Fission: Distributed Privacy-Preserving Large Language Model Inference

Mehmet Ugurbil[1], Dimitris Mouris[1], Manuel B. Santos[1], José Cabrero-Holgueras[1], Miguel de Vega[1], and Shubho Sengupta[2]

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

[2] Meta Inc.*
shubho@gmail.com

Abstract. The increased popularity of large language models (LLMs) raises serious privacy concerns, where users' private queries are sent to untrusted servers. Many cryptographic techniques have been proposed to provide privacy, such as secure multiparty computation (MPC), which enables the evaluation of LLMs directly on private data. However, cryptographic techniques have been deemed impractical as they introduce large communication and computation. On the other hand, many obfuscation techniques have been proposed, such as split inference, where part of the model is evaluated on edge devices to hide the input data from untrusted servers, but these methods provide limited privacy guarantees.

We propose Fission, a privacy-preserving framework that improves latency while providing strong privacy guarantees. Fission utilizes an MPC network for linear computations, while nonlinearities are computed on a separate evaluator network that receives shuffled values in the clear and returns nonlinear functions evaluated at these values back to the MPC network. As a result, each evaluator only gets access to parts of the shuffled data, while the model weights remain private. We evaluate fission on a wide set of LLMs and compare it against prior works. Fission results in up to eight times faster inference and eight times reduced bandwidth compared to prior works while retaining high accuracy. Finally, we construct an attack on obfuscation techniques from related works that show significant information leakage, and we demonstrate how Fission enhances privacy.

Keywords: Applied Cryptography, Large Language Models, Machine Learning, Multiparty Computation

1 Introduction

Transformer-based generative language models have recently gained widespread attention due to their exceptional performance across various natural language tasks [8,45,15,37,4]. These models power applications ranging from generating content, translating languages, analyzing sentiment, and powering chatbots and virtual assistants. Despite their remarkable capabilities, the adoption of these models raises significant privacy concerns [5]. Individuals and organizations may share private or sensitive data with these systems, whether by transcribing conversations or providing private classifications. As these models become increasingly integrated into human lives, they frequently process sensitive information such as names, addresses, credit card details, or even financial and healthcare data. The challenge becomes even more pressing with the growing demand for personalized AI agents that adapt to individual preferences and workflows [34]. These agents require continuous access to sensitive user data, such as personal habits or speech patterns, to deliver customized outputs. While this enhances utility, it also creates significant risks of data breaches or unauthorized access. For instance, DeepSeek, a Chinese AI company, has faced significant privacy concerns, including data collection practices, data flows to China, and potential security risks, prompting scrutiny from regulators and security experts, which resulted in a country-wide ban in Italy as well as bans by the US Navy, New York State, and Texas governments [28,35,33].

* Majority of work performed while at Meta.

Privacy-enhancing technologies (PETs) such as secure multiparty computation (MPC) are essential for enabling privacy-preserving machine learning (PPML) [14,16,39]. In the most prominent setting, a model provider possesses a proprietary model and aims to provide it as a service to clients for use with their private data without gaining any knowledge about the model. At the same time, the model provider does not learn anything about clients' input. Several MPC systems leverage linear secret sharing schemes as they allow linear operations to be performed with a small running time overhead. Regardless, when evaluating large language models (LLMs) with multiple layers, these overheads add up, requiring extensive GPU acceleration to remain competitive [16,39]. Also, PPML approaches often suffer from reduced accuracy in models requiring multiple nonlinear computations [19,23].

To increase efficiency while maintaining privacy, split inference emerged as a practical approach that divides the computation between an edge device and a server [17]. In this method, part of the model is evaluated on the edge device until a cutoff layer, which is the output of a hidden layer. Once the server receives the cutoff layer, it finishes evaluating the model while learning minimal information about the raw inputs since it only sees the intermediate results. This method also limits the amount of computation the edge device has to do and offloads the work to a more powerful server. However, the cutoff layer and the model evaluation can still leak a great deal of information, even though the raw input data are not revealed. There have been many proposed successful attacks that reconstruct the input data or gain meaningful information from split inference techniques [17,32,26,11].

Considering these vulnerabilities, we introduce Fission, a hybrid PPML architecture that mitigates the limitations of prior approaches. Before describing Fission in detail, we go over related works and outline our contributions.

1.1 Related Work

Cryptographic solutions. Several PET implementations strive for complete privacy of both model and input prompt, although often at the expense of performance. Wang et al. [44] pioneered the analysis of MPC applied to transformer evaluation, inspiring subsequent research into secure transformer inference. MPCFormer [23] and SecFormer [24] leverage the CrypTen [19] MPC engine, introducing distillation processes where stronger models train weaker ones to enhance accuracy. However, this approach results in a loss of approximation accuracy, requiring fine-tuning of the models. Puma [9] employs the SecretFlow-SPU [25] MPC engine and, unlike MPCFormer, approximates the GeLU function using a piecewise polynomial approach that requires comparisons and polynomials of up to degree 6. Curl [39] also leverages CrypTen and avoids comparisons and polynomial approximations by implementing an optimized lookup table protocol. Sigma [16] applies function secret sharing techniques and small lookup table evaluations for secure inference. This approach demands significant computing and communication resources, such as 1 TB of RAM and a 9 Gbps communication link. Still, a large gap remains between plaintext evaluation speeds and secure computation. Whereas in plaintext, models with billions of parameters produce multiple tokens per second, it takes minutes to produce a single token under MPC. Fission reduces this gap by leveraging plaintext computations for nonlinear function evaluations, which are usually the bottlenecks for secure computation while maintaining privacy by shuffling and splitting the data between the evaluator nodes.

Obfuscation solutions. In parallel, other lines of research explore obfuscation techniques as a means of ensuring privacy. These methods aim to mitigate privacy risks while using specific metrics to evaluate their effectiveness. One of the earliest approaches to obfuscation is split learning, where the model training process is distributed across multiple entities [42]. However, this method has been proven vulnerable to various attacks [1], including membership inference attacks and the unintended memorization of unique individual data, particularly during fine-tuning [47]. To address these concerns, several works have proposed solutions to mitigate these privacy issues. SubMix [13] introduced differential privacy noise during the decoding phase of LLMs, reducing the likelihood of the model revealing whether specific data points were part of its training set. More recent studies have explored differential privacy-based obfuscation of user inputs during inference by introducing noise. For instance, Split-and-Denoise [27], inspired by split learning, involves clients running

embeddings locally, adding differential privacy noise to these embeddings, and then sending the noisy embeddings to a server. The server processes the model and returns the response, which the client then denoises. The mutual information measure is used to analyze potential data leakage in this process.

