Technical Report on Secure Truncation with Applications to LLM Quantization

Mehmet Ugurbil, Miguel de Vega, Wicher Malten, and Manuel B. Santos

Nillion

December 29, 2023

Abstract

Over the last decade, machine learning has gained significant attention and is now being used extensively in practice. Quantization, especially for large language models, has been a useful technique to increase speed and reduce memory footprint, making these techniques more accessible and deployable in resource-constrained environments. The popularity of machine learning and the need for data privacy has led to the development of privacy-preserving machine learning systems, many of which use secure multi-party computation.

The arithmetic required for machine learning in secure multi-party computation typically relies on fixed-point arithmetic, as floating points are expensive in practice. Fixed-point arithmetic employs truncation for secure multiplication and other non-linear operations. We propose a new framework to securely truncate with perfect security and low memory footprint, which is especially efficient when the bit-width is low. For 8 bit fields, this reduces communication complexity by 61% and memory usage by 83%.

1 Introduction

Machine learning, and in particular deep learning, has emerged as a transformative technology with applications spanning various domains, from healthcare and finance to autonomous vehicles and natural language processing such as large language models (LLM) [27, 10]. The ability to analyze vast datasets and make data-driven predictions has revolutionized industries and enhanced decision-making processes [18, 21]. However, as the adoption of machine learning continues to grow, so do the concerns related to data privacy and security [22, 26]. The need to strike a balance between reaping the benefits of machine learning and safeguarding individual and organizational privacy has given rise to the field of privacy-preserving machine learning [24, 17].

Privacy-preserving machine learning using secure multi-party computation (MPC) has gained significant attention in recent years due to the property that sensitive information is not disclosed during the computation. MPC techniques include garbled circuits [2, 30, 31], function secret sharing [34, 16], fully homomorphic encryption based MPC [25, 35], and linear secret sharing schemes (LSSS) [19, 11, 5]. Most of these methods rely on 2 party computation. Function secret sharing for instance, has great applications and is very efficient for 2 party computation. For example, Sigma [16] makes LLM inference fast and practical. If we want to scale beyond 2 parties, however, some of these techniques are no longer efficient.

Quantization is a well known technique in deep learning for decreasing memory footprint as well as increasing computational performance [36, 14, 9]. The basic idea behind quantization is to represent the weights and activations of a neural network using a lower number of bits, such as 4 or 8 bit integers, rather than the standard 32-bit or 64-bit floating-point precision. This can lead to significant reductions in both model size and memory requirements, making it more feasible to deploy models on resource-constrained devices, such as mobile phones or edge devices. Secure computation gives rise to significant overhead, which is also reduced by quantization [6].

Truncation is a fundamental building block for secure integer and fixed-point arithmetic [3, 4, 12] that is needed in privacy-preserving machine learning. Typically, after a multiplication or an inner product fixed-point numbers, one must employ truncation to reduce its precision. Therefore, truncation is a basic building block for efficient quantization.

Typically, general-purpose truncation protocols for fixed-point arithmetic, as discussed in [3], need a representation with an augmented number of bits to ensure statistical privacy. This situation undermines the idea of leveraging the benefits of quantized models, given that the size of the statistical parameter often exceeds the quantized size. However, there exist techniques for truncation in rings compatible with a quantized approach [6]. In this work, we introduce a general multi-party perfectly secure truncation protocol designed for small fields. As a consequence, we reduce the representation cost observed in previous works [3] incurred by statistical security parameter.

2 Transformer Architecture

The transformer architecture is a type of neural network architecture [33] that uses attention mechanisms, which allows the model to focus on different parts of the input sequence when generating an output. It has become a foundational model for various natural language

processing (NLP) tasks and has been widely adopted in machine translation, text generation, and other sequence-to-sequence tasks. The transformer architecture is a fundamental building block for LLMs. Several different parts of an LLM architecture require the use of truncation. We now briefly describe them.

