

SECURE OUTLIER-AWARE LARGE LANGUAGE MODEL INFERENCE

Lifan Zhao

Shanghai Qi Zhi Institute
zhaolf@sqz.ac.cn

Zhixuan Fang*

Tsinghua University
Shanghai Qi Zhi Institute
zfang@mail.tsinghua.edu.cn

ABSTRACT

Secure multiparty computation allows the client to secretly inference their sensitive inputs without acquiring the proprietary machine learning model weights. As the decoder-only transformer-based large language model becomes the popular paradigm, the desire of applying MPC in large language models is increasing. However, such inference usually leads to great amount of latency, which is due to nonlinear operations in the Transformer architecture. Recent works either focus on improving cryptographic primitives or re-architecting and re-training to make LLM MPC-friendly. We, on the other hand, observe that properly addressing outlier phenomena, which are unique yet universal properties existing across different LLMs, can effectively reduce the input domain and thereby design faster protocols for non-linear operations. Hence, we propose **Secure Outlier-Aware Large Language Model Inference** framework (SOAL), which accelerates the RMSNorm operation by nearly $2\times$, SiLU by $2\times$, and Softmax by more than $5\times$. SOAL maintains similar performance as the original model without any fine-tuning requirement.

1 INTRODUCTION

As large language models (Radford et al., 2019; Gupta et al., 2022; Jiang et al., 2023) have demonstrated remarkable capability in variety of aspects, the concern of leakage of private sensitive information to cloud-based LLM inference services is increasingly raised. On one hand, precious business data, and sensitive personal data are processed by model providers’ servers without any privacy protection. On the other hand, LLM model weights, which are the product of trillions of tokens and millions of GPU-hours of training, are uploaded and stored on third-party servers, counting on their pledge. Secure multiparty computation (MPC) offers a solution where LLM inference services can be offered without leaking any information to either users or model providers. However, making the LLM inference with an MPC protocol is notably slow. Inferencing a 64-token input once using Llama2-7B (Gupta et al., 2022) via CrypTen (Knott et al., 2021) would take 169.76 seconds, while it only takes less than one second in plain text. Such latency becomes even greater as the input sequences gets longer, i.e., it would take more than 428.95 seconds for a 512-token input. Such inefficiency stems from preprocessing-based MPC frameworks’ difficulty in handling nonlinear layers, including Softmax, Activation (SiLU or GeLU), and Normalization (Layernorm or RMSNorm).

The inherent complexity of non-linear protocols comes from the disproportionate output growth across a large input domain. Meanwhile, activations during LLM inference may vary by many orders of magnitude. For example, activations for normalization layers in Llama2-7B (Gupta et al., 2022) vary from 10^{-5} to 10^3 , as shown in Figure 5b. For non-linear protocols covering such a wide input domain, directly using a lookup table (LUT) protocol would require a LUT with 32-bitwidth. As LUT protocols usually have a complexity of $O(2^n)$, it is very expensive to compute. Other methods, like Goldschmidt’s method (Goldschmidt, 1964) would similarly consequent in more iterations. Observing such flaws, previous works (Mohassel & Zhang, 2017; Cheng et al., 2023; Luo et al., 2024) have tried to substitute non-linear operations with MPC-friendly ones. SecureML (Mohassel

*Corresponding author.

& Zhang, 2017) used $\text{Softmax}(x) = \frac{\text{ReLU}(x)}{\sum \text{ReLU}(x_i)}$ to avoid exponential computation. MPCFormer (Cheng et al., 2023) substituted $\text{GeLU}(x)$ with $\text{Quad} = 0.125x^2 + 0.25x + 0.5$. Although they exploited the robustness of LLMs such that the model may still be functional after substitution, these methods face substantial accuracy degradation and require additional fine-tuning to compensate for the quality discrepancy. Such methods modify the original model, induce training overhead, and bring doubt regarding the validity and quality of the new model.

Motivated by above challenges, we ask the following research question:

Can we achieve efficient non-linear protocols specifically designed for MPC LLM inference?

Meanwhile, another line of recent research offers us a critical insight: that LLMs, especially decoder-only transformer-based LLMs, have many unique properties shared in common, which have been utilized in a variety of domains. Dettmers et al. (Dettmers et al., 2022) observed that most activations and model weights have a skewed distribution, which means their values are concentrated in a narrow range and only very few activations are significantly large. By handling outliers separately, they succeeded in making inference with FP8, a one-byte data format for floating point number, rather than FP32. This method effectively lowers the requirement for GPU memory without sacrificing much performance. Inspired by their observation, we study non-linear layers and find that similar outlier distribution phenomenon also persists. Figure 1 demonstrates the layer-wise activation distribution for normalization layers, where most activations aggregate around a certain value with small variances. Meanwhile, there are a few outlier activations that deviate far from the majority. It is the existence of such outliers that increases the complexity of MPC non-linear protocols.

Our approach and contributions. We propose **Secure Outlier-Aware Large Language Model Inference framework (SOAL)**, a two-stage framework which addresses the unique outlier features in the decoder-only transformer-based LLMs and optimizes non-linear protocols without compromising LLM performance. In detail, our contribution is threefold:

1. We discover different outlier phenomena existing in activations for non-linear layers across various LLMs. We further address such phenomena by prefixing special tokens to users’ input prompt to manipulate the occurrence of outliers. Our evaluation shows that such modification brings no performance degradation.
2. We design new protocols for normalization and activation layers, which benefit from the control of outliers. Based on a collateral feature from prefixing tokens, we improve the Softmax protocol as well. Our protocols can work with different cryptographic schemes.
3. Our SOAL achieves nearly $2\times$ speedup when inferencing Llama2-7B and GPT-2 with 512 tokens under the additive secret sharing scheme, while maintaining the accuracy of the output. SOAL also obtain efficiency improvement on both online time and total keysize under the function secret sharing scheme.

2 RELATED WORK

2.1 PRIVACY-PRESERVING MACHINE LEARNING.

MPC originates from the Billionaire problem (Yao, 1986; Shamir, 1979) whose goal is to enable multiple untrusted parties to jointly compute a function while keeping the inputs private. As LLMs show their incredible ability, recent works focus on mitigating MPC with LLM inference only. They primarily follow two complementary directions. The first focuses on accelerating secure inference through advanced cryptographic primitives. For instance, systems such as SIGMA (Gupta et al., 2024) and BOLT (Chen et al., 2024b) leverage function secret sharing (FSS) and vector oblivious linear evaluation (VOLE), respectively, to reduce communication and latency. Others, like Iron (Hao et al., 2022), PrivFormer (Dalskov et al., 2023), and PUMA (Dong et al., 2025), optimize secure matrix multiplications and nonlinear layers using a mix of secret sharing, homomorphic encryption, and polynomial approximations, demonstrating practical performance on full-scale transformer models. Innovations like BumbleBee (Huang et al., 2025), SHAFT (Kei & Chow, 2025), and NEXUS (Zhang et al., 2025) further improve scalability by introducing protocol-level optimizations for core transformer components, including attention, GELU, and Softmax. In parallel, a second thread of research modifies transformer architectures to better suit MPC. Works such as

MPCFormer (Cheng et al., 2023) and SecFormer (Luo et al., 2024), based on additive secret sharing (ASS) scheme, substitute costly nonlinearities (e.g., Softmax and GELU) with MPC-friendly alternatives like low-degree polynomials and train these altered models using knowledge distillation, achieving significant efficiency gains with minimal accuracy loss. However, the requirement of knowledge distillation may change the model weights, thereby hindering its broader usage.

2.2 OUTLIERS IN LLM ACTIVATION

In this work, we specifically focus on decoder-only transformer-based large language models (Radford et al., 2019; Gupta et al., 2022; Jiang et al., 2023). During LLM inference, the activations for each layer, no matter what input prompt is, always contain a few activations, known as outliers, that stray far from others. On the other side of the coin, the rest of the activations exhibit a skewed pattern concentrated in a narrow range. Outliers in LLM inference have drawn much attention recently (Bondarenko et al., 2021; Wei et al., 2022; Puccetti et al., 2022; Dettmers et al., 2022; Xiao et al., 2023b; Sun et al., 2024). Some of these activations are related to particular features, which are known as outlier features. Puccetti et al. (Puccetti et al., 2022) showed empirically that the occurrence of outlier features is related to the frequency of tokens in the training distribution. Dettmers et al. (Dettmers et al., 2022) showed that the scale of an LLM is related to the emergent properties of these outlier features, and utilized such features to effectively quantize model weights. Other activations may jointly pertain to specific channels and tokens. Sun et al. (Sun et al., 2024) introduced the concept of “massive activations” — rare but extremely large activation values that appear consistently in special tokens (e.g., at begin of sentence (*bos*) or newline tokens). Chen et al. (Chen et al., 2024a) further utilize such a relationship to propose a prefixing method and make INT4 quantization comparable to other dynamic per-token quantization methods. Besides linear layers’ activations, (Xiao et al., 2023a;b; Bondarenko et al., 2023) reveal the attention sink phenomenon, where those related to the *bos* token tend to be larger. In this work, we extend the study to non-linear layers and reveal that such outliers not only persist, but also can be used to optimize MPC protocols.

3 OUTLIERS IN NON-LINEAR LAYERS

In this section, we extend the LLM outlier study from the linear layer to non-linear operations in transformer-based LLMs, SoftMax, Normalization and Activation. Given an input activation for non-linear operations, outliers are a small group of values which greatly deviate from the average. Outliers require a larger input domain for non-linear MPC protocols, which consequently need either more bits to handle or more iterations to approximate.

We use RedPajama (Weber et al., 2024), a dataset released after the chosen LLMs, to run 128 inferences with different token lengths and collect activations for different non-linear operations. We conduct additional experiments using more LLMs and different datasets in Appendix A. We identify different outlier phenomena corresponding to different non-linear layers. Such outlier phenomena exist across various datasets, showing that these are ubiquitous features belonging to the decoder-only transformer-based LLMs.

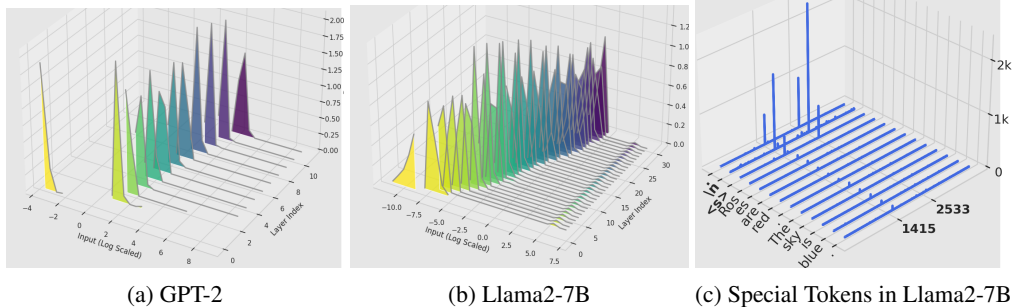


Figure 1: The activations distribution for normalization operation in (a) GPT-2 and (b) Llama2-7B. We use bold line for better visualizing the deviation from majority values. (c) Special tokens are closely related to the outliers.

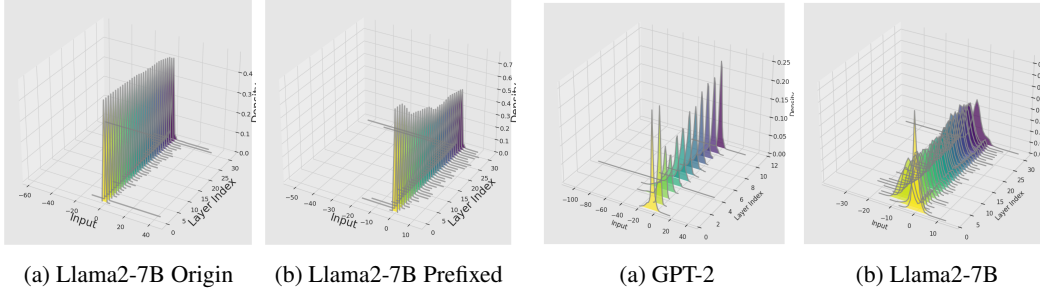


Figure 2: Activation Distribution for SiLU.

Figure 3: Activation Distribution for SoftMax.

Normalization We visualize the distribution of activations for each normalization layer in Figure 1. The long tails in most layers indicate that there exists a small group of activations whose values greatly depart from the majority. It is the existence of these outliers which greatly extend the domain requirement for reciprocal square root protocol design. Fortunately, the occurrence of these outliers shows a strong connection with special tokens. As shown in Figure 1c, in Llama2-7B, outliers locates at the positions where the token “.” (period), “\n” (newline) and “<s>” (begin of sentence (BOS)) appear. Such observations align with the outlier findings in linear layers from previous works (Sun et al., 2024; Xiao et al., 2023a). We will describe the detail method to extract special tokens and address these *Special-Token Outlier* later in Section 4.1.1.

Activation We display the input distribution for Activation layer in Figure 2a. Values aggregate and form spikes around zero where the activation function is most sensitive, while outliers appear in the first and last few layers. We additionally show that outliers in activation layers are also affected by the aforementioned special tokens in Figure 2b.

SoftMax We further explore the layerwise activations distribution for SoftMax in Figure 3. The majority of activation values concentrate around their layerwise average values, which also matches the observation in FlashDecoding++(Hong et al., 2024) that $> 99.99\%$ x_i are within a certain range. We will later demonstrate that the outliers can be effectively captured by *Conformant Maxima*—the local maxima we discovered, which are derived from values gathered from predefined locations within attention logits.

In conclusion, these different yet consistent observations on non-linear layers point toward an answer to the question posed earlier, that if we can address these outliers properly, the skewed distribution of the rest activations can be utilized to design better protocols for these non-linear operations.