While obfuscation techniques aim to enhance privacy, their lack of rigorous security formalization can lead to unforeseen vulnerabilities. UnSplit [10], Pasquini et al. [32] and Qiu et al. [36] demonstrate attacks on dense neural networks trained with split learning. Chen et al. [7] have recently exposed critical weaknesses when combining split learning with LLM fine-tuning, even with differential privacy, questioning prior approaches. Also, Morris et al. [29] highlight how embeddings fail to sufficiently obscure input data.

In addition to split inference with differential privacy methods, alternative obfuscation techniques have been proposed to enhance privacy during inference. Xiang et al. [46] introduced a method that adds a random phase shift and performs dense neural network inference using complex values to conceal inputs. Yuan et al. [49] and PermLLM [50] propose permutation-based methods to mask data owners' inputs and outputs from the model server. These works claim that permuted hidden layers cannot be used to reverse engineer the input prompt and assess privacy using the distance correlation metric.

However, we devise an attack that show latent distribution can be used to extract relevant information about the inference sample. Fission overcomes this challenge by utilizing an MPC network and a separate evaluator network that only receives parts of permuted data. The intermediate values are revealed after they have been shuffled and partitioned; otherwise, they are kept in the MPC nodes as secret shares. Additionally, with Fission, the model weights are kept private throughout the evaluation. Fission significantly reduces the information leaked by the same attack compared to related works [17,50].

1.2 Our Contribution

Various PPML approaches have been proposed with different efficiency-privacy trade-offs. Cryptographic approaches fall short on execution time due to communication and computation overheads, while obfuscation techniques fall short on privacy. To demonstrate this, we devise an attack on split inference and PermLLM by training a binary classification model on the cutoff layer information of each framework. Our attack shows that we can extract information about the input from intermediate activation layers, rendering PermLLM and split inference solutions vulnerable. This attack can be of independent interest.

In this paper, we introduce Fission, a hybrid privacy-preserving compute framework that utilizes MPC nodes for linear operations as well as additional nodes that we call evaluators to speed up the computation of nonlinear functions. Contrary to PermLLM, Fission is resilient against our aforementioned attack, as each evaluator has insufficient information to successfully train a binary classifier. Fission combines ideas from both cryptographic and obfuscation solutions, running linear layers under MPC and then shuffling and splitting the data before revealing it to the evaluator nodes to compute the nonlinearities in plaintext. The evaluators then secret share the results back to the MPC nodes, who continue the computation until the next nonlinear layer. This results in more than 8× faster evaluation for BERT [8], 5× faster evaluation for ModernBERT [45], and more than 3× faster evaluation of LLama 3 1B [15] over CrypTen [19]. Our contributions can be summarized as follows:

- *Fission:* A novel secure inference framework comprising an MPC network and a separate evaluator network.
- Performance evaluations of Fission and related works on several LLMs with up to 8× latency improvements.
- Information leakage attack on previous split inference techniques, which are of independent interest.

2 Preliminaries

2.1 Secure Multiparty Computation

MPC enables multiple parties to compute a function over their private inputs while ensuring that no party learns anything beyond the final output [48]. The most widely used MPC technique is additive secret sharing,

which splits private values into shares distributed among non-colluding parties. Given a value x in a finite ring $\mathbb{Z}_Q$, its secret-shared representation $[\![x]\!]$ is formed by assigning each party $\mathcal{P}_i$ a share $[\![x]\!]_i$. The key property of additive secret sharing is that the sum of all shares reconstructs x as $\sum_{i=1}^{\mathsf{n}} [\![x]\!]_i \mod Q$. Since individual shares appear random and reveal no information about x, the scheme's security relies on ensuring that no subset of colluding parties can recover the secret unless all shares are combined.

Linear operations over secret shares are achieved with only local operations. We can add or multiply a secret share by a public value locally as $\alpha \cdot x = \sum_{i=1}^{\mathsf{n}} \alpha \cdot [\![x]\!]_i \mod Q$ and $x+\alpha = ([\![x]\!]_1+\alpha)+\sum_{i=2}^{\mathsf{n}} [\![x]\!]_i \mod Q$, respectively. Adding two secret-shared values, $[\![x]\!]$ and $[\![y]\!]$, is done locally by each party as $[\![z]\!]_i = [\![x]\!]_i + [\![y]\!]_i \mod Q$. This ensures that the sum of the shares correctly reconstructs the sum of the original values, $z = x+y$. To reveal a secret-shared value x, each party sends their share $[\![x]\!]_i$ to a designated party (or broadcasts it). The receiving party sums the shares to reconstruct x, ensuring privacy until the final step. Unlike addition, multiplication requires interaction and precomputed randomness. A common technique is Beaver triples [3], which involves randomness (a, b, c) such that $c = a \cdot b$. Given two secret-shared values $[\![x]\!]$ and $[\![y]\!]$, parties use these triples to compute $[\![x \cdot y]\!]$ without leaking any information. First, they reveal $[\![d]\!] = [\![x]\!] - [\![a]\!]$ and $[\![e]\!] = [\![y]\!] - [\![b]\!]$ and locally multiply d with $[\![b]\!]$ and $[\![a]\!]$ with e, while the first party also computes $d \cdot e$. They each add $[\![c]\!]$ to shares of $d \cdot b$ and shares of $a \cdot e$. The first party also adds to this $d \cdot e$. The result of these operations, $[\![c]\!] + [\![d \cdot b]\!] + [\![a \cdot e]\!] + d \cdot e$, leads to $[\![x \cdot y]\!]$

Since secret sharing operates over finite rings, MPC frameworks often use fixed-point arithmetic to represent floating-point numbers [39]. A real number x is approximated by scaling it to an integer representation as $\tilde{x} = \lfloor x \cdot 2^f \rceil$, where f is the precision. To retrieve the original value after computation, the output is rescaled by dividing by 2^f. This approach enables secure computation over continuous values while maintaining compatibility with integer-based MPC protocols.

