac{x}{\sqrt{2}}\right)\right),$$

so the claim follows as $|x|$ is even and x and erf are both odd. Regarding $\mathsf{D}_{\mathsf{SiLU}}$, observe that $\sigma(-x) = 1 - \sigma(x)$, for sigmoid σ. Thus, we have:

$$\mathsf{D}_{\mathsf{SiLU}}(x) = \begin{cases} x \cdot (1 - \sigma(x)) & x \geq 0, \\ -x \cdot \sigma(x) & x < 0 \end{cases} = \begin{cases} x \cdot \sigma(-x) & x \geq 0, \\ -x \cdot \sigma(x) & x < 0 \end{cases} = \mathsf{D}_{\mathsf{SiLU}}(-x).$$

Since these two activation functions are also bounded and converge to 0 for $|x| \to \infty$, we can use $\Pi^{\mathsf{F}}_{n,a,c,p,j}$ protocol (Fig. 11) to compute both GeLU and SiLU by evaluating: $\mathsf{GeLU}(x) = \mathsf{ReLU}(x) - \Pi^{\mathsf{D}_{\mathsf{GeLU}}}_{n,a,c,p,j}(x)$ and $\mathsf{SiLU}(x) = \mathsf{ReLU}(x) - \Pi^{\mathsf{D}_{\mathsf{SiLU}}}_{n,a,c,p,j}(x)$.

Note that ReLU can be computed using $\mathsf{ReLU}(x) = (x < 0) \cdot x$. Since we compute $x < 0$ in step 3. of the $\Pi^{\mathsf{F}}_{n,a,c,p,j}$ protocol (Fig. 11), we can reuse this value.

Parameters. Again, we use the hardcoded bitwidth value from CrypTen $n = 64$ and $c, p = 0$. As noted before, [33] clips the interval in GeLU and accepts a constant error on the order of $\sim 10^{-5}$. We follow a similar approach and accept errors $< 10^{-5}$. For GeLU and SiLU we set the interval to be $[-4, 4]$ and $[-16, 16]$, respectively. These intervals incur a maximum error on the order of 10^{-8} and 10^{-7}, and the size of the original LUT is 2^{18} and 2^{20}, respectively.

B.3 Trigonometric functions

It is known that sin is an odd function and cos is an even function. However, these two functions do not converge, making protocol $\Pi^{\mathsf{F}}_{n,a,c,p,j}$ (Fig. 11) unsuitable. Despite this, we can still use the core concepts from that protocol and adapt them to these trigonometric functions. The key insight is that both functions are periodic with a period of 2π. We leverage this periodicity by scaling these trigonometric functions to have period 1. This is achieved by generating LUT for $\mathsf{sin}(2\pi \cdot x)$ and $\mathsf{cos}(2\pi x)$, where $x \in [0, 1]$. Then, for input x, we evaluate the LUT using the secure Biorthogonal protocol at point $x/(2\pi)$. We execute division (Π^{div}) based on the Escudero et al. [25, §5.1] probabilistic truncation protocol over rings. The protocol for sin is formally described in Fig. 12. The cos protocol is similar but does not require the sign correction in the last step.

$\Pi^{\mathsf{sin}}_{n,f,j}(\llbracket x \rrbracket)$

Parameters: n is the full bit-width, f is the number of bits used for the fractional part, and $p = 0$ for even functions) and j is the compression variable for the Biorthogonal protocol.

Compute sign and absolute value, $\llbracket \mathbf{sgn}(x) \rrbracket$ and $\llbracket |x| \rrbracket$:

1. $\langle\langle x \rangle\rangle \leftarrow \Pi^{\mathsf{A2B}}_{n,n}(\llbracket x \rrbracket)$
2. $\langle\langle b \rangle\rangle \leftarrow \langle\langle y \rangle\rangle \gg (n-1)$
3. $\llbracket b \rrbracket \leftarrow \Pi^{\mathsf{B2A}}_{1,n}(\langle\langle b \rangle\rangle)$
4. $\llbracket \mathrm{sgn}(x) \rrbracket \leftarrow 1 - 2 \cdot \llbracket b \rrbracket$
5. $\llbracket |x| \rrbracket \leftarrow \llbracket \mathrm{sgn}(x) \rrbracket \cdot \llbracket x \rrbracket$

Compute sin:

6. Use encoding of 2π, $q \coloneqq \lfloor 2\pi \cdot 2^f \rfloor$, and compute $\llbracket |x'| \rrbracket \leftarrow \Pi^{\mathsf{div}}(\llbracket |x| \rrbracket, q)$.
7. $\llbracket y' \rrbracket \leftarrow \Pi^{\mathsf{Bior\text{-}LUT}}_{n,n,j,\mathsf{sin}(2\pi \cdot)}(\llbracket |x'| \rrbracket)$
8. Output: $\llbracket \mathrm{sgn}(x) \rrbracket \cdot \llbracket y \rrbracket \in \mathbb{Z}_{2^n}$ ▷ *For* sin *it returns* $\llbracket y \rrbracket$.

Fig. 12: Protocol for sin function based on the secure Biorthogonal protocol.

B.4 The Softmax Function

Recall that softmax is given by the following expression:

$$y_i = \frac{e^{x_i - \max(x)}}{\sum_{j=0}^{k-1} e^{x_j - \max(x)}},$$

where $x \in \mathbb{R}^k$ and max is the maximum operation over the vector x. To be able to compute the expression above, we need three main functions: maximum, natural exponentiation, and reciprocal. The maximum operation is computed using the protocol implemented in CrypTen (Section C.1.4 [46]). We note that both CrypTen adopts a tree-reduction algorithm but also provides a pairwise method with constant round complexity.

Reciprocal. From the softmax expression, we have that there exists at least one index j such that $e^{x_j - \max(x)} = 1$. Also, since there are at most k terms, we have that the reciprocal function can be only evaluated within the $[1, k]$ interval. We set $k = 64$ as per in GPT-2 model [62]. Then, we apply the secure $\Pi^{\mathsf{Bior\text{-}LUT}}_{n,l,j,\mathsf{F}}$ to the interval $[1, 64]$.

Table 5: Comparison of the mean absolute error (MAE) between Haar and Bioriothogonal LUT protocol with probabilistic and deterministic truncation in $(1, 64)$.

Op.	Probabilistic Truncation		Deterministic Truncation	
	Haar	Biorthogonal	Haar	Biorthogonal
log	8.63e-3	2.53e-2	9.76e-03	4.63e-04
reciprocal	2.26e-3	4.58e-4	1.87e-03	5.79e-04
sqrt	1.29e-1	8.84e-4	6.04e-02	6.41e-05
invsqrt	7.58e-4	5.40e-5	9.36e-04	2.20e-05

§ The light blue background indicates the lowest MAE error.

C Truncation with Haar vs Biorthogonal

We compare the accuracy of Haar and Biorthogonal DWT approaches using both probabilistic and deterministic truncation methods in Table 5. Our findings indicate that the Biorthogonal approach consistently outperforms the Haar approach. The reason for this is that Haar averages out two consecutive values whereas Birthogonal uses five consecutive elements, thus retaining more information and capturing the trend of the function. Finally, for the both Haar and Biorthogonal approaches we observe only a marginal difference between the probabilistic and deterministic truncation protocols.