4 METHOD

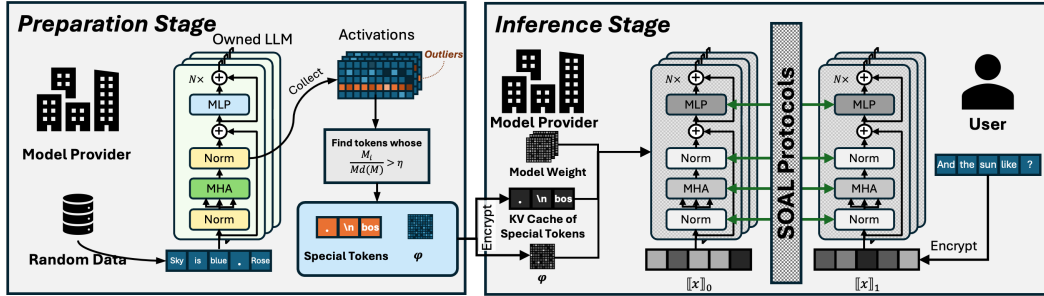


Figure 4: SOAL Framework. In preparation stage, model providers collect activations by performing extra inferences with their LLMs. They then obtain special tokens and auxiliary parameters φ from these activations. In the inference stage, model provider and user follow our new protocols to efficiently perform MPC LLM inference.

SOAL focuses on the standard two-party computation (2PC) with one trusted dealer (Beaver, 1991; Bendlin et al., 2011; Damgård & Zakarias, 2013) that has received significant attention in privacy-

preserving LLM inference (Gupta et al., 2022; Knott et al., 2021; Jawalkar et al., 2024; Luo et al., 2024; Kei & Chow, 2025). That is, there are two semi-honest (honest but curious) parties, P_0 as the user who has the input prompt x_0 and P_1 as the model provider who offers the inference service with the model weights x_1 . In the offline phase, a trusted dealer offers correlated randomness to accelerate online inference. Together, the 2PC want to perform the computation of a public function $y = F(x_0, x_1)$ without revealing anything more than the function output y to each other. Detailed preliminaries about MPC and LLM are provided in Appendix B.

In general, SOAL is a two-stage framework as shown in Figure 4. In the preparation stage, as detailed in Section 4.1, the model provider utilizes previous observations on outliers in non-linear layers and prepares model-specific auxiliary parameters used for secure inference. In the inference stage, which we describe fully in Section 4.2, the model provider and the user follow the new designed non-linear protocols to perform faster secure LLM inferences.

4.1 PREPARATION STAGE: ADDRESS OUTLIER

4.1.1 SPECIAL-TOKEN OUTLIER

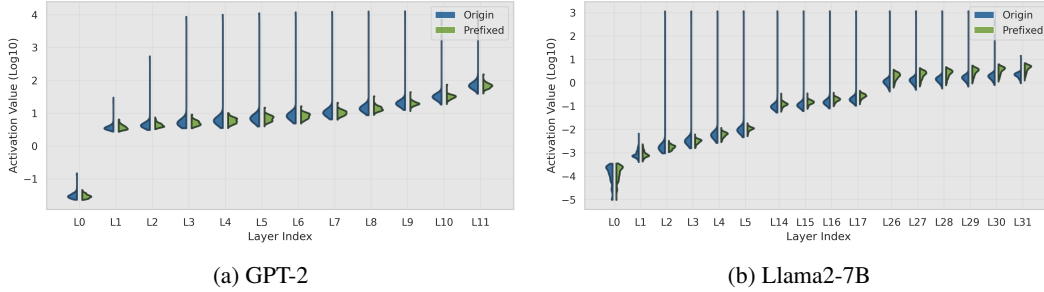


Figure 5: Comparison of layer-wise normalization activation distribution in violin plot. Blue half represents for original distribution, and green part is the distribution after prefixing special tokens.

As mentioned above, the outliers in normalization layers not only exist, but also occur along with special tokens. To identify the special tokens, we use the activations for pre-layer normalization and acquire the special-token outliers as follow:

We collect activations $X \in \mathbb{R}^{N \times L \times F}$ through N inference for each layer. We then statistic token-wise maximum value, $M \in \mathbb{R}^T$, where T is the tokenizer size. For i -th token whose ratio of its maximum value over the median maximum value of all other tokens exceeds a threshold η , such as $\frac{M_i}{\text{median}(M)} > \eta$, we record the corresponding token as a special token. We set $\eta = 8$ here since we want to shrink the input domain as much as possible.

Now, with the special tokens at hand, we can address these outliers in normalization layers. Since the outliers appear where these special tokens first emerge, we prefix these tokens ahead of user’s input prompts to manipulate such outliers appearing only at the prefixed positions. To visualize the effect, we inference twice with the same prompt, except that in the second time, we prefix these special tokens ahead of that prompt. As shown in Figure 5, after prefixing the special tokens, the green parts no longer have long-tails in any layers, which means that the outliers have been limited within those prefixed positions. What’s more, due to LLM’s autoregressive property, these special tokens can be stored in the format of KV-cache (Radford et al., 2019) to avoid redundant computation, and use as the model-specific auxiliary parameters during inference stage. Compared to the overhead of fine-tuning the entire model for different tasks, such a token-identifying process can be easily done by the model provider offline within a few minutes.

4.1.2 CONFORMANT MAXIMA

Outliers in SoftMax conform with a pattern that we can utilize. We observe that the maxima of softmax input present a connection with special tokens. Given the collected softmax inputs $x \in \mathbb{R}^{N \times L \times L}$ from N inferences of L -token length prompts, we present a heatmap with the shape $L \times L$ in Figure 6. Each grid therefore represents for one position in x . We then use the grayscale to

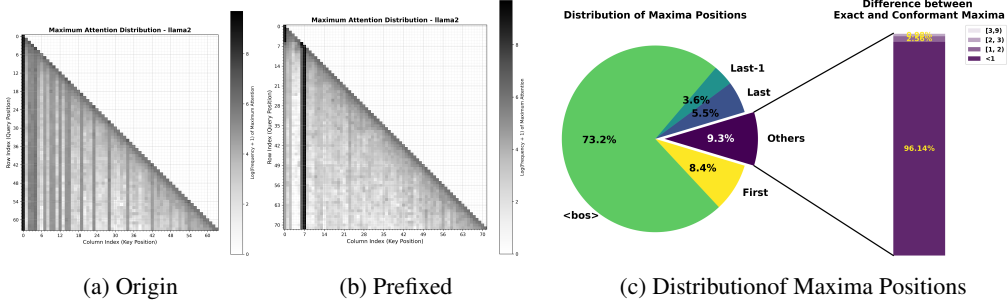


Figure 6: (a) Heatmap of origin maxima positions of Llama2-7B. Grayscale means the times maxima appear at that position. (b) Heatmap with prefixed special tokens. (c) Pie chart of the occurrence of exact maxima. For rest of maxima locating in other positions, conformant maxima are close to them.

Algorithm 1: Secret Reciprocal Square Root, Π_{rsqrt}

Input: For $\sigma \in \{0, 1\}$, P_σ holds the shares

$$\llbracket x \rrbracket_\sigma$$

Output: For $\sigma \in \{0, 1\}$, P_σ holds the shares $\llbracket y \rrbracket_\sigma$ where $y = \text{RMSNorm}(x)$.

```

1  $\llbracket x' \rrbracket_\sigma := \llbracket x \rrbracket_\sigma - 1$ 
2  $\llbracket x'_2 \rrbracket_\sigma := \Pi_{\text{square}}(\llbracket x' \rrbracket_\sigma)$ 
3  $\llbracket x'_3 \rrbracket_\sigma := \Pi_{\text{mul}}(\llbracket x' \rrbracket_\sigma, \llbracket x'_2 \rrbracket_\sigma)$ 
4  $\llbracket y \rrbracket_\sigma := a + b\llbracket x' \rrbracket_\sigma + c\llbracket x'_2 \rrbracket_\sigma + d\llbracket x'_3 \rrbracket_\sigma$ 
5 foreach  $iter < 2$  do
6    $\llbracket y \rrbracket_\sigma := \llbracket y \rrbracket_\sigma \cdot (3 - \llbracket x' \rrbracket_\sigma \cdot \llbracket y' \rrbracket_\sigma^2) / 2$ 
7 return  $\llbracket y \rrbracket_\sigma$ ;
```

Algorithm 2: Secret RMSNorm, Π_{RMSNorm}

Input: For $\sigma \in \{0, 1\}$, P_σ holds the input share $\llbracket x \rrbracket_\sigma$, and scale share $\llbracket \varphi_i \rrbracket_\sigma$

Output: For $\sigma \in \{0, 1\}$, P_σ holds the shares $\llbracket y \rrbracket_\sigma$ where $y = \text{RMSNorm}(x)$.

```

1  $\llbracket x' \rrbracket_\sigma := \Pi_{\text{mul}}(\llbracket x \rrbracket_\sigma, \llbracket \varphi_i \rrbracket_\sigma)$ 
2  $\llbracket x' \rrbracket_\sigma := \Pi_{\text{square}}(\llbracket x' \rrbracket_\sigma)$ 
3  $\llbracket d \rrbracket_\sigma := \sum(\llbracket x' \rrbracket_\sigma) + \epsilon$ 
4  $\llbracket d' \rrbracket_\sigma := \Pi_{\text{rsqrt}}(\llbracket d \rrbracket_\sigma)$ 
5  $\llbracket y \rrbracket_\sigma := \Pi_{\text{mul}}(\llbracket d' \rrbracket_\sigma, \llbracket x' \rrbracket_\sigma)$ 
6 return  $\llbracket y \rrbracket_\sigma$ ;
```

represent the number of times maxima locate at that position. Without prefixing, maxima may appear where special tokens first locate; therefore, there are some random vertical lines in Figure 6(a). Meanwhile, with special tokens prefixed, the maxima concentrate at the position of “<bos>” tokens and the first one and last two tokens of each row. Such phenomenon has been identified as the attention sink, by StreamLLM (Xiao et al., 2023b). They pointed out that certain token may receive more attention than others due to the occurrence of these tokens in the training datasets. Such phenomenon gives us a potential way to simplify maximum finding operation in SoftMax.

We record the exact maximum values $x_{\max} \in \mathbb{R}^{N \times H \times L}$ of softmax inputs from N different inferences. We then collect the activations at (1) where the *bos* token locates, (2) the first token position, and (3) the last two positions of each row. The local maxima are selected from these activations, and we call them *Conformant Maxima*. As shown in Figure 6c, over 90% exact maxima’s positions fall in these predefined positions. What’s more, for those maxima mismatched, the differences between the exact maxima and our conformant maxima are also small. Hence, by prefixing special tokens and collecting conformant maxima, we can avoid the $O(n \log n)$ computation to find the exact maximum over an n -token length input.

4.2 INFERENCE STAGE: OPTIMIZE MPC PROTOCOLS

The bond between addressing outliers and optimizing MPC protocols lies in input domain reduction. In this subsection, we demonstrate our new non-linear protocols under ASS scheme. The protocols under FSS scheme will be introduced in Appendix E.

4.2.1 NORMALIZATION

Normalization is used between multi-head attention layers and feed-forward layers to keep activations from exploding. For the input $X \in \mathbb{R}^{L \times F}$, where F is the feature dimension, RMSNorm

Algorithm 3: Secret Sigmoid for LLM, Π_{sigmoid}

Input: For $\sigma \in \{0, 1\}$, P_σ holds the input share $\llbracket x \rrbracket_\sigma$, and $\llbracket \nu_0 \rrbracket_\sigma, \llbracket \nu_1 \rrbracket_\sigma$.
Output: For $\sigma \in \{0, 1\}$, P_σ holds the shares $\llbracket y \rrbracket_\sigma$ where $y = \sigma(x)$.

```

1  $\llbracket x \rrbracket_\sigma := -\log_2 e \cdot \llbracket x \rrbracket_\sigma$ 
2  $\llbracket x_i \rrbracket_\sigma := \Pi_{\text{trunc}}(x'), \llbracket x_f \rrbracket_\sigma := \llbracket x \rrbracket_\sigma - \llbracket x_i \rrbracket_\sigma$ 
3  $\llbracket \nu_0 \rrbracket_0 = \llbracket \nu_0 \rrbracket_0 + 2^{x_{f0}}, \llbracket \nu_1 \rrbracket_1 = \llbracket \nu_1 \rrbracket_1 + 2^{x_{f1}}$ 
4  $\llbracket v_2 \rrbracket_\sigma := \Pi_{\text{mul}}(\llbracket \nu_0 \rrbracket_\sigma, \llbracket \nu_1 \rrbracket_\sigma)$ 
5  $\llbracket v_1 \rrbracket_\sigma, \llbracket \bar{v}_1 \rrbracket_\sigma = \Pi_{\text{LUT}}(\llbracket x_i \rrbracket_\sigma)$ 
6  $\llbracket v_2 \rrbracket_\sigma := \Pi_{\text{mul}}(\llbracket v_2 \rrbracket_\sigma, \llbracket \bar{v}_1 \rrbracket_\sigma)$ 
7  $\llbracket \delta \rrbracket_\sigma := v_1 + v_2$ 
8  $\llbracket \alpha \rrbracket_\sigma := \llbracket e - f\delta + g\delta^2 \rrbracket_\sigma$ 
9  $\llbracket r \rrbracket_\sigma := 2 \cdot \llbracket \alpha \rrbracket_\sigma - \Pi_{\text{mul}}(\llbracket \delta \rrbracket_\sigma, \Pi_{\text{square}}(\llbracket \alpha \rrbracket_\sigma))$ 
10  $\llbracket y \rrbracket_\sigma := \Pi_{\text{mul}}(\llbracket r \rrbracket_\sigma, \llbracket v_1 \rrbracket_\sigma)$ 
11 return  $\llbracket y \rrbracket_\sigma$ ;
```