2.2 Large Language Models Primer

Transformer-based LLMs are a specific type of neural network that utilize attention mechanisms to prioritize different segments of the input sequence when generating an output [15]. Similar to other neural architectures, LLMs begin with embedding layers that convert input tokens into dense vector representations. These include a lookup table that maps each token to a fixed-length vector, called word embeddings, and additional vectors that encode the position of each token in a sequence, called positional embeddings. Since embeddings rely on lookup tables, they can be mathematically represented as multiplications with one-hot vectors and then a summation to extract the value of interest.

A transformer block integrates multiple linear and nonlinear layers in a structured manner. It typically consists of:

- Self-attention to capture relationships in a sequence;
- Linear transformations to project the representation;
- Activation functions to introduce expressiveness;
- Normalization layers to stabilize training.

Stacking multiple transformer blocks enables deep learning models to capture increasingly complex dependencies and generate high-quality outputs. A fundamental component of transformers is self-attention, which enables the model to assign different levels of importance to different tokens within a sequence. The self-attention mechanism operates using a query matrix Q and a set of key-value matrices K and V, all of which are derived from learned linear transformations. The attention is computed as $\mathsf{softmax}(Q \cdot K^T/\sqrt{d}) \cdot V$, where d represents the dimensionality of the keys. In practice, when applied in a decoder, self-attention often includes a masking step to ensure that a token's prediction is influenced only by prior tokens, thereby enforcing causality in autoregressive models. From a computational perspective, transformer models consist of linear and nonlinear layers. Similarly, feedforward neural network layers consist of an inner product followed by an activation function, which introduces a nonlinearity essential for model expressiveness.

Nonlinear layers, such as rectified linear units (ReLU) [12], sigmoid linear units (SiLU), and softmax, pose additional computational challenges. These functions are not straightforward to evaluate efficiently,

particularly in secure computation settings. For instance, $\mathsf{ReLU}(x) = \mathtt{max}(0, x)$ can be computed via a secure comparison, whereas functions like SiLU and softmax require more complex approximations. Methods such as polynomial approximations, used in frameworks like CrypTen [19], enable secure evaluation at the cost of some precision loss. More advanced approaches, such as Sigma [16] and Curl [39] reduce approximation errors by leveraging ReLU alongside precomputed lookup tables. Unfortunately, as the models grow bigger, small approximation errors propagate and can significantly degrade the final accuracy. Additionally, normalization layers, such as layer normalization (LayerNorm) and root mean squared normalization (RMSNorm), introduce further computational complexity. These operations require nonlinear functions such as reciprocal and reciprocal square root calculations, which are slow to evaluate and lead to high inaccuracies. On the other hand, split inference runs these in the clear, so it is fast and accurate, but not secure. Fission hybrid approach overcomes both of these challenges by using ideas from both MPC and split inference.

3 Attacking Obfuscation Techniques

Split inference frameworks have been subject to scrutiny due to unclear information leakage during the reveal steps. Because the revealed data are highly informative, without proper refinement and reduction, they may be relevant to a party looking to extract specific characteristics [17,32,26]. On the other hand, PermLLM [50] uses MPC to hide input weights but reveals shuffled values before nonlinearities to a single party so it can evaluate these in plaintext. In this section, we present an attack on split inference frameworks as well as the first successful attack on PermLLM.

3.1 Attacking Split Inference

Split inference runs part of the model on the edge device and the rest on a server. To perform this split, a middle layer is chosen as the cutoff layer, which is sent over to the server [42,46]. Thus, the server can only learn from the information present in the cutoff layer. To construct an attack, we assume an attacker can freely use the system for inference and is also capable of obtaining the cutoff layer activations. Thus, the attacker knows the truth value of a sample and the intermediate activation of such a sample. We use this information and train a binary classification neural network on the cutoff layer information of the split inference framework. Our goal is to extract relevant information of the initial sample from the intermediate activation. We consider a binary classification task designed to predict whether the input text belongs to class `A` or class `B`. We use a balanced dataset with equal portions of each class and partitioned into training and test subsets. For each sample, we take the first N tokens and pass them through LLM, then we extract the output from an intermediate layer. These intermediate representations simulate the information that would be revealed to the server in a split inference setting. We use these representations as inputs to train a separate classifier to the original class label. This setup reflects the perspective of an adversarial attacker attempting to infer sensitive information. The experiment is repeated 100 times to report the standard deviation.

For our experiment setup, we use the UCI Machine Learning SMS Spam Collection Dataset [2] for the binary classification task, where we downsample to get a balanced dataset with 50% of the classes being spam. For each sample, we use the first 30 tokens of each sequence. Furthermore, to properly verify the claims, we do an 80-20% division of the dataset for training and test sets. We train for 100 epochs with a learning rate of 0.0001 and batch size 32. The training results show how we can predict the correct class from the cutoff information with $94.1 \pm 0.6\%$ test accuracy. Hence, a malicious server that only has access to an intermediate layer can predict the binary classification test with high accuracy, showing that there is a significant information leakage.

3.2 Attacking PermLLM

Unlike split inference, PermLLM [50] does not reveal layers as is. It uses random permutations to shuffle the values before revealing them to a single MPC node to compute the nonlinearities in plaintext. This is effective in reducing the information content revealed, however, it still leaks the whole distribution to the

computing party. Specifically, shuffling destroys the positional information contained in the particular hidden layer activations, however, the distribution of the values in the sample still remains. PermLLM claims that the permuted vector is almost irrelevant to the original vector in the statistical sense; however, we show in this attack that this is not the case.

To devise our attack on PermLLM, we use the same setup as split inference. However, since the layer values revealed to the node are permuted in this case, the attacker does not have the location information of the values. Therefore, it cannot input them into the neural network directly. Neural networks are sensitive to the ordering of the inputs, so shuffled data would be ineffective to learn from, but the attacker can sort the values to input orderly data into the model. Although the sorted values are not what the attacker initially got, nothing prevents them from doing this afterward.