Transformers process sequences of tokens, where each token is typically a word or subword, and represent them as one-dimensional vectors of size d. These vectors are often referred to as embeddings. Next, transformers rely on self-attention, which is a mechanism that allows a system to weigh the importance of different parts of an input sequence when making predictions. It takes a query matrix Q and a set of key-value matrix pairs K, V and produces an output as follows:

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

To compute matrix multiplication we run many inner products that need to be followed by truncation. For softmax, one could make use of exponential calculated by approximating the limit [19]: $\lim_{n\to\infty}(1+\frac{x}{n})^n$, followed by division using Newton-Raphson approximation [20]. Similarly, the inverse square root could also be calculated using Newton-Raphson method [20]. These all require multiplications followed by truncation.

Next in the transformer architecture comes the feed forward neural network. These are computed using an inner product followed by an activation function. ReLU or GeLU activations are commonly used. ReLU can be implemented with only a comparison, while GeLU can be approximated by a ReLU plus a Taylor expansion of the difference around zero [16]. Finally, layer normalization uses inverse square root. Similarly, these elements require multiplications followed by truncation.

3 Data Representation

We follow the approach from previous works [3, 6] and we consider secure computation with binary values encoded as 0 and 1, signed integers and fixed-point rationals. Signed integer values are defined by $\mathbb{Z}_{\langle k\rangle} = \{\bar{x} \in \mathbb{Z} | -2^{k-1} \leq \bar{x} < 2^{k-1}\}$, and encode values using $\texttt{encode} : \mathbb{Z}_{\langle k\rangle} \to \mathbb{Z}_q$ as $\texttt{encode}(\bar{x}) = \bar{x} \mod q$, where $q > 2^k$. Fixed-point rational numbers are encoded as integers $\bar{x} = \tilde{x}2^f \in \mathbb{Z}_{\langle k\rangle}$, defined by $\mathbb{Q}_{\langle k,f\rangle} = \{\tilde{x} \in \mathbb{Q} | \tilde{x} = \bar{x} \cdot 2^{-f}, \bar{x} \in \mathbb{Z}_{\langle k\rangle}\}$. Fixed-point multiplication in MPC requires $q > 2^{2k}$ to prevent overflows.

Quantization from a floating-point number x to a fixed-point number $\tilde{x}$ can be achieved by picking a precision parameter f, then multiplying the number by 2^f and rounding down to the nearest integer.

$$\tilde{x} = \lfloor x \cdot 2^f \rfloor$$

The fixed-point number can be converted back simply via a division by 2^f. Note that accuracy may be lost during the quantization, hence the converted back number might loose precision.

$$x \approx \frac{\tilde{x}}{2^f}$$

This is akin to *uniform quantization* in machine learning given by the formula $Q(x) = \texttt{int}(x/S) + Z$ [14], where Q is the quantization function, x is the real valued input, S is the

real valued scaling factor, `int` maps input to the nearest integer value, and Z is the integer zero point.

4 Trust Model

We will set up a truncation protocol which is secure in the same setting as the underlying LSSS-based MPC protocol. Examples include:

- BGW/GRR with $t < n/2$ passive corruptions [1, 13]
- ATLAS with $t < n/2$ active corruptions [15]

For performance calculations, we work in the passive adversary setup, where there are $t < n/2$ corrupt parties. Although our protocol works for any linear secret sharing scheme, we assume all values are secret shared using Shamir's secret sharing scheme [32] over a prime field $\mathbb{F}_q$.

Further, perfect security against an active adversary that can corrupt $t < n/3$ parties can be achieved using verifiable secret sharing (VSS) [13] and maliciously secure protocols [29] such as BGW.

5 Truncation in Small Field

Various forms of overhead occur when we use MPC protocols. Specifically, when performing multiplication, one needs to double the field size to prevent overflows. For example, multiplication of two 8 bit integers may lead to a 16 bit integer. Additionally, when implementing truncation, extra statistical security bits λ - typically around 40 bits - are crucial for the single-round probabilistic truncation protocol to work securely as outlined in [3]. Further, we either need an even larger modulus for pseudo random integer generation [7] or to generate the random integer using random bits, which requires a random number along with secure multiplication [8].

Working with $128+\lambda$ bits (128 to prevent multiplication overflow and 40 for statistical security) for 64-bit arithmetic does not seem an excessive overhead. However, for smaller sizes it defeats the purpose of quantization. For example, for 8-bit arithmetic, this turns 16-bit fields into 56-bit fields, more than 3 times as many bits!