Algorithm 4: Secret Softmax in LLM, Π_{softmax}

Input: For $\sigma \in \{0, 1\}$, P_σ holds the shares $\llbracket x \rrbracket_\sigma$.
Output: For $\sigma \in \{0, 1\}$, P_σ holds the shares $\llbracket y \rrbracket_\sigma$ where $y = \text{softmax}(x)$.

```

1  $\llbracket \tau \rrbracket_\sigma = \Pi_{\text{max}}(\llbracket x_{[b]} \rrbracket_\sigma, \llbracket x_{[i-1]} \rrbracket_\sigma, \llbracket x_{[i]} \rrbracket_\sigma, \llbracket x_{[0]} \rrbracket_\sigma)$ 
2  $\llbracket x' \rrbracket_\sigma := \llbracket x \rrbracket_\sigma - \llbracket \tau \rrbracket_\sigma$ 
3  $\llbracket e \rrbracket_\sigma := \Pi_{\text{exp}}(\llbracket x' \rrbracket_\sigma)$ 
4  $\llbracket d \rrbracket_\sigma := \sum \llbracket e \rrbracket_\sigma$ 
5  $\llbracket m \rrbracket_\sigma, \llbracket \bar{m} \rrbracket_\sigma := \Pi_{\text{msb}}(d)$ 
6  $\llbracket d' \rrbracket_\sigma = \Pi_{\text{mul}}(\llbracket d \rrbracket_\sigma, \llbracket m \rrbracket_\sigma)$ 
7  $\llbracket d' \rrbracket_\sigma := \Pi_{\text{NR}}(\llbracket d' \rrbracket_\sigma)$ , where the initial guess is 1.45 - 0.5  $\llbracket d' \rrbracket_\sigma$ . // Two NR iterations.
8  $\llbracket y \rrbracket_\sigma := \Pi_{\text{mul}}(\llbracket e \rrbracket_\sigma, \Pi_{\text{mul}}(\llbracket d' \rrbracket_\sigma, \llbracket \bar{m} \rrbracket_\sigma))$ 
9 return  $\llbracket y \rrbracket_\sigma$ ;
```

of each row $\mathbf{x}$ is calculated with $\gamma \cdot \frac{\mathbf{x}}{\sqrt{\frac{1}{F} \sum_{j=1}^F (x_j)^2 + \epsilon}} + \beta$, where γ, β are fixed parameters. The complexity lies in the reciprocal square root function which need multiple iterations to approximate.

We demonstrate our new protocol based on an approximation algorithm in Algorithm 2. As shown in Figure 5, the activations concentrate closely around their layerwise centers after the special-token outliers have been constrained in the prefixed token of KV-cache. Such concentration centers shift according to specific layers, from a scale of 10^{-5} to 10^1 . To further shrink the input domain, we scale the activations with layerwise secret values $\llbracket \varphi \rrbracket_\sigma$, and such scaling can later be canceled out via the division. Thus, we change the $\text{RMSNorm}(\mathbf{x})$ to $\gamma \cdot \frac{\varphi_i \cdot \mathbf{x}}{\sqrt{\frac{1}{F} \sum_{j=1}^F (\varphi_i \cdot x_j)^2 + \epsilon}} + \beta$. Now as the

input domain for the reciprocal square root has been reduced, we can use a third-degree polynomial centering at $x = 1$ to get the initial guess. The coefficients a, b, c, d are acquired by BFGS (Broyden, 1970) with MSE loss, and are set to $a = 0.913389, b = -0.860195, c = 1.028723, d = -0.359165$ in this work. With the proper initial guess, we can use only two Newton-Raphson iterations to obtain the precise result.

4.2.2 ACTIVATION

Activation is designed to bring non-linearity to LLM models. For the input $X \in \mathbb{R}^{L \times F}$, the SiLU function is calculated with $\text{SiLU}(x) = x \cdot \sigma(x)$, where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

As mentioned above, the outliers of SiLU inputs exhibit a connection to the special tokens. As shown in Figure 2b, with prefixed special tokens, the range of input domain becomes acceptable for efficient MPC protocol design. Again, the calculation of prefixed outliers can be omitted by the KV-cache. We now reformulate the sigmoid as $\sigma(x) = \frac{2^{-x_i \cdot (1-\xi)}}{2^{-x_i \cdot (1-\xi)} + 2^{x_f \cdot 2^{x_i \cdot \xi}}}$, where x_i, x_f represent the integer and fraction part of x , and the sign $\xi = 1\{x < 0\}$ is 1 if x is less than zero, otherwise 0. Hence, we implement the sigmoid protocol in Algorithm 3.

We first convert the exponential base to 2 and use local truncation protocol to get fraction $\llbracket x_f \rrbracket_\sigma$ and integer parts $\llbracket x_i \rrbracket_\sigma$. As the fraction part is small, each party can use the owned share $x_{f\sigma}$ to calculate a local exponential result $2^{x_{f\sigma}}$ which can be further re-shared secretly with the two predefined zero secret share $\llbracket \nu_0 \rrbracket_\sigma$ and $\llbracket \nu_1 \rrbracket_\sigma$. With one multiplication, we obtain the $\llbracket 2^{x_f} \rrbracket_\sigma$. Now, we can further simply the equation as $\frac{v_1}{v_1 + v_2 \cdot \bar{v}_1}$, where $v_1 = 2^{-x_i \cdot (1-\xi)}$, $\bar{v}_1 = 2^{x_i \cdot \xi}$ and $v_2 = 2^{x_f}$. As $\llbracket x_i \rrbracket_\sigma$ is an integer with small input domain, we can use LUT protocol to get v_1 and $\bar{v}_1$, both of which depend on the integer part. Then, in Lines 6-7, we obtain the denominator by adding v_1 and $v_2 \cdot \bar{v}_1$ together, whose range lies between (1, 2.5). Given the denominator's domain is small, we use a second-degree polynomial approximation to get a decent initial guess of the reciprocal in Line 7, where $e = 1.830181, f = 1.078845, g = 0.205469$. At last, we need one more round of Newton-Raphson iteration and multiply v_1 back to get the sigmoid result.

4.2.3 SOFTMAX

Softmax is the key element in Transformer structure. Given $X \in \mathbb{R}^{L \times L}$ where L is the token length, the softmax of $\mathbf{x} = \{x_0, x_1, \dots, x_i\}$, the i -th row of X , is calculated with $\text{softmax}(\mathbf{x}) =$

Algorithm 5: Secret Exponential, Π_{exp}

Input: For $\sigma \in \{0, 1\}$, P_σ holds the shares $\llbracket x \rrbracket_\sigma$, and $\llbracket \nu_0 \rrbracket_\sigma, \llbracket \nu_1 \rrbracket_\sigma$.

Output: For $\sigma \in \{0, 1\}$, P_σ holds the shares $\llbracket y \rrbracket_\sigma$ where $y = \exp(x)$.

- 1 $\llbracket x' \rrbracket_\sigma := \log_2 e \cdot \llbracket x \rrbracket_\sigma$
- 2 $\llbracket x_i \rrbracket_\sigma := \Pi_{\text{trunc}}(\llbracket x' \rrbracket_\sigma, s)$,
 $\llbracket x_f \rrbracket_\sigma := \llbracket x' \rrbracket_\sigma - \llbracket x_i \rrbracket_\sigma$
- 3 $\llbracket \nu_0 \rrbracket_0 := \llbracket \nu_0 \rrbracket_0 + 2^{x_{f_0}}$, $\llbracket \nu_1 \rrbracket_1 := \llbracket \nu_1 \rrbracket_1 + 2^{x_{f_1}}$
- 4 $\llbracket m \rrbracket_\sigma := \Pi_{\text{Mul}}(\llbracket \nu_0 \rrbracket_\sigma, \llbracket \nu_1 \rrbracket_\sigma)$
- 5 $\llbracket e \rrbracket_\sigma := \Pi_{\text{LUT}}(\llbracket x_i \rrbracket_\sigma)$
- 6 $\llbracket y \rrbracket_\sigma := \Pi_{\text{Mul}}(\llbracket m \rrbracket_\sigma, \llbracket e \rrbracket_\sigma)$
- 7 **return** $\llbracket y \rrbracket_\sigma$;

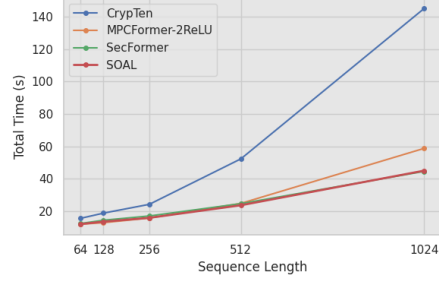


Figure 7: GPT-2 Inference Speed Comparison

Model	Method	SoftMax		Normalization		Activation		Total	
		Time (s)	Comm.(GB)	Time (s)	Comm.(GB)	Time (s)	Comm.(GB)	Time (s)	Comm.(GB)
GPT-2	CrypTen	30.79	52.54	3.07	0.88	10.89	19.68	48.86	76.79
	SOAL	6.80	4.52	2.99	1.18	9.17	15.26	23.30	24.64
Llama2-7B	CrypTen	199.99	375.03	27.86	12.25	87.30	169.31	428.95	702.44
	SOAL	26.62	32.13	14.87	16.01	37.13	67.19	193.59	261.15
Mixtral 8x7B	CrypTen	242.98	376.01	65.86	24.31	264.52	441.00	1104.46	1611.20
	SOAL	39.66	32.45	31.25	32.25	104.44	161.11	668.23	984.50

Table 1: Time and communication cost for secure inference of 512-token prompt. Average of 5 runs.

$\frac{\exp(\mathbf{x} - \max(\mathbf{x}))}{\sum_{j=1}^n \exp(x_j - \max(\mathbf{x}))}$. Among the three obstacles in calculating Softmax under MPC setting — maximum, exponential and reciprocal — finding the maximum is the most unique one, as its communication cost grows $O(n \log n)$ log-linearly according to the actual input length n .

The common practice of subtracting the exact maxima is to avoid numerical exploding problem caused by backpropagation during training, but, the maxima can mathematically be replaced by any arbitrary number τ during inference. As mentioned in Section 4.1.2, conformant maxima matches exact maxima for most of the time, meanwhile the difference is small when mismatched. It is proper candidate for τ to avoid the log-linear complexity of exact maximum finding. Hence, our softmax protocol can now handle any length of input, always with fixed communication cost and rounds.

We present our protocol for Softmax in Algorithm 4. Noticing that the number of reciprocal operation is much smaller than that of exponential, we directly get the most significant bits of the denominators. Therefore, by multiplying m with denominator d , we scale the denominator to a small domain where the reciprocal can be obtained with two NR iterations.

As our conformant maxima no longer promises the non-positive property of the SoftMax input, we display a new exponential protocol for the secret exponential function that can handle both positive and negative values in Algorithm 5. We first local truncate the $\llbracket x \rrbracket_\sigma$ by precision bits plus extra s bits to get $\llbracket x_i \rrbracket_\sigma$ and $\llbracket x_f \rrbracket_\sigma$, where $x = x_i \cdot 2^{f+s} + x_f$. Due to the product of powers rule, each party can locally calculate the exponential result of their share x_f at hand. Afterward, they can secretly reshare the result $2^{x_{f_0}}$ to $\llbracket \nu_0 \rrbracket_\sigma$ or $\llbracket \nu_1 \rrbracket_\sigma$. With one more multiplication, we can get correct result of 2^{x_f} . Then, with the help of extra s bits, we can use a smaller look up table to cover wider input domains. Additionally, since we have known the remaining positive inputs are fairly small, as shown in Figure 6c, we can shift the input range of the look up table to cover more negative input domain. Finally, by multiplying them together, we achieve the exponential results.

5 EXPERIMENT

5.1 EXPERIMENT SETUP

Setup. We choose LLMs including GPT-2 (Radford et al., 2019), Llama2-7B (Touvron et al., 2023), Mistral 8x7B (Jiang et al., 2023), all of which are decoder-only transformer-based language models. We include Mistral 8x7B to demonstrate SOAL framework can work on LLMs with different structures, i.e. Mixture of Expert. Note that, although BERT is commonly used in previous works, it is an encoder-decoder transformer that does not present the phenomena mentioned above.

Tokens	Time (s)		Comm(GB)		Key Size (GB)	
	Sigma	SOAL	Sigma	SOAL	Sigma	SOAL
64	1.097	0.852	0.370	0.36	6.806	5.938
128	1.936	1.493	0.824	0.783	14.292	11.915
256	4.166	2.996	1.983	1.819	33.191	25.873
512	10.048	7.577	5.302	4.648	86.686	61.830
1024	25.136	18.486	15.949	13.341	256.449	165.093

Table 2: Comparison of time, communication cost and size of generated key for Sigma and SOAL on GPT-2.

	GPT2		Llama2-7B		Mixtral 8x7B	
	Origin	SOAL	Origin	SOAL	Origin	SOAL
WikiText-2	36.25	37.50	5.55	5.58	6.412	6.547
C4	33.75	34.75	6.126	6.233	3.841	3.930

Table 3: Perplexity on C4 and WikiText2 datasets across different models.

	Origin	SOAL
Arc Challenge $\uparrow$	0.4334	0.4343
Arc Easy $\uparrow$	0.7635	0.7618
HellaSwag $\uparrow$	0.5713	0.5730
PIQA $\uparrow$	0.7807	0.7769
Winograde $\uparrow$	0.6938	0.6993
PPL(WikiText) $\downarrow$	5.55	5.58

Table 4: Evaluation of Llama2-7B performance using SOAL framework.