We train a model based on those sorted values instead of the original layer values. We can still predict correctly with an accuracy of $68.7\% \pm 0.7\%$, meaning that the shuffled values leak a great deal of information about the underlying input as the attacker still has access to the distribution of the values. We confirm that this is better than random by training a model on random noise, which achieves only $50.4\% \pm 0.8\%$. PermLLM successfully reduces the information that an honest but curious attacker can obtain by using random permutation to break the location information in the data. The accuracy of our attack goes down by around 26%, however, there still remains another 18% gap between PermLLM and random noise, suggesting that there is room for improvement.

4 Accelerating Private Inference

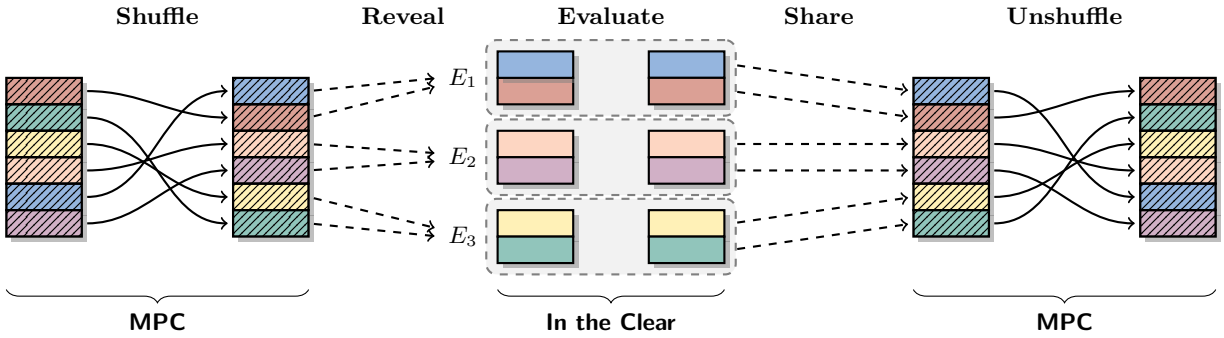

Fig. 1: To evaluate a non-linearity, the MPC parties shuffle the output of each matrix multiplication and reveal parts of the shuffled output to the evaluator parties $E_1, \ldots, E_N$ to compute the non-linearity in the clear. Next, all the outputs are secret shared between the MPC nodes that unshuffle to the correct order. The patterned boxes represent secret shares (owned by MPC parties), while the solid boxes represent values in the clear owned only by one evaluator.

In this section, we show how to modify the simple and insecure scheme of Section 3 to prevent any party from learning the data distribution of the nonlinear layers. To do so, we first delve into the core reason split inference is insecure for various models. Let's take as an example vision models; the first activation layers retain almost raw visual features allowing reconstruction of the input images, and splitting after the first few layers allows for a server to obtain "pixel-perfect copies" of the original images [18]. Similarly, embeddings in language models can be inverted to recover the original text [22]. It is clear that for such inversion attacks to take place, the attacker needs to have access to all the outputs of linear layers. Additionally, in most models (save for vision models), the attacker would also need to know the weights that were used in the linear layers to reverse the operations.

4.1 A Strawman Approach for Evaluating Nonlinearities

Our first contribution is to design a protocol that maintains the benefits of fast running time and high accuracy that split inference offers while removing the leakage of private inputs. Prior approaches shift the splitting part towards the middle and the end of the model, assuming that the information has been obfuscated enough so it cannot be leaked [42,46], but as we demonstrated in Section 3, that is insecure. Instead, our work takes a different approach. Our key improvement is to split the outputs of the nonlinear layers across multiple nodes so that each node alone has insufficient information to perform any meaningful attack. At the same time, each node can evaluate a nonlinearity over clear data, secret share the output, and continue by evaluating the linear layers under MPC. However, this solution still suffers from a similar attack as previous split inference approaches. Since each party receives some outputs of the linear layers, it can use the weights (if they are public or known to that party) to infer parts of the inputs.

To decrease the effectiveness of the attack, we introduce a random permutation before splitting the data into multiple parties to prevent any possible reconstruction attack. By randomly shuffling the data and then revealing parts of it to multiple parties, we can ensure that the shuffled revealed elements cannot be used for inferring anything about the inputs. We present an overview of our approach in Fig. 1. First, the parties evaluate the linear layers under MPC, then shuffle the data, and split-reveal to evaluate the nonlinear layers. By split-reveal, we refer to the technique of revealing different chunks of data to different parties. Finally, the parties secret share the outputs of the nonlinearities, then apply the inverse permutation.

So far, we have intentionally skipped over a crucial detail: the shuffling. For the parties to receive clear and permuted data securely, the permutation needs to be kept secret and performed under MPC; that is, no party knows how the data are permuted. Multiple works have focused on MPC shuffling targeting both semi-honest [6,30] and malicious security [20,41,21] models, as well as different combinations of the number of parties, with the most efficient works relying on replicated secret sharing with three parties. Although the approaches above perform oblivious shuffling under MPC, their end-to-end overhead is impractical for our setting. Most of these protocols require each party to shuffle, randomize, and send the data to the next party, a method that requires high communication. In the context of an LLM, performing MPC shuffling before each nonlinear layer would be unfeasible. Setting the end-to-end latency aside, there is another reason this methodology falls short. To the best of our knowledge, oblivious shuffling has been explored for a small number of parties. In our case, after the split-reveal technique, this would require each party to learn a significant portion of the output of the linear layers and would prevent our methodology from scaling to an arbitrary number of parties.

4.2 Fission for Evaluating Nonlinear Layers

We propose Fission,[3] a framework that leverages our split-reveal technique for distributed privacy-preserving LLM inference. In our setting, we assume that one party, the model owner, owns a proprietary model and that another party, i.e., a client, wants to perform private inference on that model. Note that the model owner should not learn anything about the inputs or outputs of the client, while the client should not learn anything about the proprietary model. Both parties can secret share their private inputs to an MPC cluster consisting of M nodes. The MPC nodes can then evaluate the proprietary model under MPC using the secret inputs and generate private output shares, which they can then return to the client.