We can, however, get rid of the statistical bits in exchange for a comparison protocol to calculate *bitwise less than* [23, 3, 8] as in $\pi_{\texttt{TRUNC-PR}}$ below. This protocol works in a finite field F_q, where $l = \lceil \log_2 q \rceil > 2k$ for secret input $\tilde{x} \in \mathbb{Q}_{\langle k,f \rangle}$. It takes the input in secret shared signed integer form $[x]$ and clear value m returning a sharing of $\lfloor x/2^m \rfloor + u$, where there is a random term $u \in \{0, 1\}$ [8, 28]. On a high level, we do this by first generating a random value r that is bitwise shared as $([r_0], \ldots, [r_{l-1}])$ with $r_i \in \{0, 1\}$. Then we reveal the secret masked by our random value. Using a bitwise comparison, we check if there was an overflow and we use this in calculating the remainder of the revealed value by 2^m. We then use this with the first m bits of the bitwise random number to securely calculate the truncated value.

Quantized Probabilistic Truncation, $\pi_{\texttt{TRUNC-PR}}$.

$$[x/2^m] \leftarrow \pi_{\texttt{TRUNC-PR}}([x], m)$$

1. $([r_0], \ldots, [r_{l-1}]) \leftarrow F_{\texttt{RAN-BITWISE}}()$
   (a) $[r'] \leftarrow \sum_{i=0}^{m-1} 2^i \cdot [r]_i$
   (b) $[r] \leftarrow \sum_{i=0}^{l-1} 2^i \cdot [r]_i$
2. $c \leftarrow F_{\texttt{REVEAL}}(2^{k-1} + [x] + [r])$
3. $[c < r] \leftarrow F_{\texttt{BIT-LESS-THAN}}(c, ([r_0], \ldots, [r_{l-1}]))$
4. $[c'] \leftarrow (1 - [c < r]) \cdot (c \mod 2^m) + [c < r] \cdot (c + q \mod 2^m)$
5. $[w] \leftarrow [x] + [r'] - [c']$
6. $[x/2^m] \leftarrow [w] \cdot (2^m)^{-1}$
7. OUTPUT $[x/2^m]$

Correctness: Note that the secret $\tilde{x} \in \mathbb{Q}_{\langle k,f \rangle}$ is first translated into $\bar{x} \in \mathbb{Z}_{\langle k \rangle}$ as $\bar{x} = \tilde{x}2^f$ which is encoded as $x = \bar{x} \mod q$. Let $b = (2^{k-1} + \bar{x}) \mod q$, hence $0 \leq b < 2^k$. Next $b' = b \mod 2^m$, $x' = \bar{x} \mod 2^m$, then $b' = x'$ for any $0 < m < k$. In the protocol, we first generate a bitwise shared random number in the field during preprocessing. This vector of sharings of bits we denote $([r_0], \ldots, [r_{l-1}])$, and we set $r \leftarrow \sum_{i=0}^{l-1} 2^i \cdot r_i$, where r_i is the i-th bit of r and $l = \lceil \log_2 q \rceil > 2k$. We set $r' = r \mod 2^m$. We add r to the secret as well as the 2^{k-1} term to account for negative numbers, revealing the result $c = b + r \mod q$. Now, if $c \geq r$, then $c = b + r$, else $c = b + r - q$, thus $c = b + r - (c < r) \cdot q$. Thus, we compare the clear number c to the bitwise shared random number r to check for overflows. Then, we take the modulo 2^m in the clear as follows, $c' = c \mod 2^m$ if $c \geq r$, and else if $c < r$ then $c' = c + q \mod 2^m$; or in a single equation $c' = (1 - (c < r)) \cdot (c \mod 2^m) + (c < r) \cdot (c + q \mod 2^m)$. If $c \geq r$, then $c' = (b + r) \mod 2^m = b' + r' - 2^m \cdot u$, where $u \in \{0, 1\}$. Else if $c < r$, then there was an overflow, which means $b + r > q$, hence $b + r = c + q$. Then as before, $c' = (c + q) \mod 2^m = b' + r' - 2^m \cdot u$, where $u \in \{0, 1\}$. We get $x' = b' = c' - r' + 2^m \cdot u$, so that $w = x + r' - c' = x - x' - 2^m \cdot u$, but $x - x' = 2^m \lfloor x/2^m \rfloor$. Thus, $w \cdot (2^m)^{-1} = (2^m \lfloor x/2^m \rfloor - 2^m \cdot u) \cdot (2^m)^{-1} = \lfloor x/2^m \rfloor - u$. This is the probabilistic truncation result we wanted. Note that $Pr(u = 1) = Pr(x' + r' \geq 2^m)$, which is the rounding property.