As the goal of SOAL is to show the effectiveness of integrating these ubiquitous phenomena of LLM inference with MPC protocols, we employ ASS and FSS as foundational schemes. For ASS scheme, we mainly use CrypTen (Knott et al., 2021) and adopt truncation protocols from (Santos et al., 2024)¹. We also include MPCFormer (Cheng et al., 2023) and SecFormer (Luo et al., 2024) with their codes yet changing the truncation method as well. For FSS scheme, we use the Sigma (Gupta et al., 2024) protocol. Our experiments are conducted on two Nvidia A100 GPU with 10GB/s bandwidth. In the experiment, we use [869, 29901, 22550, 5167, 2073, 29928, 13, 1] for Llama2-7B, [17405, 628, 198, 50256] for GPT-2, and [28705, 13, 262, 1] for Mixtral 8x7B.

5.2 EFFICIENCY OPTIMIZATION

To evaluate the efficiency of SOAL, we record the time and communication cost to inference one token given a 512-token prompt in Table 1 on different LLMs. As we can see, in the original CrypTen, the cost of non-linear protocols dominates the total cost. With the help of SOAL, the improvement in the non-linear operations is significant. We gain about $2\times$ speedup in SiLU/GeLU operations. We achieve more than $3\times$ faster in SoftMax operation as SOAL can find conformant maxima within a constant round. For normalization, although CrypTen only sets 3 Newton-Raphson iterations for its reciprocal square root protocol, and 8 iterations of Taylor Expansion for the exponential used for the initial guess, the corresponding input domain only covers (0.1, 145.6) with $< 0.1\%$ error, whereas the actual input domain always exceeds such a narrow range. Hence, we follow CrypTen’s suggestion, modify the protocol’s default configuration to get precise result where we use 0.02 as the initial guess and use 11 Newton-Raphson iterations. On the other hand, our SOAL framework eliminates the outliers and uses the provided scaling secret $\llbracket \varphi \rrbracket_\sigma$ to control the input domain. Therefore, we only need two iterations. In total, SOAL shows nearly $2\times$ efficiency than the original CrypTen protocol across different LLMs.

As the non-linear overhead increases quadratically when the input length gets longer, we conduct the experiment to show the input length’s impact on LLM inference in Figure 7. As one can see, due to the benefit of softmax, the longer the input sequence is, the more time can be saved by SOAL. Meanwhile, our method achieves similar speedup as SecFormer (Luo et al., 2024), and becomes faster than MPCFormer (Cheng et al., 2023) when making inference with longer prompt. It is worth noting that we do not require any fine-tuning effort.

SOAL also works on other cryptographic primitives. Table 2 shows SOAL’s inference result with Sigma. Although SOAL saves minor cost on the communication, it largely reduces the key size, which need to be transferred for each inference. We as well improve the online inference time.

5.3 PERFORMANCE EVALUATION

We use Huggingface’s Evaluation library to evaluate SOAL (Llama2-7B) with different benchmarks, including Arc Challenge, Arc Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), Winograde (Sakaguchi et al., 2021). We also use PPL on C4 (Raffel et al., 2020) and

¹The origin implementation of CrypTen uses local truncation, which has a secure flaw pointed out by (Li et al., 2023).

WikiText-2 (Merity et al., 2017) to evaluate other models. Even though MPCFormer and SecFormer fine-tune their models for downstream tasks, their method primarily have 1-2% performance degradation reported in their papers. We don’t include their results as they only finetuned the BERT, and GLUE benchmark (Wang et al., 2018) is designed for encoder-decoder LLM.

The performance remains almost the same as the original Llama2-7B, since SOAL only slightly changes the user input instead of changing the entire model weight. Additionally, perplexities over different LLMs are provided in Table 3. The perplexities are close to the origin result, showing that our framework has the robustness to work with LLMs of different structures.

6 CONCLUSION

We present SOAL, an efficient MPC framework optimized for decoder-only LLMs. We offer a new direction to improve MPC LLM inference, which involves utilizing LLM’s unique properties. We extend the study of outliers, a well-known phenomenon in linear layer, to non-linear layer. SOAL addresses these outlier phenomena and reduces the corresponding input domain. Therefore, we design new non-linear protocols based on it and gain nearly $2\times$ speedup given long inputs on different LLMs. We believe there exist more unique features in LLM, waiting to be explored and exploited.

ACKNOWLEDGMENTS

This work is supported by Shanghai Qi Zhi Institute Innovation Program. The work of Zhixuan Fang is also supported by Tsinghua University Dushi Program.

REFERENCES

- Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual international cryptology conference*, pp. 420–432. Springer, 1991.
- Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 169–188. Springer, 2011.
- Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.
- Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7947–7969, 2021.
- Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Quantizable transformers: Removing outliers by helping attention heads do nothing. *Advances in Neural Information Processing Systems*, 36:75067–75096, 2023.
- Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 337–367. Springer, 2015.
- Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1292–1303, 2016.
- Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pp. 341–371. Springer, 2019.
- Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.
- Mengzhao Chen, Yi Liu, Jiahao Wang, Yi Bin, Wenqi Shao, and Ping Luo. Prefixquant: Static quantization beats dynamic through prefixed outliers in llms. *arXiv preprint arXiv:2410.05265*, 2024a.

- Yihan Chen, Zhijian Luo, and Yuzhe Zhang. Bolt: Scalable private inference for transformer models. In *IEEE Symposium on Security and Privacy (S&P)*, 2024b.
- Yao Cheng, Sameer Wagh, et al. Mpcformer: Fast privacy-preserving inference for transformers. In *International Conference on Learning Representations (ICLR)*, 2023.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- Anders Dalskov, Luca Kamm, and Marcel Keller. Privformer: Privacy-preserving transformer inference via 3-party computation. In *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2023.
- Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, pp. 621–641, 2013.
- Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.
- Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, Wen-Guang Chen, et al. Puma: Secure inference of llama-7b in five minutes. *Security and Safety*, 4:2025014, 2025.
- Robert E Goldschmidt. *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. Llama: A low latency math library for secure inference. *Proceedings on Privacy Enhancing Technologies*, 4:274–294, 2022.
- 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*, 4:615–629, 2024.
- Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. *Advances in neural information processing systems*, 35:15718–15731, 2022.
- Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL <https://arxiv.org/abs/1606.08415>.
- Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhang Dong, and Yu Wang. Flashdecoding++: Faster large language model inference with asynchronization, flat gemm optimization, and heuristics. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 148–161, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/5321b1dabed2be188d796c21b733e8c7-Paper-Conference.pdf.
- Ziqi Huang, Thomas Smith, and Vladimir Kolesnikov. Bumblebee: Efficient and scalable mpc for large transformers. In *Network and Distributed System Security Symposium (NDSS)*, 2025.
- Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: Fss-based secure training and inference with gpus. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 597–616. IEEE, 2024.

- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. Mistral 7b, 2023. URL <https://arxiv.org/abs/2310.06825>.
- Andes Y. L. Kei and Sherman S. M. Chow. Shaft: Secure, handy, accurate, and fast transformer inference. In *Network and Distributed System Security Symposium (NDSS)*, 2025.
- Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.
- 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)*, pp. 5377–5394, 2023.
- Jinglong Luo, Yehong Zhang, Zhuo Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. Secformer: Fast and accurate privacy-preserving inference for transformer models via smpc. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13333–13348, 2024.
- Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.
- Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pp. 19–38. IEEE, 2017.
- Giovanni Puccetti, Anna Rogers, Aleksandr Drozd, and Felice Dell’Orletta. Outlier dimensions that disrupt transformers are driven by frequency. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1286–1304, 2022.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.
- Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure rnn inference. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1003–1020. IEEE, 2021.
- Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications, 2025. URL <https://arxiv.org/abs/2402.07927>.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.
- Manuel B Santos, Dimitris Mouris, Mehmet Ugurbil, Stanislaw Jarecki, Jos   Reis, Shubho Sengupta, and Miguel de Vega. Curl: Private llms through wavelet-encoded look-up tables. *Cryptology ePrint Archive*, 2024.
- Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- Noam Shazeer. Glu variants improve transformer, 2020. URL <https://arxiv.org/abs/2002.05202>.
- Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. Grotto: Screaming fast (2+ 1)-pc or z2n via (2, 2)-dpfs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2143–2157, 2023.
- Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. Massive activations in large language models. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=F7aAhfitX6>.

- Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- 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 *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pp. 353–355, 2018.
- Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A {GPU} platform for secure computation. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 827–844, 2022.
- Maurice Weber, Dan Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, et al. Redpajama: an open dataset for training large language models. *Advances in neural information processing systems*, 37:116462–116492, 2024.
- Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems*, 35:17402–17414, 2022.
- Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023a.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2023b.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.
- Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pp. 162–167. IEEE, 1986.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.
- Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. In *Network and Distributed System Security Symposium (NDSS)*, 2025.

APPENDIX

A MORE OBSERVATIONS

A.1 MORE LLMs AND DATASETS RESULTS

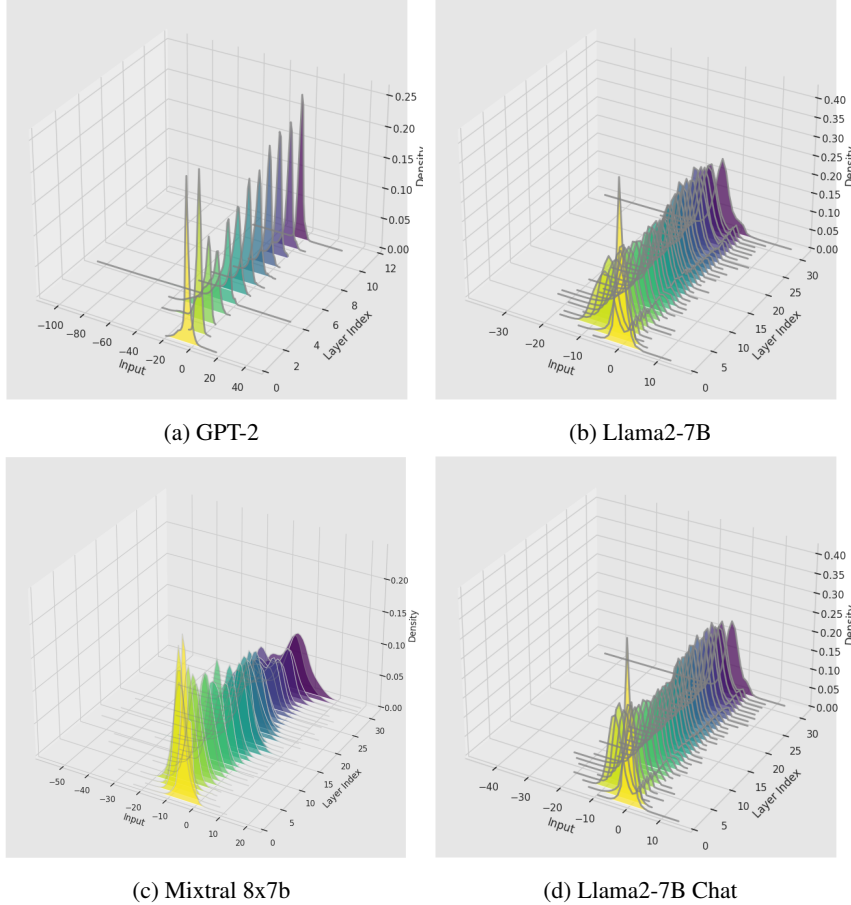


Figure 8: Histogram of activations for SoftMax in different LLMs. The extended line on x-y plane is for better visualize the location of maximum and minimum values.

In Figure 8, we present more softmax activation’s histograms for each layer. Even though they all are decoder-only transformer-based model, they do have some slight structure difference between each other. Comparing to GPT-2, Llama2-7B uses RMSNorm instead of LayerNorm, SiLU instead of GeLU, and add gate projection layer to MLP layer. Llama2-7B Chat is a fine-tuned version of Llama2-7B. Mixtral 8x7B is different to Llama2-7B in that its use a Mixture-of-Expert structure in MLP layer, which consists of a gate router and 8 expert layers. The softmax activations, however, present the similar pattern.

In Figure 9, we present more heatmaps of maxima position in self-attention of different models. With the help of prefixed tokens, the maxima of softmax now mostly show up at the *bos* token position, the first and those last two elements of each row.