Our initial approach from Section 4.1, which relied on shuffling under MPC, was constrained by the fact that the number of MPC nodes was relatively small for efficient shuffling. As a result, the split-reveal technique revealed a lot of information to each party. For instance, with $M = 3$, and an input tensor of 120 elements, each party would receive 40 random elements. Although related works [50,49] claim that as long as the inputs to the nonlinear functions are permuted, they are safe to reveal, our attacks from Section 3 prove otherwise. To address this limitation, Fission splits the computing nodes into two types of nodes: the evaluator nodes, which receive portions of plaintext shuffled arrays and compute nonlinear functions; and

[3] Fission occurs when a neutron slams into a larger atom, forcing it to excite and split into two smaller atoms, known as fission products.

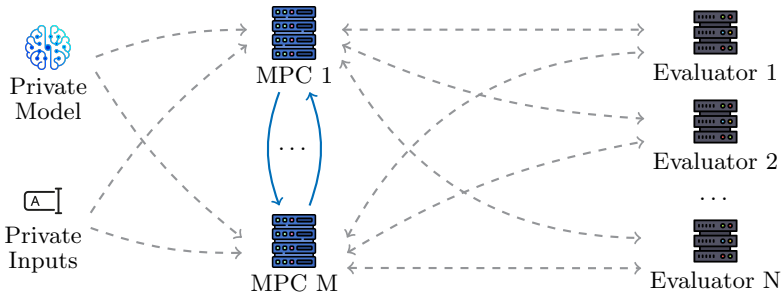

Fig. 2: Fission architecture overview consisting of an MPC cluster of M nodes and a cluster of N evaluators.

the MPC nodes, which run the underlying MPC protocol for the rest of the layers and perform oblivious shuffling. This allows having a smaller MPC network with efficient shuffling and a bigger evaluators network to increase privacy. We show an outline of the architecture of our Fission framework in Fig. 2, where on the left-hand side, both the inputs and the model are secret shared to the MPC parties, who in turn evaluate the linear layers and invoke the N evaluators for the nonlinearities. At one extreme, having only a single evaluator bears down to whether the permutation alone is sufficient to protect the privacy of the data. At another extreme, having as many evaluators as the size of the tensor fed into the nonlinear function significantly limits the amount of information that each evaluator receives. We discuss this trade-off in more detail in Section 4.4.

Fission technique can be applied to all the nonlinearities that LLMs and machine learning models entail. For general activation functions, Fission is not affected by the input tensor size and we flatten the whole tensor as we have to evaluate the nonlinearity over every point. Take a general tensor of size $[k][l][m]$, we flatten it into $[k \cdot l \cdot m]$, and split the flattened tensor across the N evaluators. We apply this technique in all activation functions like exponential, logarithm, reciprocal, square root, inverse square root, sigmoid, tanh, the error function, GeLU, SiLU, etc.

One function that needs careful consideration is softmax, as it operates over vectors and normalizes them into probabilities proportional to the exponentials of the input numbers. Recall that softmax is defined as $\sigma(v)_i = e^{v_i} / \sum_{j=1}^{K} e^{v_j}$, which requires all the elements of each vector. To accommodate for this, we keep the last dimension of each tensor, flatten all the previous dimensions, and reveal a vector. For instance, a tensor of size $[k][l][m]$ is flattened into $[k \cdot l][m]$ and we split the flattened tensor across the N evaluators across the $k \cdot l$ dimension. Depending on our privacy needs, we can evaluate the softmax function by breaking it down into exponential and reciprocal functions and evaluating them. For layer normalization, we need to calculate the mean and standard deviation and then normalize each value by subtracting the mean and dividing by the standard deviation. Similarly, we can either pass in the whole vector to evaluate the normalized vector. Or we calculate the mean and variance under MPC and pass in the variance to calculate the inverse square root, and then multiply the values by this inverse to get the normalization.

4.3 Linear Layers

As described in Section 2.1, we use Beaver triples to compute linear layers [3]. We start with matrix triples $A \in \mathbb{Z}_Q^{m \times k}$, $B \in \mathbb{Z}_Q^{k \times n}$, $C \in \mathbb{Z}_Q^{m \times n}$ such that $A \cdot B = C$, where k, m, n are positive integers defining the size of the matrices. These matrices are secret shared between the MPC nodes. During the online computation phase, MPC nodes use these triples to multiply secret shared matrices $X \in \mathbb{Z}_Q^{m \times k}$ and $Y \in \mathbb{Z}_Q^{k \times n}$, where A masks X and B masks Y.

For a given LLM and sequence length, we run through the LLM to calculate all the triples we are going to need. We use a separate party as a trusted third party to generate all these triples and secret share them among the MPC nodes. Then, each MPC node caches these triples in queues in a hash table using their sizes (k, m, n) as keys. During the online computation phase, the MPC nodes simply pop the triples needed from the queue with the needed size.

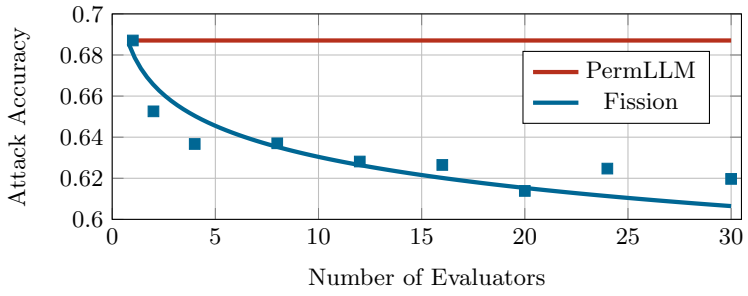

Fig. 3: The binary classification accuracy of the attack as a function of the number of evaluators compared to PermLLM.