Security: We can leak information only in steps where a shared value is reconstructed. These values are of the form $y = x + r$, where the secret is x and the random value r is uniformly random in the field. Since r is uniformly random, $x + r$ is a one-time padding of secret x, therefore is perfectly secure. Since the sub-protocols $F_{\texttt{RAN-BITWISE}}$ [28] and $F_{\texttt{BIT-LESS-THAN}}$ [23] are perfectly secure, we conclude that $\pi_{\texttt{TRUNC-PR}}$ is perfectly secure.

Performance: Let us assume that the random bits are produced in the pre-processing

phase. Regarding the online round complexity, we have $\log_2 \log_2 q - 1$ rounds for the comparison [23] protocol plus a round for the reveal operation, which results in a total of $\log_2 \log_2 q$ online rounds. Regarding the total communication complexity, we need a bitwise shared random number plus $l/2$ multiplications offline, l multiplications online for the comparison protocol and a reveal operation, resulting in a total communication complexity of $2.5l + 1$ field elements sent and received between any two nodes. We are assuming GRR style multiplication of Shamir shares where each node shares their share and sends a share to every other party [13]. In contrast, the probabilistic truncation protocol with statistical security [3] needs l random bits plus a reveal operation in the online phase. This is summarized in Table 1. It is interesting to see that our truncation protocol is better communication-wise than [3] only for small elements. For 32 bits, the statistical probabilistic truncation already requires less bandwidth. Specifically, the inflection point is for 27 bits for $\lambda = 40$ bits. This motivates the use of quantization to operate below this inflection point. Notice that regardless of the number of bits, we have less memory requirements than [3].

	Our Quantized Probabilistic Truncation	Statistical Probabilistic Truncation [3]	Ratio
Field Elements Sent	$2.5l + 1$	$l + 1$	$\frac{2.5l+1}{l+1}$
Bits Sent	$2.5l^2 + l$	$l^2 + (\lambda + 1)l + \lambda$	$\frac{2.5l^2+l}{l^2+(\lambda+1)l+\lambda}$
Memory per Element	l	$l + \lambda$	$\frac{l}{l+\lambda}$
Bits Sent, $l = 8$	168	432	0.389
Bits Memory, $l = 8$	8	48	0.167
Bits Sent, $l = 16$	656	952	0.689
Bits Memory, $l = 16$	16	56	0.286
Bits Sent, $l = 32$	2592	2376	1.090
Bits Memory, $l = 32$	32	72	0.444

Table 1: Comparison of communication and memory complexity for l bit prime. Communication is between any two nodes. We use $\lambda = 40$ bits for calculations.

6 Conclusion

We have presented a new MPC truncation protocol for linear secret sharing schemes that works well in small fields, therefore is useful for quantization. This has implications in machine learning applications, specifically in LLM quantization, where model weights might be stored and computed in smaller fields. By working in a small field without a statistical security parameter, we can load larger models into memory using quantization.

References

[1] Gilad Asharov and Yehuda Lindell. "A full proof of the BGW protocol for perfectly secure multiparty computation". In: *Journal of Cryptology* 30.1 (2017), pp. 58–151.

[2] Marshall Ball et al. "Garbled neural networks are practical". In: *Cryptology ePrint Archive* (2019).

[3] Octavian Catrina and Sebastiaan De Hoogh. "Improved primitives for secure multiparty integer computation". In: *Security and Cryptography for Networks: 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings 7*. Springer. 2010, pp. 182–199.

[4] Octavian Catrina and Amitabh Saxena. "Secure computation with fixed-point numbers". In: *Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14*. Springer. 2010, pp. 35–50.