In Figure 10, we demonstrate more histograms and violin plot of activations for Normalization in different LLMs. We also show histograms of activations with prefixed tokens for better comparison. After prefixing tokens, the long tails in the left columns of pictures have been eliminated. We also provide violin plots in Figure 11 for more concise view. Additionally, we use the same LLM model with prompts sampled from WikiText (Merity et al., 2017), RedPajama(?) and RedPajama

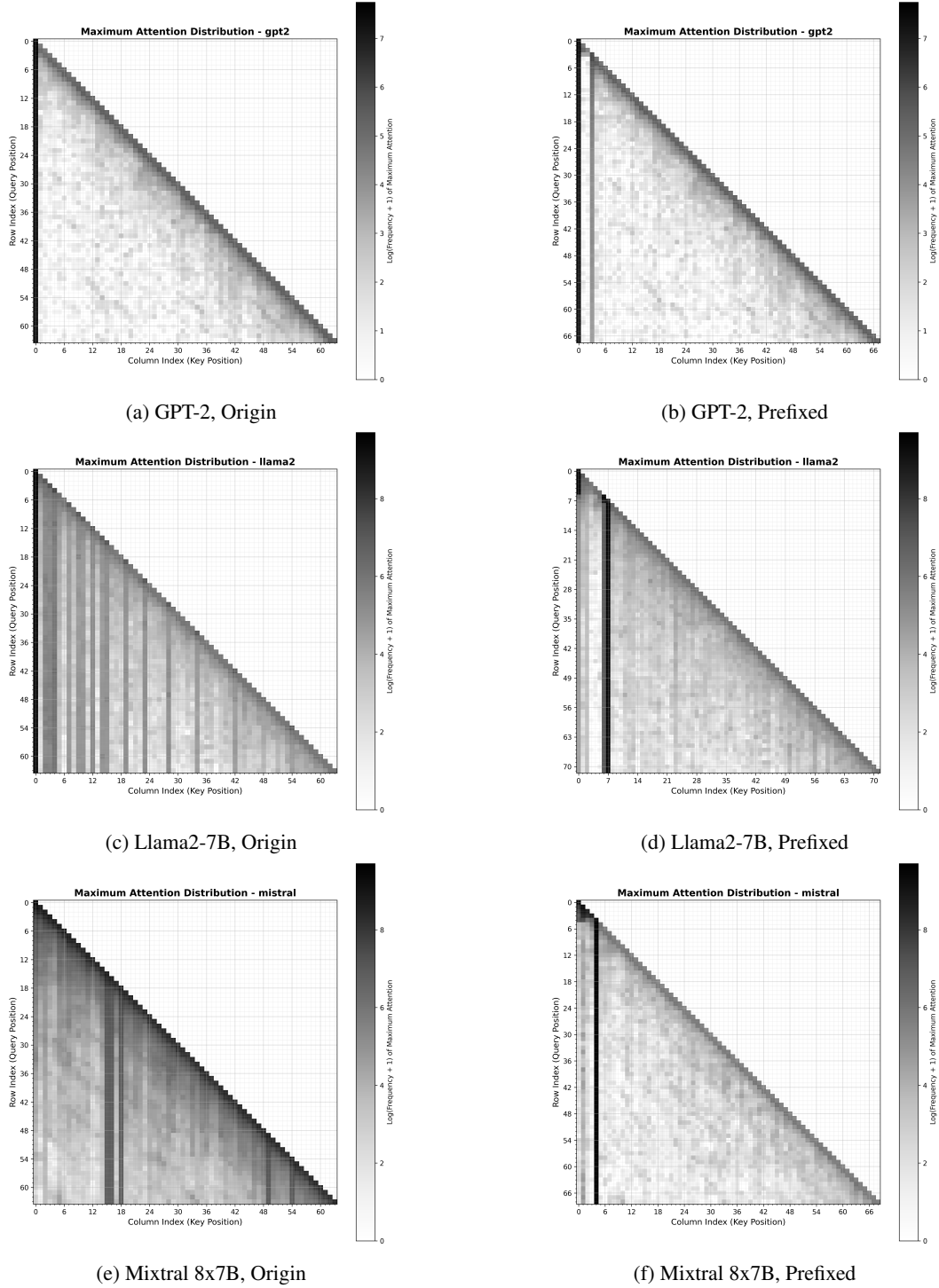
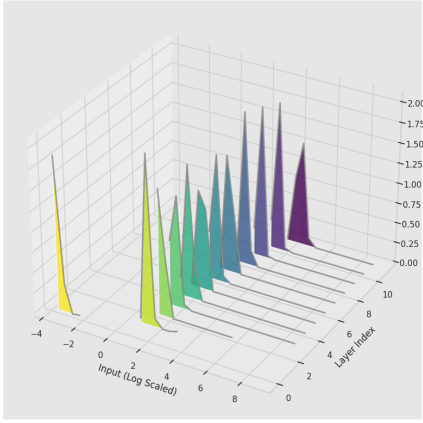
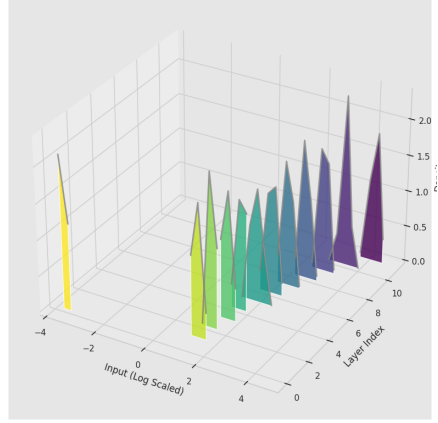


Figure 9: Heatmap of maxima position in Self-Attention of GPT-2 and Mixtral 8x7B. Grayscale means the times maxima appear at that position.

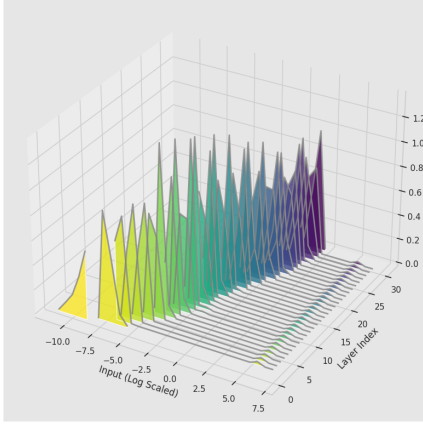
with scrambled order. Both the *Special-Token Outlier* and *Conformant Maxima* phenomena persists meaning that these features are adhere to LLM, rather than specific input distribution.



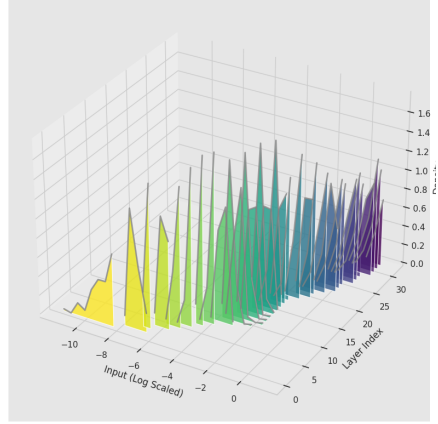
(a) GPT-2 origin



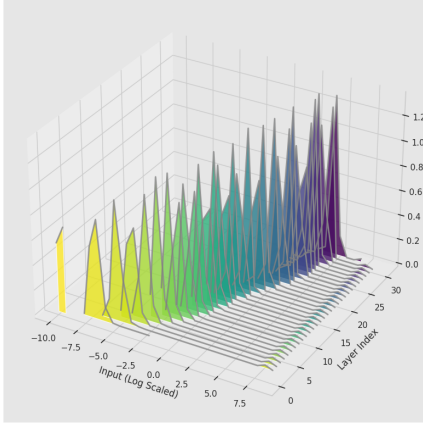
(b) GPT-2 prefixed



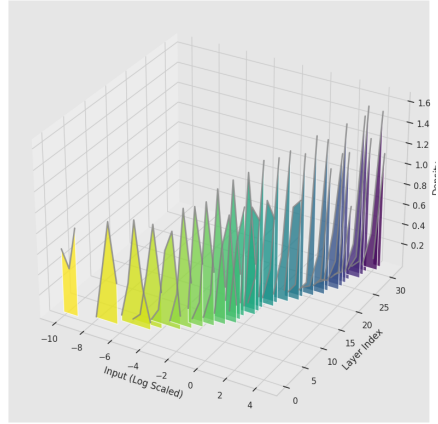
(c) Llama2-7B origin



(d) Llama2-7B prefixed



(e) Mixtral 8x7B origin



(f) Mixtral 8x7B prefixed

Figure 10: Histogram of activations for origin and prefixed LayerNorm in different LLMs. The extended line on x-y plane is for better visualizing the location of maximum and minimum values. The x-axis is log scaled.

A.2 EXTRA LLMs

The rapid development and release cycle of large language models (LLMs) significantly outpaces the progress of cryptographic research. Therefore, to ensure comparability with existing literature

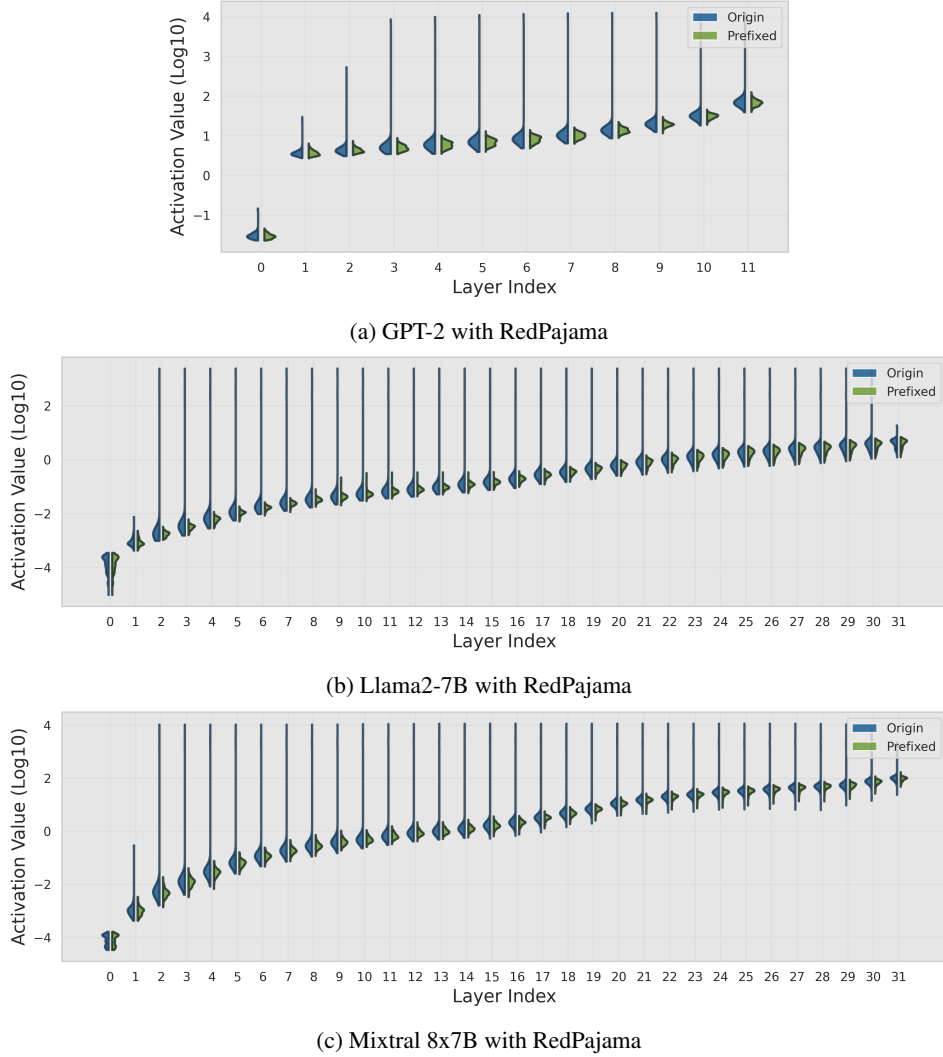


Figure 11: Violin plot of layerwise activations for normalization in different models. (Log scaled).

on secure LLM inference, this study utilizes the well-established GPT-2 and Llama2-7B models. Furthermore, as demonstrated in Figure 13, the observed phenomenon is not limited to specific architectures; we corroborate our findings using the more recent LLama3-8B (Grattafiori et al., 2024), Gemma2-9B (Team et al., 2024), and Qwen3-8B (Yang et al., 2025) models.

A.3 CROSS DOMAIN AND CROSS MODAL

To address the concern about the outliers remain valid for domain shift, we conducted extensive new experiments covering cross-domain, and cross-modal scenarios, detailed below:

1. Robustness to Domain Shift

- Code: We validated the approach on the Llama2-7B model using the ‘nick007x/github-code-2025’ dataset. As shown in Figure 14, the experimental results confirmed that our observations regarding both the special token outliers and conformant maxima remain valid in the code domain.
- Multilingual: Multilingual brings low-frequency tokens to prompts, while long-context data uses different rotary embedding methods during inference. As Llama2-7B only supports English and limits to 4096 tokens, we used Llama3.1-8B instead.

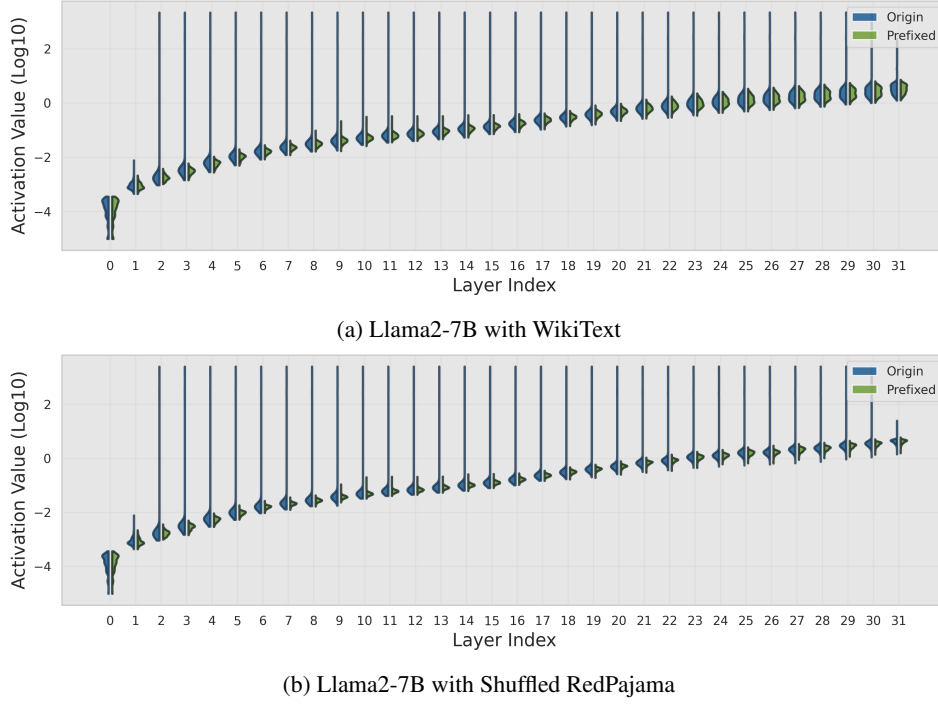


Figure 12: Violin plot of normalization activation using different datasets. (Log scaled).