Since we have to use integers due to secret sharing, we have to rely on fixed-point numbers instead of floating-point numbers. That means that when we multiply two numbers, we end up with a number that has double the precision, $\tilde{x} \cdot \tilde{y} = \lfloor x \cdot 2^f \rceil \cdot \lfloor y \cdot 2^f \rceil = \lfloor x \cdot y \cdot 2^{2f} \rceil$, where f is the fixed point precision parameter. Normally, in secure computation, a truncation operation is carried out by dividing the number by 2^f to reduce the precision back down to f. This means that we have to truncate after every multiplication; however, if the next step involves a nonlinearity that the evaluator nodes carry out in the clear, we don't have to. The evaluator nodes can simply turn the product that has precision $2f$ to a floating point number by dividing by 2^{2f}. Then, they can evaluate the nonlinear operation and turn the number into a fixed point number with precision f. This way, we save ourselves a lot of truncation operations, which increase latency since they require computation and communication.

4.4 Security Discussion

We carry out attacks from Section 3 on the Fission architecture. Recall that, Fission not only permutes the data, but it also splits the data into chunks for multiple evaluators. From this, we pick a random chunk and consider it as our input features. After this split-reveal, we train a model based on the sorted values that any single evaluator sees. For a growing number of evaluators, we see a downtrend in the accuracy obtained, meaning we are decreasing the information leaked by adding more evaluators. This is displayed in Fig. 3. Here, we observe that there is a decrease in accuracy as the number of evaluators increases. We already see a drop to 0.65 from 0.69 when there are two evaluators, and the decline continues. The points lie on a curve $y = 0.68 - 0.02 \cdot \log(x)$ where y is the attack accuracy and x is the number of evaluators.

One way to reduce the information leakage further is to split the data into chunks, shuffle the data within each chunk, and randomly send these chunks to different nodes to evaluate the nonlinear functions. This way, each node only sees part of the distribution, and the information leakage is reduced.

Fission significantly reduces the information leakage as compared to other obfuscation techniques, however, distribution information could still be present if the parameters of the models are not carefully chosen. In such scenarios, it may be feasible for an attacker to extract distributions even from the reduced data. To fully remove information multiple queries are batched together in a single request and then shuffled altogether. Thus, the distributions of the individual queries are hidden by each other. Without access to all the original query values, the attacker cannot determine the positional or distribution information of each query, thereby gaining no meaningful information. When combined with Fission, batching offers strong resilience against attacks effectively eliminating information leakage.

In Fission, we assume honest but curious nodes that do not collude and share data with each other. If these assumptions fail and two evaluators collude, this would equate to reducing the number of evaluators. If the MPC nodes collude with each other, they would not learn anything unless all the MPC nodes colluded. Lastly, if an MPC nodes colludes with an evaluator node, they would know the shuffling and part of the revealed data, so the amount of information they could obtain would depend on the ratio of linear equations to the unknowns.

Table 1: Accuracy of Fission for privacy-preserving inference versus the PyTorch accuracy over clear data.

Model	Variant	Parameters	Dataset	Accuracy		
				PyTorch	Fission	Diff
BERT [8]	Tiny	14M	QNLI	0.815	0.815	0.02%
			SST-2	0.832	0.832	0.00%
	Base	110M	QNLI	0.909	0.909	0.04%
			SST-2	0.923	0.917	0.62%
	Large	340M	QNLI	0.918	0.915	0.38%
			SST-2	0.925	0.924	0.12%
Modern BERT [45]	Base	149M	QNLI	0.931	0.930	0.20%
			SST-2	0.944	0.934	1.09%
GPT	GPT-2 [37]	137M	Lambada	0.355	0.354	0.05%
	Neo [4]	1.3B	Lambada	0.519	0.519	0.04%

Fission can be naturally extended to a malicious setting, where MPC nodes can run under malicious setting. For the evaluators, the MPC nodes can embed random values into the activation function evaluation. These values can later be revealed and verified to ensure correctness and fairness, serving as a defense against adversarial behavior from evaluators.

In practice, it is infeasible to run many evaluators as coordinating many nodes becomes impractical. Here, we can leverage private cloud solutions to increase the number of nodes. By dividing a single node with a trusted execution environment [38] instance to multiple nodes, we can have as many evaluators in a single node as there are CPU cores. The correctness of execution can be checked with attestations without seeing the computation or the data itself. This way all the evaluators can be run in their secure enclaves without revealing any data, even to each other. Furthermore, evaluators can attest to the correctness of their executions.

5 Experimental Evaluations

Our goal in experiments is to show that while keeping privacy 1) Fission has high accuracy comparable to PyTorch, 2) Fission runs end-to-end inference on LLMs with 1B parameters in a matter of seconds, 3) Fission runs significantly faster than prior works, 4) As the number of evaluators increase, Fission becomes more efficient.

5.1 Implementation & Experimental Setup

Implementation. We build Fission on top of CrypTen [19], which exposes MPC primitives via common machine learning abstractions such as tensors and modular neural networks. Fission natively supports both CPU and GPU backends and the communication between them.

We create a client-server setup between each MPC node and each evaluator node. Each time a nonlinearity has to be evaluated, the MPC nodes send requests to the evaluator servers, which wait to receive the data from all MPC nodes to reconstruct the data. Then, each evaluator node computes the nonlinear function on the received data. The evaluator servers then secret share the data back to the MPC nodes.

Experimental Setup. For our experiments, we used cloud instances with 80 virtual CPUs, together with two H100 GPUs running on Ubuntu 24.04. We ran each of the MPC nodes on different GPUs, while each of the evaluator nodes ran on different CPU cores. Our code is available at `https://github.com/jimouris/curl`.

Table 2: Latency (seconds) and communication (GB) evaluations of Fission for privacy-preserving LLM inference (with embeddings).