[5] Valerie Chen, Valerio Pastro, and Mariana Raykova. "Secure computation for machine learning with SPDZ". In: *arXiv preprint arXiv:1901.00329* (2019).

[6] Anders Dalskov, Daniel Escudero, and Marcel Keller. "Secure evaluation of quantized neural networks". In: *arXiv preprint arXiv:1910.12435* (2019).

[7] Ivan Damgård and Rune Thorbek. "Non-interactive proofs for integer multiplication". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2007, pp. 412–429.

[8] Ivan Damgård et al. "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation". In: *Theory of Cryptography Conference*. Springer. 2006, pp. 285–304.

[9] Tim Dettmers et al. "Llm. int8 (): 8-bit matrix multiplication for transformers at scale". In: *arXiv preprint arXiv:2208.07339* (2022).

[10] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[11] Ye Dong et al. "Puma: Secure inference of llama-7b in five minutes". In: *arXiv preprint arXiv:2307.12533* (2023).

[12] Xiaoqi Duan et al. "ACCO: Algebraic Computation with Comparison". In: *Proceedings of the 2021 on Cloud Computing Security Workshop*. 2021, pp. 21–38.

[13] Rosario Gennaro, Michael O Rabin, and Tal Rabin. "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography". In: *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*. 1998, pp. 101–111.

[14] Amir Gholami et al. "A survey of quantization methods for efficient neural network inference". In: *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[15] Vipul Goyal et al. "ATLAS: efficient and scalable MPC in the honest majority setting". In: *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*. Springer. 2021, pp. 244–274.

[16] Kanav Gupta et al. "Sigma: Secure gpt inference with function secret sharing". In: *Cryptology ePrint Archive* (2023).

[17] Ehsan Hesamifard et al. "Privacy-preserving machine learning as a service." In: *Proc. Priv. Enhancing Technol.* 2018.3 (2018), pp. 123–142.

[18] Michael I Jordan and Tom M Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260.

[19] Brian Knott et al. "Crypten: Secure multi-party computation meets machine learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 4961–4973.

[20] Stan Korzilius and Berry Schoenmakers. "Divisions and Square Roots with Tight Error Analysis from Newton–Raphson Iteration in Secure Fixed-Point Arithmetic". In: *Cryptography* 7.3 (2023), p. 43.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[22] Bo Liu et al. "When machine learning meets privacy: A survey and outlook". In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–36.

[23] Wicher Malten, Mehmet Ugurbil, and Miguel de Vega. *More efficient comparison protocols for MPC*. Cryptology ePrint Archive, Paper 2023/1934. 2023.

[24] Payman Mohassel and Yupeng Zhang. "Secureml: A system for scalable privacy-preserving machine learning". In: *2017 IEEE symposium on security and privacy (SP)*. IEEE. 2017, pp. 19–38.

[25] Shinji Ono et al. "Privacy-Preserving Feature Selection with Fully Homomorphic Encryption". In: *Algorithms* 15.7 (2022), p. 229.

[26] Nicolas Papernot et al. "Towards the science of security and privacy in machine learning". In: *arXiv preprint arXiv:1611.03814* (2016).

[27] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[28] Tord Ingolf Reistad and Tomas Toft. "Secret sharing comparison by transformation and rotation". In: *International Conference on Information Theoretic Security*. Springer. 2007, pp. 169–180.

[29] Dragos Rotaru et al. "Actively secure setup for SPDZ". In: *Journal of Cryptology* 35.1 (2022), p. 5.

[30] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. "Deepsecure: Scalable provably-secure deep learning". In: *Proceedings of the 55th annual design automation conference*. 2018, pp. 1–6.

[31] Jonas Sander et al. "DASH: Accelerating Distributed Private Machine Learning Inference with Arithmetic Garbled Circuits". In: *arXiv preprint arXiv:2302.06361* (2023).

[32] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[33] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[34] Sameer Wagh. "Pika: Secure computation using function secret sharing over rings". In: *Cryptology ePrint Archive* (2022).

[35] Yichuan Wang et al. "Deep learning data privacy protection based on homomorphic encryption in AIoT". In: *Mobile Information Systems* 2021 (2021), pp. 1–11.

[36] Jiwei Yang et al. "Quantization networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7308–7316.