For multilingual, we used 8 languages ['en', 'de', 'fr', 'it', 'pt', 'hi', 'es', 'th'] from huggingface dataset 'intfloat/multilingual.cc_news'. We show the result in Figure 15.

- Long-context: For long-context, we used 'jdoo2/Qwen2.5-32B-Instruct_long_context_range80-100_train_data_100k' dataset and selected inputs longer than 8192 tokens. The special token is $\langle |begin_of_text| \rangle$ whose token id is 128000 in both cases. The result is shown in Figure 16.

2. Robustness to Cross-Modal

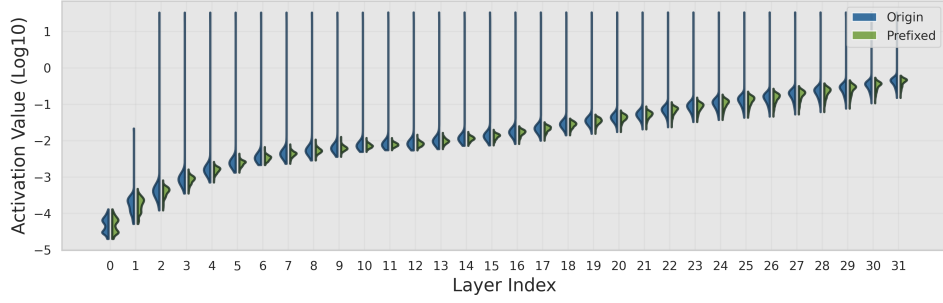
To further prove the generality of the phenomenon, we extended the validation to a multi-modal setting. We conducted experiments using the Qwen2-VL-8B model with a Image-Text-to-Text dataset, 'Open-Bee/Honey-Data-15M', and present the result in Figure 17. After filtering out the direct influence of the image tokens, the presence of special token outliers was consistently observed, demonstrating that this characteristic is shared even in multimodal domain.

3. Robustness to Finetuned Model Llama2-7B-chat is a RLHF finetuned version based on Llama2 7B. We conducted experiments on the instructed version of Llama2 with Redpajama datasets. As shown in Figure 18, the special token outliers persists in the finetuned version.

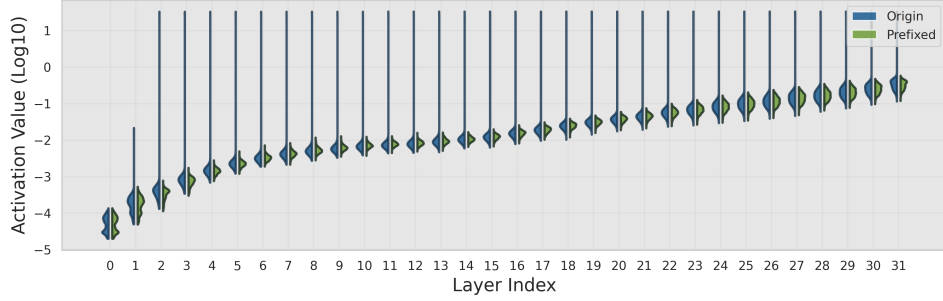
B PRELIMINARIES

B.1 THE STRUCTURE OF THE TRANSFORMER MODELS

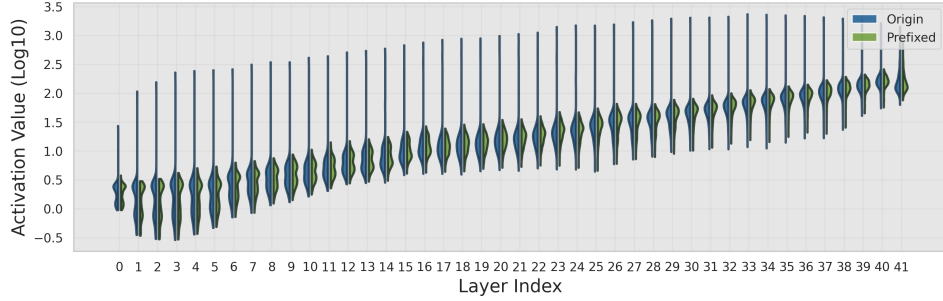
Modern Large language models are comprised of n transformer blocks stacked one after another. Each block contains a self-attention layer and a feed-forward layer, each preceded by a LayerNorm or RMSNorm operation and followed by a linear transform and a residual connection. Let L be the sequence length, F be the dimension of hidden feature, d be the head dimension and h be the number of attention heads such that $F = h \times d$. In self-attention layer, the normalized input $X \in L, F$ gets linear transformed to query Q , key K and value V hidden states by three corresponding linear layers, $W_Q, W_K, W_V \in \mathbb{R}^{F, F}$. Afterward, attention result is calculated by first softmaxing the matrix



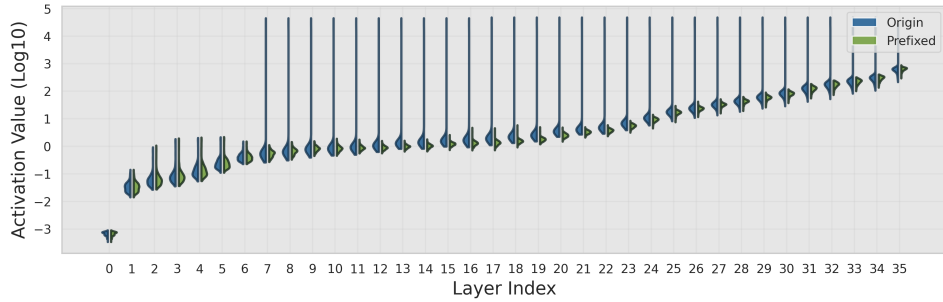
(a) Llama3-8B with Redpajama



(b) Llama3-8B with WikiText



(c) Gemma2-9B with Redpajama



(d) Qwen3-8B with Redpajama

Figure 13: Violin plots of more LLMs using prompts sampled from different datasets.

multiplication result of query and key and then multiplying value states, $A = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$. In feed-forward layer, the normalized inputs also pass through linear layers, and then one of them go through the activation gate, e.g. GeLU or SiLU. Thus, matrix multiplication are the dominated linear operations, and non-linear operations involve softmax, normalization, and activation layers.

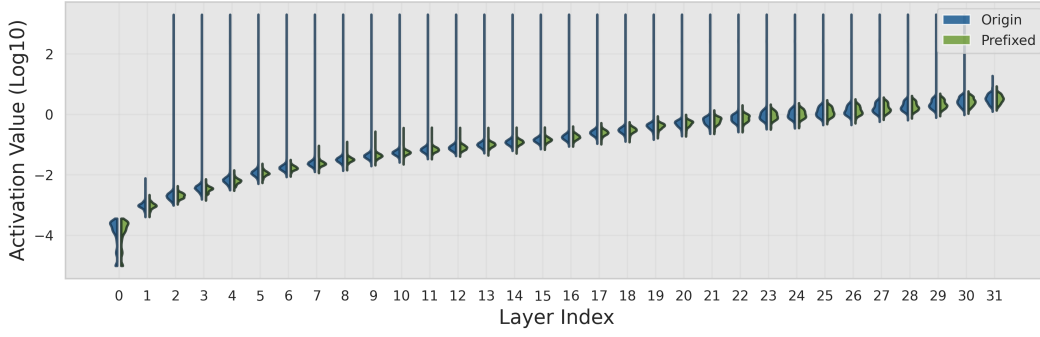


Figure 14: Llama2-7B with Code

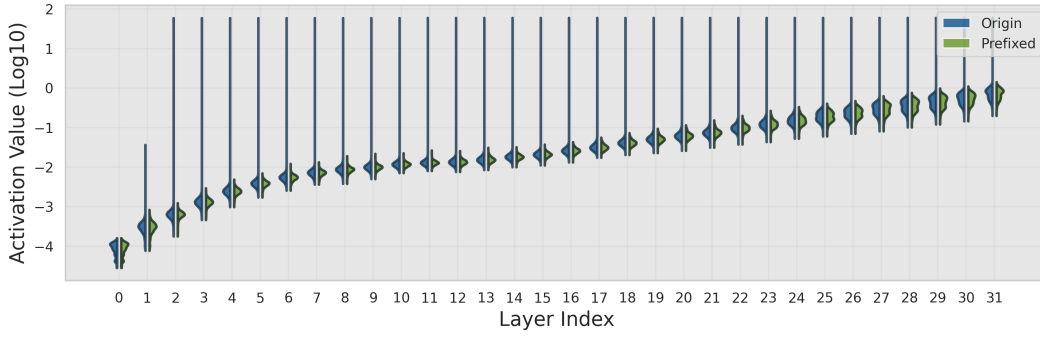


Figure 15: Llama3.1-8B with Multilingual

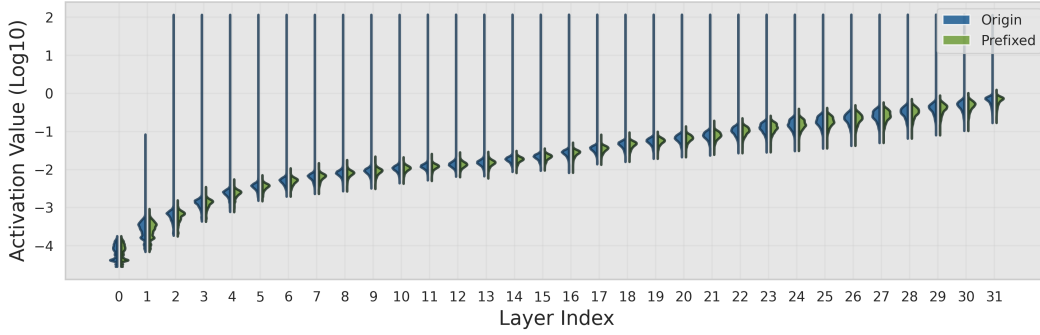


Figure 16: Llama3.1-8B with Long-Context

B.2 MULTI-PARTY COMPUTATION

This work focuses on standard 2PC in the preprocessing model (Beaver, 1991; Bendlin et al., 2011; Damgård & Zakarias, 2013) that has received significant attention in privacy-preserving LLM inference (Gupta et al., 2022; Knott et al., 2021; Jawalkar et al., 2024; Luo et al., 2024; Kei & Chow, 2025). That is, there are two semi-honest (honest but curious) parties, P_0 as the client who has the input prompt x_0 and P_1 as the server who offers the inference service with the model weights x_1 . Together, they want to perform the computation of a public function $y = F(x_0, x_1)$ without revealing anything more than the function output y to each other. The entire inference is separated into two phases. During the preprocessing phase, correlated randomness is generated and consumed by P_0 and P_1 during the online phase. In this work, we take the common assumption that there exists a trusted dealer to offer such randomness (Gupta et al., 2022; Jawalkar et al., 2024; Knott et al., 2021; Storrier et al., 2023; Watson et al., 2022).

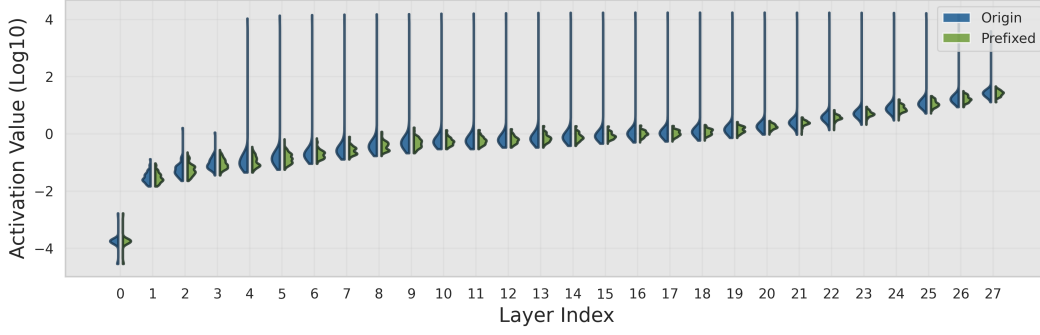


Figure 17: Qwen2 VL 8B with Image-Text-to-Text

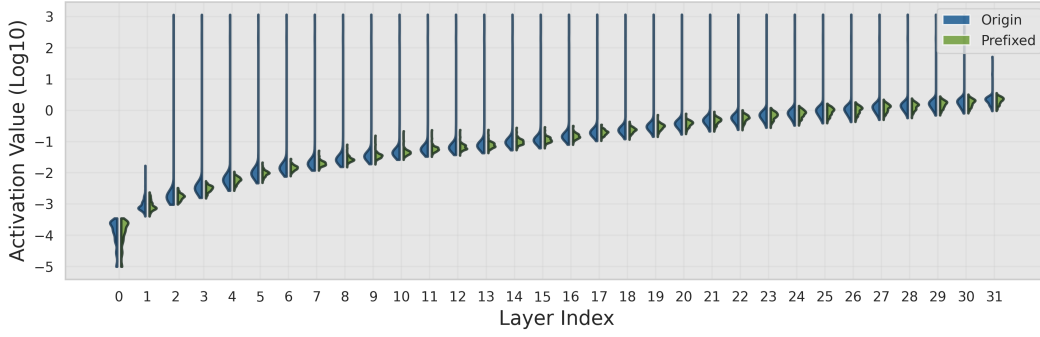


Figure 18: Llama2-7B-chat with Redpajama.

B.3 MPC FOR LLM INFERENCE

Transformer-based LLMs consist of four operations. One is matrix multiplication used in the linear layer. The other three are SoftMax, Normalization and Activation, which are non-linear operations that are hard to compute.