Model	Variant	Sequence Length	Fission		
			Latency (sec)	Com. (GB)	
				MPC	Eval.
BERT [8]	Tiny	64	0.58	0.083	0.033
		128	0.61	0.102	0.067
		256	0.69	0.140	0.136
	Base	64	2.10	1.156	0.098
		128	2.21	1.268	0.215
		256	2.67	1.518	0.506
	Large	64	4.28	3.160	0.208
		128	4.80	3.429	0.467
		256	5.98	4.042	1.136
Modern BERT [45]	Base	64	3.06	1.448	0.078
		128	3.46	1.720	0.192
		256	5.00	2.318	0.522
	Large	64	4.99	3.193	0.141
		128	5.49	3.644	0.332
		256	7.49	4.623	0.866
GPT	GPT-2 [37]	64	2.29	1.409	0.118
		128	2.78	1.530	0.255
		256	3.17	1.801	0.586
	Neo [4]	64	13.66	11.793	0.381
		128	13.22	12.325	0.812
		256	14.87	13.540	1.825
Llama 3 [15]	1B	64	15.89	12.977	0.168
		128	16.10	14.034	0.403
		256	19.56	16.348	1.074

5.2 Accuracy Experiments

CrypTen [19] relies on polynomial approximations while Curl [39] uses lookup tables that are approximated by discrete wavelet transforms. These inherently introduce approximation errors, which degrade the performance of the models. In contrast, Fission has no approximation errors due to nonlinearities, as it evaluates nonlinear layers in the clear. Note that we still have to approximate floating-point numbers using a fixed-point representation due to secret sharing. We use 64-bit integers with 16 bits of precision to represent the floating point numbers, which, from empirical observations, is enough to preserve accuracy and minimize approximation errors.

We ran accuracy tests with various LLMs; from variants of BERT and ModernBERT on QNLI [43] and SST-2 [40] classification tasks to GPT-2 and GPT Neo 1.3B on the Lambada [31] next-word prediction task. We report these values in Table 1. We observe that the accuracy numbers are similar to the same plaintext models in PyTorch, meaning we have no accuracy loss. We can use the same plaintext models in secure computation as is, hence, we do not need any expensive knowledge distillation operation nor any alterations to nonlinearities that change the model [23,24].

Table 3: Latency (seconds) and communication (GB) comparisons between Fission and CrypTen for privacy-preserving LLM inference. This table ignores embeddings as CrypTen does not support them.

Model	Variant	Sequence Length	Fission			CrypTen	
			Latency (sec)	Com. (GB)		Latency (sec)	Com. (GB)
				MPC	Eval.		
BERT [8]	Tiny	64	0.43	0.006	0.002	1.12	0.057
		128	0.43	0.008	0.004	1.24	0.139
		256	0.45	0.014	0.011	1.4	0.361
	Base	64	1.6	0.765	0.067	6.68	2.623
		128	1.77	0.860	0.152	9.91	5.537
		256	2.04	1.078	0.380	17.86	13.527
	Large	64	3.71	2.643	0.177	15.71	7.598
		128	4.16	2.896	0.405	23.42	15.363
		256	5.12	3.475	1.010	42.23	36.666
Modern BERT [45]	Base	64	2.56	1.111	0.078	10.92	2.908
		128	3.03	1.358	0.190	13.64	6.710
		256	4.40	1.902	0.519	24.65	18.280
	Large	64	4.1	2.753	0.140	15.01	6.355
		128	4.62	3.178	0.330	19.18	12.941
		256	6.97	4.103	0.862	36.16	31.883
GPT	GPT-2 [37]	64	1.50	0.765	0.066	6.73	2.622
		128	1.69	0.859	0.151	9.01	5.533
		256	2.09	1.076	0.377	15.36	13.520
	Neo [4]	64	11.42	10.117	0.327	25.55	18.902
		128	10.44	10.621	0.705	36.03	30.764
		256	13.53	11.778	1.611	64.09	60.312
Llama 3 [15]	1B	64	10.31	8.707	0.168	18.34	13.746
		128	11.50	9.698	0.403	28.25	23.174
		256	14.01	11.875	1.071	44.99	49.843

5.3 Execution Time Experiments

We report the latency and communication between the MPC nodes and the evaluator nodes. We focus on the inference times of the variants of BERT, ModernBERT, GPT, and Llama with sequence lengths of 64, 128, and 256, following previous works [16,39]. We report the end-to-end running times, including the embedding layer for Fission in Table 2, showing we can evaluate the full Llama 3 1B model in under 20 seconds with less than 20 GB of communication and the full ModernBERT model in under 5 seconds with less than 3 GB of communication.

We compare our results with CrypTen as our baseline without an embedding layer since it does not support embeddings. We report these numbers in Table 3. We observe improvements of $3 - 8\times$ in running times and up to $12\times$ in communication bandwidth. We observe the best improvements for BERT-Base/Large and GPT-2 models as these models have a higher nonlinear operation to linear operation ratio, and the worst performance for Llama as this model has a larger number of linear operations. The gap between Fission and CrypTen grows as the sequence length increases, as Fission gets increasingly more performant with longer sequence lengths due to the softmax operation that greatly slows down full MPC solutions. Furthermore, we compare with the state-of-the-art Sigma and observe that we are faster for larger sequence length sizes. We were not able to replicate their experiments due to memory issues.

Table 4: Llama (sequence length 64), varying number of evaluators (no embeddings).

Evaluators	2	4	8	16	32
Latency (s)	10.32	10.30	9.44	9.40	10.43

Finally, in Table 4, we demonstrate that an increase in the number of evaluators decreases the total runtime by evaluating the Llama 3 1B model with a sequence length of 64 for a varying number of evaluators. By increasing the number of evaluators, each evaluator has to process fewer elements, which results in faster running times. The ideal number of evaluators for this model seems to be around 16. Increasing the number of evaluators beyond 16 gives diminishing returns in computation time while the protocol starts experiencing a significant communication overhead from the evaluators.

6 Concluding Remarks

While obfuscation techniques offer a compelling approach to model evaluation, they suffer from critical privacy shortcomings. Conversely, cryptographic solutions provide strong security guarantees, often at the expense of a significant computational overhead. To reconcile these tradeoffs, we present Fission, a framework that evaluates linear layers under MPC and nonlinear layers in the clear using a separate evaluator network. Fission's hybrid design enables private, highly accurate, and performant LLM inference.

We evaluated Fission on multiple LLMs, including BERT, ModernBERT, GPT, and Llama 3, achieving up to 8× performance speedups over prior privacy-preserving methods. Beyond performance, Fission prioritizes usability by exposing a tensor-based programming model with either CPU or GPU backends, making it accessible for machine learning researchers and developers familiar with common machine learning frameworks such as PyTorch. Finally, we introduced a novel attack on obfuscation techniques revealing information leakage. These findings are of independent interest and underscore the need for secure and fast alternatives, such as Fission, in privacy-sensitive machine learning applications.

Acknowledgments

The authors would like to thank João Ribeiro for the fruitful discussions.

References

1. Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit Ahmet Çamtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. Can we use split learning on 1D CNN models for privacy preserving training? In Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ASIACCS 20: 15th ACM Symposium on Information, Computer and Communications Security*, pages 305–318, Taipei, Taiwan, October 5–9, 2020. ACM Press.
2. Tiago A Almeida, José María G Hidalgo, and Akebo Yamakami. Contributions to the study of sms spam filtering: new collection and results. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 259–262, 2011.
3. 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 Berlin Heidelberg, Germany.
4. Sid Black et al. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021.
5. Tom Brown et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
6. Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 342–372, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

7. Guanzhong Chen et al. Unveiling the vulnerability of private fine-tuning in split-based frameworks for large language models: A bidirectionally enhanced attack. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 2904–2918, New York, NY, USA, 2024. Association for Computing Machinery.
8. Jacob Devlin et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, June 2019.
9. Ye Dong et al. Puma: Secure inference of llama-7b in five minutes. *arXiv:2307.12533*, 2023.
10. Ege Erdoğan, Alptekin Küpçü, and A Ercüment Çiçek. Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, pages 115–124, 2022.
11. Lars Wolfgang Folkerts and Nektarios Georgios Tsoutsos. Testing robustness of homomorphically encrypted split model llms. *Cryptology ePrint Archive*, 2024.
12. Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3):121–136, 1975.
13. Antonio Ginart, Laurens van der Maaten, James Zou, and Chuan Guo. Submix: Practical private prediction for large-scale language models. *arXiv:2201.00971*, 2022.
14. Charles Gouert et al. Ripple: Accelerating programmable bootstraps for fhe with wavelet approximations. In *Information Security*, pages 273–293, Cham, 2025. Springer Nature Switzerland.
15. Aaron Grattafiori et al. The Llama 3 Herd of Models. *arXiv:2407.21783*, 2024.
16. Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT inference with function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2024(4):61–79, October 2024.
17. Zecheng He, Tianwei Zhang, and Ruby B Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, 2019.
18. Zecheng He, Tianwei Zhang, and Ruby B Lee. Attacking and protecting data privacy in edge–cloud collaborative inference systems. *IEEE Internet of Things Journal*, 8(12):9706–9716, 2020.
19. Brian Knott et al. Crypten: Secure multi-party computation meets machine learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021.
20. Nishat Koti et al. Ruffle: Rapid 3-party shuffle protocols. *Proceedings on Privacy Enhancing Technologies*, 2023(3):24—-42, April 2023.
21. Nishat Koti et al. Graphiti: Secure graph computation made more scalable. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 4017–4031, New York, NY, USA, 2024. Association for Computing Machinery.
22. Chenxi Li et al. Unveiling the unseen: Exploring whitebox membership inference through the lens of explainability. *arXiv:2407.01306*, 2024.
23. Dacheng Li et al. MPCFormer: Fast, performant and private transformer inference with MPC. In *International Conference on Learning Representations (ICLR)*, pages 1–16, 2023.
24. Jinglong Luo et al. Secformer: Fast and accurate privacy-preserving inference for transformer models via smpc. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 13333–13348, 2024.
25. Junming Ma et al. 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.
26. Kiwan Maeng et al. Bounding the invertibility of privacy-preserving instance encoding using fisher information. *Advances in Neural Information Processing Systems*, 36:51904–51925, 2023.
27. Peihua Mai et al. Split-and-denoise: protect large language model inference with local differential privacy. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.
28. Sarah Mercer, Samuel Spillard, and Daniel P Martin. Brief analysis of deepseek r1 and it's implications for generative ai. *arXiv:2502.02523*, 2025.
29. John X Morris et al. Text embeddings reveal (almost) as much as text. *arXiv:2310.06816*, 2023.
30. Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Buddhavarapu, and Benjamin M. Case. Delegated private matching for compute. *Proceedings on Privacy Enhancing Technologies*, 2024(2):49–72, April 2024.
31. Denis Paperno et al. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv:1606.06031*, 2016.
32. Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. In *CCS 2021*, pages 2113–2129, 2021.

33. Yifan Peng, Qingyu Chen, and George Shih. Deepseek is open-access and the next ai disrupter for radiology, 2025.
34. Charith Peris et al. 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.
35. Elvira Pollina. Deepseek blocked on apple and google app stores in italy. *Reuters*, 2025.
36. Xinchi Qiu et al. Evaluating privacy leakage in split learning. *arXiv:2305.12997*, 2023.
37. Alec Radford et al. Language Models are Unsupervised Multitask Learners, 2019.
38. Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa*, volume 1, pages 57–64. IEEE, 2015.
39. Manuel B. Santos et al. Curl: Private LLMs through Wavelet-Encoded Look-Up Tables. In *Conference on Applied Machine Learning in Information Security (CAMLIS) 2024*, 2024.
40. Richard Socher et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.
41. Xiangfu Song, Dong Yin, Jianli Bai, Changyu Dong, and Ee-Chien Chang. Secret-shared shuffle with malicious security. In *ISOC Network and Distributed System Security Symposium – NDSS 2024*, San Diego, CA, USA, February 26 – March 1, 2024. The Internet Society.
42. Praneeth Vepakomma et al. Split learning for health: Distributed deep learning without sharing raw patient data, 2018.
43. Alex Wang et al. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv:1804.07461*, 2018.
44. Yongqin Wang et al. Characterization of mpc-based private inference for transformer-based models. In *ISPASS*, pages 187–197, 2022.
45. Benjamin Warner et al. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference, 2024.
46. Liyao Xiang et al. Interpretable complex-valued neural networks for privacy protection. *arXiv:1901.09546*, 2019.
47. Biwei Yan et al. On protecting the data privacy of large language models (llms): A survey, 2024.
48. 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.
49. Mu Yuan, Lan Zhang, and Xiang-Yang Li. Secure transformer inference protocol, 2023.
50. Fei Zheng et al. PermLLM: Private Inference of Large Language Models within 3 Seconds under WAN. *arXiv:2405.18744*, 2024.