- The SoftMax function is the key element in the multi-head attention layer to calculate the attention score. Softmax function receives attention logits $X \in \mathbb{R}^{h \times L \times L}$ as input, where h is the number of heads and L is the sequence length. For the i -th row of X , $\mathbf{x} = \{x_0, x_1, \dots, x_L\}$, the masked attention is calculated with $\text{softmax}(\mathbf{x}) = \frac{\exp(\mathbf{x} - \max(\mathbf{x}))}{\sum_{j=1}^L \exp(x_j - \max(\mathbf{x}))}$.
- Normalization is used between multi-head attention layers and feed-forward layers to keep activations from exploding or vanishing. For the input $X \in \mathbb{R}^{L \times F}$, where F is the feature dimension, LayerNorm of each row $\mathbf{x}$ is calculated with $\gamma \cdot \frac{\mathbf{x} - \mu}{\sqrt{\frac{1}{F} \sum_{j=1}^F (x_j - \mu)^2 + \epsilon}} + \beta$, where μ is the mean of $\mathbf{x}$, and γ, β are fixed parameters during inference. RMSNorm, used by many recent LLMs (Gupta et al., 2022; Jiang et al., 2023), has similar format except for not subtracting the average.
- Activation is designed to bring non-linearity to the model. Recent LLMs mainly adopt GeLU (Hendrycks & Gimpel, 2023) and SwiGLU (Shazeer, 2020) as their activation functions. For the input $X \in \mathbb{R}^{L \times F}$, the SiLU function is calculated with $\text{SiLU}(x) = x \cdot \sigma(x)$, where $\sigma(x) = \frac{1}{1 + e^{-x}}$ is the sigmoid function. Similarly, the GeLU function can also be approximated with a sigmoid function $\text{GeLU}(x) = 0.5x \cdot \sigma(1.702x)$.

For non-linear operations with a large input domain, one either needs more bits to handle or more iterations to approximate, which consequently becomes the bottleneck. We describe detailed protocols from other works in Appendix B. Other difficulty, i.e., maximum finding in SoftMax, is related to sequence length. To find the maximum of an L -token input always requires executing $2 \lceil \log_2(L) \rceil$

comparisons. Such problem is getting critical as LLMs have been proven to perform better with proper prompts provided (Sahoo et al., 2025).

B.4 ASS DEFINITION

We describe the additive secret sharing scheme here as a means to preserve privacy during LLM inference. Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{U}_N$ denote the set of n -bit unsigned integers, where $N = 2^n$. First of all, we use fixed-point representation, parameterized by bit-width n and precision f , to encode a real value $x_r \in \mathbb{R}$ into an n -bit integer $x \in \mathbb{U}_N$ such that $x = \lfloor x_r \cdot 2^f \rfloor \bmod N$. Then, for $x \in \mathbb{U}_N$, additive secret shares are generated by randomly selecting a number $r \in \mathbb{U}_N$, and letting $\llbracket x \rrbracket_0 = r$ and $\llbracket x \rrbracket_1 = (x - r) \bmod N$. To reconstruct the original value, the two parties can reveal their secret shares and simply calculate $(\llbracket x \rrbracket_0 + \llbracket x \rrbracket_1) \bmod N$. Note that SOAL does not specify cryptographic primitives.

Since neither $\llbracket x \rrbracket_0$ nor $\llbracket x \rrbracket_1$ alone can infer the original value x , we need protocols to perform basic operations, e.g., beaver triple protocol (Beaver, 1991) for multiplication. We use the notation $\Pi_F(\llbracket x \rrbracket_\sigma)$, $\sigma \in \{0, 1\}$, where F stand for the detail operation. More complex non-linear operations are usually built upon basic operations with different high-level algorithms. For example, exponential protocols can be iterative approached by approximation method, like Taylor expansion (Knott et al., 2021; Luo et al., 2024), as well as exact calculated by bit-wise method, like LUT (Rathee et al., 2021; Gupta et al., 2024). Our contribution also focus on high-level algorithm design, thereby we adopt these protocols in a block box manner

B.5 FSS DEFINITION

In this section, we introduce the scheme of Function Secret Sharing (FSS). For the calculation of $f(x)$, unlike Additive Secret Sharing (ASS) which splits the input x into x_0, x_1 , FSS scheme split the function g . FSS consists of a pair of algorithms (Gen, Eval). Gen splits f into two function shares f_0, f_1 and Eval takes function share f_σ and input x , and returns $f_\sigma(x)$, which FSS ensures $f_0(x) + f_1(x) = f(x)$ for all x .

Definition B.1 (FSS: Syntax (Boyle et al., 2015; 2016)). A (2-party) function secret sharing (FSS) scheme is a pair of algorithms (Gen, Eval) such that:

- Gen($1^\lambda, \hat{f}$) is a PPT key generation algorithm that given 1^λ and $\hat{f} \in \{0, 1\}^*$ (description of a function f) outputs a pair of keys k_0, k_1 . We assume that $\hat{f}$ explicitly contains descriptions of input and output groups $\mathbb{G}^{in}, \mathbb{G}^{out}$
- Eval(σ, k_σ, x) is a polynomial-time evaluation algorithm that given $\sigma \in \{0, 1\}$ (party index), k_σ (key defining $f_\sigma : \mathbb{G}^{in} \rightarrow \mathbb{G}^{out}$) and $x \in \mathbb{G}^{in}$ (input for f_σ) outputs a group element $y_\sigma \in \mathbb{G}^{out}$ (the value of $f_\sigma(x)$).

Gen produces a pair of keys (k_0, k_1) which are called FSS keys and the number of bits required to store one FSS key is called *key size* (Gupta et al., 2024). As our method uses FSS as a black box manner, we would refer reader for the correctness and security definition in (Boyle et al., 2016).

To secretly process the input x with FSS, we need to preprocess x by hiding it with random values. Consider secure computation of a circuit with gate $\{g_i\}_i$ and wires $\{w_i\}_i$. We describe the 2PC protocol with preprocessing using FSS from (Boyle et al., 2019) in two phases: a preprocessing phase and an online phase.

B.5.1 PREPROCESSING PHASE

In this phase, the dealer generates all necessary correlated randomness before receiving the actual private inputs. For each input wire w_i in the circuit, a random mask r_i is sampled from the appropriate group. For each gate g in the circuit that takes an input from wire w_i and produces an output on wire w_j , the dealer generates an FSS key for its offset function $g^{[r_i, r_j]}(x) = g(x - r_i) + r_j$ for each party. For any wires that correspond to a party’s private inputs or outputs, that party also receives the corresponding random masks.

B.5.2 ONLINE PHASE

This phase is executed after the parties have received their keys. For the party σ that has input wire w_i with value x_i , party σ first masks the value with r_i to get $\hat{x}_i = x_i + r_i$ and sends it to the other party $1 - \sigma$. The parties then proceed to evaluate the circuit gate-by-gate in topological order. To evaluate a gate g with input wire w_i and output wire w_j , both parties apply their respective FSS key shares to the masked input $\hat{x}_i$ and get the secret shares of the masked output value $\hat{x}_j$. By reconstructing these shares, the parties now have the masked output values $\hat{x}_j = g^{[r_i, r_j]}(\hat{x}_i) = g(\hat{x}_i - r_i) + r_j = g(x_i) + r_j$. To get the decrypted output, the party that owns the output wire subtracts the corresponding mask to obtain the unmasked final result.

A protocol for securely computing a function F with a preprocessing phase can be formally defined by two sub-protocols:

- Gen_F : The key generation protocol executed in the preprocessing phase. This protocol takes input masks r_{in} and output masks r_{out} as input and generates the FSS keys for the parties.
- Eval_F : The evaluation protocol executed in the online phase. This protocol takes the masked input $\hat{x} = x + r_{\text{in}}$ and the FSS keys as input. At the end of the protocol, each party learns a secret share of the masked result $\hat{y} = y + r_{\text{out}}$.

As our framework uses these protocols in the black box manner, we refer the reader to (Boyle et al., 2015; 2019) for the detailed security analysis.

C CRYPTEN PROTOCOLS FOR NON-LINEAR LAYERS

In this section, we introduce the protocols used by CrypTen for non-linear operations. CrypTen uses numerical approximations to compute the complex functions like the exponential and reciprocal. Even though they mentioned that one can modify the domain of convergence as well as change configurable parameters to optimize protocols, such modifications have to universally fit the inputs with many orders of magnitude across different layers.

C.1 EXPONENTIAL

For exponential operation, CrypTen uses the limit approximation

$$e^x = \lim_{n \rightarrow \infty} \left(1 + \frac{x}{2^n}\right)^{2^n}.$$

They set $n = 8$ by default which makes the approximation effective for an input domain between $[-4, 4]$ with 3% relative error.

C.2 RECIPROCAL

For reciprocal operation, CrypTen use Newton-Raphson iterations to compute the reciprocal function. Given the initial guess y_0 , the update function is as follows:

$$y_{n+1} = y_n(2 - xy_n).$$

Such a function requires the initial guess to be $0 < y_0 < \frac{2}{x}$ to converge. By default, the initial guess is generated with $y_0(x) = 3e^{0.5-x} + 0.003$ and the number of iterations is set to 10.

Note that in detail implementation, CrypTen executes the update function as $y_{n+1} = 2y_n - 2xy_n^2$, since the square protocol costs one element fewer than the multiplication protocol. However, such a square protocol may turn results to zero when inputs are small because of fixed-point representation.

For example, if the initial guess y_0 is 0.003, squaring first would give us $y_0^2 = \frac{199^2}{65536} = \text{int}(0.604) = 0$. Hence, we do not use square protocol for CrypTen’s reciprocal in our experiments.

C.3 RECIPROCAL SQUARE ROOT

CrypTen uses an efficient Newton-Raphson update function as follows:

$$y_{n+1} = \frac{1}{2}y_n(3 - xy_n^2).$$

And the initial guess is obtained by $y_0 = 2.2 \cdot e^{-(\frac{x}{2}+0.2)} + 0.2 - \frac{x}{1024}$. They set the number of iterations to 3.

As mentioned in Section 4, such an input domain cannot cover the real domain requirement for LLMs, like Llama2-7B and Mixtral. To cover a larger input domain, one would have to increase the number of iterations and use smaller initial guess since the corresponding exponential cannot provide an accurate result when the input is large.

C.4 SIGMOID

CrypTen also uses Newton-Raphson iterations to approach the result. It first uses a comparison protocol to find the sign of the input $\llbracket x \rrbracket_\sigma$. Then, they compute the sigmoid for $|x|$ so that the initial value can be set to 0.75, as the output of the sigmoid for positive values is between $[0.5, 1]$. Afterward, the outputs for negative inputs can be obtained with $1 - \sigma(|x|)$.

D SIGMA PROTOCOLS FOR NON-LINEAR LAYERS

In this section, we introduce Sigma’s non-linear protocols (Gupta et al., 2024).

D.1 NEGATIVE EXPONENTIAL

In LLM Inference, the only place need to use exponential is in SoftMax, so Sigma focus on negative exponential $\text{nExp}(x) = e^{-x}$ for $x \in \mathbb{R}^+$. For a fixed-point representation with a precision bit of $f = 12$, Sigma clips the inputs to the interval $[0, 16) \in \mathbb{R}^+$, since anything larger than 16 will become zero due to $\lfloor e^{-x} \cdot 2^{12} \rfloor = 0$. For $x \in [0, 16)$, they need 16 bits to represent the input value. They then choose to truncate the clipped input c into two parts $c = c_1 \parallel c_0$ where c_1 comprises the upper 8 bits and c_0 comprises the lower 8 bits. They then use two 8-bit LUTs with c_0 and c_1 , respectively. Therefore, they transform the negative exponential into (Gupta et al., 2024)

$$\lfloor \text{nExp}(\frac{c}{2^{12}} \cdot 2^f) \rfloor \approx \text{ARS}_{n,f}(T_1[c_1] \cdot T_0[c_0]),$$

where T_1 and T_0 are 8-bit LUTs with n -bit values such that $T_1[i] = \lfloor \text{nExp}(i/2^4) \cdot 2^f \rfloor$ and $T_0[i] = \lfloor \text{nExp}(i/2^{12}) \cdot 2^f \rfloor$ for all $i \in \mathbb{U}_{2^8}$. ARS stands for Arithmetic Right Shift, for which Sigma provides an efficient truncation Π_{GapARS} that can reduce ARS to a Logical Right Shift if input value v has a known gap such that $v \in [0, 2^{n-2}) \cup [2^n - 2^{n-2}, 2^n)$.

D.2 RECIPROCAL

For the reciprocal in SoftMax, Sigma first chops off lower bits as the denominator $\sum_{i=0}^L e^{(x_i - x_{\text{Max}})}$ of SoftMax is sure to be between $[1, L]$. Therefore, they reduce precision to 6, and use a q -bit LUT to read the output of inverse, where $q = 6 + \lceil \log_2(L + 1) \rceil$.

D.3 RECIPROCAL SQUARE ROOT

Sigma converts a fixed-point number into a floating point representation to represent a large dynamic domain using a small number of bits. Let $k \in \mathbb{U}_{64}$ be a number such that $2^{k-1} \leq x < 2^k$. The exponent is $e = k - 1$, and the mantissa is $m \approx \frac{x \cdot 2^{n-k}}{2^{n-8}} - 128 = \text{TR}_{n,n-8}(x \cdot u) \bmod 128$, where $u = 2^{n-k}$. To do so, they first use interval lookup protocols to obtain e and u . After converting u to m , Sigma gets a 13-bit floating point representation as $p = m \parallel e$.

D.4 GeLU/SiLU

Sigma observes that $\delta(x) = \text{ReLU}(x) - \text{GeLU}(x)$ and $\delta(x) = \text{ReLU}(x) - \text{SiLU}(x)$ exhibit the symmetric property that the difference $\delta(x)$ outside $(-4, 4)$ is negligible. Based on this observation, they first get the sign of input x with $d = \Pi_{\text{DReLU}}(x)$, and get the clipped input c between $(-4, 4)$. For those values within the interval, they use a 8-bit LUT protocols to get the $\delta(x) = \Pi_{\text{LUT}}^{8,n,T}(c)$. To restore GeLU or SiLU output, one can simply calculate $\Pi_{\text{select}}^n(d, x) - \Pi_{\text{LUT}}^{8,n,T}(c)$.

Algorithm 6: Sigma Reciprocal Square Root $\Pi_{\text{rsqrt}}^{n, f^{\text{in}}, f^{\text{out}}}(\hat{x})$

```

1  $(\hat{e}_b, \hat{u}_b) \leftarrow \Pi_{\text{IntervalLookup}}^{n, \mathbb{G}, p, q, v}(\hat{x})$ 
2  $\hat{u} = \text{reconstruct}(\hat{u}_b)$ 
3  $\hat{t} \leftarrow \hat{\Pi}_{\text{Mul}}^n(\hat{x}, \hat{u})$ 
4  $\hat{m}_b \leftarrow \hat{\Pi}_{\text{TR}}^{n, n-8}(\hat{t}) \bmod 128$ 
5  $\hat{p}_b \leftarrow \text{extend}^{7, 13}(\hat{m}_b) \cdot 2^6 + \hat{e}_b$ 
6  $\hat{p} = \text{reconstruct}(\hat{p}_b)$ 
7 return  $\Pi_{\text{LUT}}^{13, n, T}(\hat{p})$ 

```

Algorithm 7: Sigma GeLU(GPU) $\Pi_{\text{GeLUGPU}}^{n, m, f}(\hat{x})$

```

1  $\hat{y} \leftarrow \hat{\Pi}_{\text{TR}}^{m, f-6}(\hat{x} \bmod 2^m)$ 
2  $\hat{i}_b \leftarrow \Pi_{\text{DReLU}}^{m-f+6}(\hat{y} + 255) \oplus \Pi_{\text{DReLU}}^{m-f+6}(\hat{y} - 256)$ 
3  $\hat{d}_b \leftarrow \Pi_{\text{DReLU}}^{m-f+6}(\hat{y})$ 
4  $(\hat{i}, \hat{d}) = \text{reconstruct}(\hat{i}_b, \hat{d}_b)$ 
5  $\hat{z} = \hat{y} \bmod 256$ 
6  $\hat{c} \leftarrow \Pi_{\text{selectlin}_\gamma}^8(\hat{i}, \hat{d}, \hat{z})$ 
7 return  $\Pi_{\text{select}}^n(\hat{d}, \hat{x}) - \Pi_{\text{LUT}}^{8, n, T}(\hat{c})$ 

```

E SOAL-FSS PROTOCOLS FOR NON-LINEAR LAYERS

We introduce our new protocols based on FSS here. Although Sigma(Gupta et al., 2024) mainly focuses on online communication costs, the key size of FSS scheme is non-negligible. To perform one inference with a 512-token input for GPT-2, it generates a key of 86.686 GB for each party. Therefore, the goal of our FSS design is to lower the total key size while at the same time maintaining the efficiency of the online cost.

E.1 NORMALIZATION

As mentioned in Section 4.1, in each layer, the majority of activations for Normalization are concentrated around a certain value and have a rather small variance. Therefore, our RSqrt protocol is highly entangled with the Normalization protocol. We first multiply the input with an encrypted value φ to secretly scale activations into our desired domain. Then, we can use a 12-bit LUT to gain a correctly scaled output. At last, the scaled value would automatically be canceled out by the multiplication operation. We demonstrate our Normalization protocol in Algorithm 8

Algorithm 8: SOAL Normalization $\Pi_{\text{Norm}}^{n, f}(\hat{x})$

Input: For $\sigma \in \{0, 1\}$, Party σ hold $\hat{x}$ and $\hat{\varphi}$.

```

1  $\hat{y} \leftarrow \hat{\Pi}_{\text{GapARS}}^{n, f}(\hat{\Pi}_{\text{Mul}}^n(\hat{x}, \hat{\varphi}))$ 
2  $\hat{z} \leftarrow \hat{\Pi}_{\text{TR}}^{n-4, 4}(\hat{y} \bmod 2^4)$ 
3  $\hat{d} \leftarrow \hat{\Pi}_{\text{LUT}}^{12, n, T}(\hat{z})$ 
4 return  $\hat{\Pi}_{\text{Mul}}^n(\hat{y}, \hat{d})$ 

```

E.2 ACTIVATION

As the Sigma’s GeLU and SiLU protocols has clipped the values to $[-4, 4]$, which is similar to our idea of reducing input domain, we did not implement new SiLU or GeLU protocol under FSS schemes. We adopt Sigma’s protocol.

E.3 EXPONENTIAL

As our method finds τ for each row of SoftMax inputs, the property of negativity no longer holds for exponential calculation. Our FSS implementation follows the same idea that the input of x can be split into upper x_u and lower bits x_l . We find that for the lower bits, its exponential e^{x_u} can be calculated locally, which saves one LUT protocol and allows a larger domain for x_u . The only problem is that the actual input x has been masked by r , which makes the received input become $\hat{x}_l = x_l + r_l$. As the exponential of a sum is equal to the product of separate exponentials, we need to secretly divide the online result by e^{r_l} . Our main idea is integrating such division with ASS. Given a random value $r' \in \mathbb{U}_N$, we can have $\llbracket e \rrbracket_0 = r'$, $\llbracket e \rrbracket_1 = e^{-r_l} - r' \bmod N$ and store $\llbracket e \rrbracket_\sigma$ into keys respectively. To mask the output, we need another $r'' \in \mathbb{U}_N$, we store $\llbracket r'' \rrbracket_\sigma$ into keys as well.

During the online phase, we can first locally compute $e^{\hat{x}_l} = e^{x_l} \cdot e^{r_l}$. Then, we multiply it with $\llbracket e \rrbracket_\sigma$ from provided key and get $\llbracket t \rrbracket_\sigma = e^{\hat{x}_l} \cdot \llbracket e \rrbracket_\sigma$. At last, the parties reconstruct the result by revealing $\llbracket t \rrbracket_\sigma + \llbracket r'' \rrbracket_\sigma$ to each other.

$$\begin{aligned}\llbracket t \rrbracket_0 + \llbracket r'' \rrbracket_0 &= e^{x_l} \cdot e^{r_l} \cdot r' + \llbracket r'' \rrbracket_0, \\ \llbracket t \rrbracket_1 + \llbracket r'' \rrbracket_1 &= e^{x_l} \cdot e^{r_l} \cdot (e^{-r_l} - r') + \llbracket r'' \rrbracket_1,\end{aligned}$$

By doing so, we only need to store $2n$ bits in the key. Online phase only need the communication of n bits in 1 round. For the rest of the part, we follow the same implementation as Sigma, using a 8-bit LUT to calculate $\hat{x}_u$ and get the result by multiplying them. We demonstrate the exponential algorithm in Algorithm 9

Algorithm 9: SOAL Exponential $\Pi_{\text{Exp}}^{n,f}(\hat{x})$

Input: For $\sigma \in \{0, 1\}$, Party σ hold $\hat{x}$ and k_σ .

- 1 $\hat{x}_u \leftarrow \hat{\Pi}_{\text{TR}}^{n-f+1,f}(\hat{x} \bmod 2^f)$; $\hat{x}_l \leftarrow \hat{x} - 2^f \cdot \hat{x}_u$
 - 2 $\hat{e} \leftarrow \Pi_{\text{Exp}}^*(\hat{x}_l) // \text{local exp}$
 - 3 $\llbracket \hat{t}_l \rrbracket_\sigma \leftarrow \Pi_{\text{Mul}}^n(\hat{e}, \llbracket r' \rrbracket_\sigma) // \llbracket r' \rrbracket_\sigma \text{ is from } k_\sigma$
 - 4 $\hat{t}_l \leftarrow \text{reconstruct}(\llbracket \hat{t}_l \rrbracket_\sigma + \llbracket r'' \rrbracket_\sigma)$
 - 5 $\hat{t}_l \leftarrow \hat{\Pi}_{\text{TR}}^{n,f}(\hat{t}_l)$
 - 6 $\hat{t}_u \leftarrow \hat{\Pi}_{\text{LUT}}^{8,n,T}(\hat{x}_l)$
 - 7 $\hat{t} \leftarrow \hat{\Pi}_{\text{Mul}}^n(\hat{t}_u, \hat{t}_l)$
 - 8 **return** $\Pi_{\text{GapARS}}^{n,f}(\hat{t})$.
-

F MORE EXPERIMENT RESULTS

F.1 TIME AND COMMUNICATION COST FOR SECURE LLM INFERENCE WITH 128-TOKEN INPUT.

In Table 5, we show the time and communication cost with an input of 128 tokens on different LLMs. The improvements on non-linear protocols are constant, while the cost of the linear layer might be dominant when the input is short. Thus, advanced cryptographic primitives may improve the basic operations, i.e., multiplication, faster; at the same time, our SOAL can still gain benefit from such improvement.

F.2 DETAIL COMPARISON FOR SOFTMAX AND LAYERNORM

Here we explicitly list the size of generated key during preprocessing phase, and the time and communication cost during online phase.

G DETAIL ACTIVATION LAYER INPUT DISTRIBUTION

We present the original input distribution for activation layers in GPT2 and Llama2.

Model	Method	SoftMax		Normalization		Activation		Total	
		Time (s)	Comm.(Gb)	Time (s)	Comm.(Gb)	Time (s)	Comm.(Gb)	Time (s)	Comm.(Gb)
GPT2	CrypTen	8.19	3.75	6.293	0.22	4.18	4.92	21.29	11.09
	SOAL	4.27	0.50	2.31	0.29	3.00	3.81	12.04	6.91
Llama2 7B	CrypTen	31.47	27.01	24.47	3.07	25.52	42.33	159.71	180.46
	SOAL	10.58	3.53	7.73	4.01	9.88	16.80	95.13	132.39
Mixtral 8x7B	CrypTen	57.23	27.25	67.93	6.08	83.03	110.25	675.16	854.28
	SOAL	22.20	3.61	16.86	8.08	35.59	40.35	473.25	759.95

Table 5: Time and Communication Cost for SOAL ASS LLM Inference with 128-token input. Average of 5 runs.

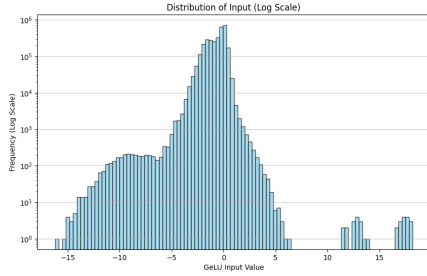


Figure 19: GPT2 GeLU Input Distribution, Layer 1

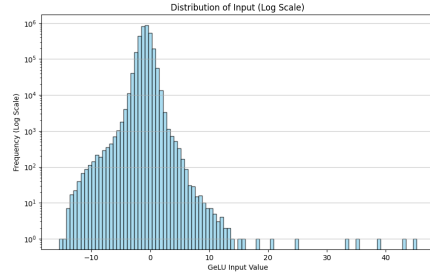


Figure 20: GPT2 GeLU Input Distribution, Layer 11

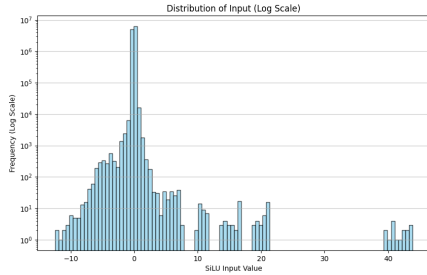


Figure 21: Llama2-7B SiLU Input Distribution, Layer 1

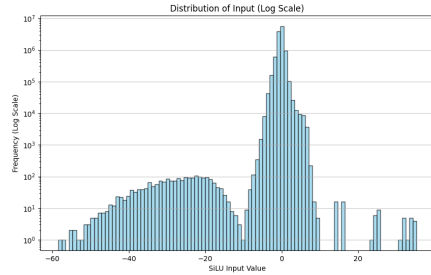


Figure 22: Llama2-7B SiLU Input Distribution, Layer 30

Length	Softmax						Normalization						Total					
	Time (s)			Key Size (Gb)			Time (s)			Key Size (Gb)			Time (s)			Comm(Gb)		
	Sigma	SOAL	Comm(Gb)	Sigma	SOAL	Comm(Gb)	Sigma	SOAL	Comm(Gb)	Sigma	SOAL	Comm(Gb)	Sigma	SOAL	Comm(Gb)	Sigma	SOAL	Comm(Gb)
32	0.083	0.060	0.009	0.010	0.004	0.028	0.183	0.100	0.028	0.007	0.007	0.007	0.666	0.542	0.175	3.555	3.202	0.175
64	0.124	0.086	0.033	0.040	0.0134	0.056	0.314	0.134	0.056	0.015	0.015	0.015	1.097	0.852	0.370	6.806	5.938	0.370
128	0.285	0.168	0.131	0.157	0.051	0.111	0.527	0.196	0.111	0.030	0.030	0.030	1.936	1.493	0.824	14.292	11.915	0.824
256	0.879	0.473	0.522	0.622	0.196	0.223	0.958	0.320	0.223	0.059	0.059	0.059	4.166	2.996	1.983	33.191	25.873	1.983
512	3.298	1.796	2.078	2.477	0.772	0.445	1.836	0.682	0.445	0.118	0.118	0.118	10.048	7.577	5.302	86.686	61.830	5.302
1024	10.974	6.071	8.297	9.877	3.062	0.890	3.294	1.089	0.890	0.236	0.236	0.236	25.136	18.486	15.949	256.449	165.887	15.949

Table 6: Detail time, communication cost and key size comparison for SOAL FSS