ZK-GPT: An Efficient Non-interactive Zero-knowledge Proof Framework for LLM Inference

Abstract

Large Language Models (LLMs) are widely employed for their ability to generate human-like text. However, service providers may deploy smaller models to reduce costs, potentially deceiving users. Zero-Knowledge Proofs (ZKPs) offer a solution by allowing providers to prove LLM inference without compromising the privacy of model parameters. Existing solutions either do not support LLM architectures or suffer from significant inefficiency and tremendous overhead. To address this issue, this paper introduces several new techniques. We propose new methods to efficiently prove linear and non-linear layers in LLMs, reducing computation overhead by orders of magnitude. To further enhance efficiency, we propose constraint fusion to reduce the overhead of proving non-linear layers and circuit squeeze to improve parallelism. We implement our efficient protocol, specifically tailored for popular LLM architectures like GPT-2, and deploy optimizations to enhance performance. Experiments show that our scheme can prove GPT-2 inference in less than 25 seconds. Compared with state-of-the-art systems such as Hao et al. (USENIX Security'24) and ZKML (Eurosys'24), our work achieves nearly 279× and 185× speedup, respectively.

1 Introduction

Recently, Large Language Models (LLMs) (*e.g.*, GPT-4 [2]) have revolutionized dialogue synthesis, applied in ChatBots [3], web search [34], and assisting programmers [65]. Due to their demanding resource requirements, AI companies typically deploy LLMs on cloud servers. Users access these models through APIs. Considering the huge daily serving costs, service providers are incentivized to cheat by using smaller models and giving lower-quality results to save costs. In fact, many users have observed the performance degradation of GPT-4 and ChatGPT [21, 22, 52] and suspect that OpenAI is using a smaller model. However, the inference process is black-boxed to users. The service providers will not reveal the model for users to validate the service since the model parameters are trade secrets [10]. Hence, it is difficult for users to validate the service's integrity, e.g., the specified model computes the output.

The zero-knowledge proof (ZKP) [29] is a promising solution to prove the integrity of machine learning (ML) services. In ZKP, a prover produces a proof π to convince the verifier that the result of a public function f on public input x and the secret input of the prover w is indeed $y = f(x, w)$. The secret input w is usually referred to as the witness. ZKP guarantees that the verifier will output rejection with overwhelming probability if the prover cheats on calculation, while the proof π does not reveal any additional information about the secret w. Using f as the function that represents the calculation of the model, x as user input, and w as model weights within the ZKP protocol, the service provider can prove the correctness of LLM inference. In real-world applications, regulatory authorities can frequently and anonymously query the LLM service and require the service provider to prove the validity of the generated answers via ZKP. If the provider fails to provide the valid proof, they may face penalties. This approach effectively safeguards consumer rights while imposing small overhead on existing infrastructure.

In recent years, extensive research has been conducted on ZKP for ML, covering a range of models from classic decision trees to modern neural networks (NN). However, most of these works either do not support LLM architectures [9, 18, 19, 36, 38, 41, 63, 71], or support LLMs [10, 31, 42, 57] but face significant limitations, such as inefficiency and high communication overhead. For example, concurrent work [10] requires over an hour to generate a single proof for the inference of GPT-2 [51]. Such lengthy proof generation times render these approaches impractical for real-world applications involving LLMs. The core challenge in developing practical ZKP schemes for LLMs lies in efficiently proving different layers involved in LLMs, such as matrix multiplication, attention [59], GeLU [32], and normalization [5]. This challenge arises for two main reasons: (1) the linear layers in LLMs are large in scale and frequently invoked, making their proof generation time consuming. (2) ZKP primarily operates within

arithmetic fields that only support addition and multiplication operations, making it difficult to efficiently prove non-linear layers.

In this paper, we aim to address the challenge of designing efficient ZKPs for both linear and non-linear layers, with the goal of generating practical ZKPs for GPT models.

Efficient proof for linear layers. The linear layers in LLMs encounter weight matrices that are extremely large. This makes proving matrix multiplications time consuming. Therefore, we first focus on improving the efficiency of these proofs. Thaler et al. [58] proposed a dedicated protocol for verifiable matrix multiplication, which we incorporate into our system. While the algorithm achieves optimal prover time in theory, there remains significant room for improving its practical implementation. In particular, the execution time of this protocol is bottlenecked by the computation of the bookkeeping table for each matrix. The bookkeeping table stores evaluations of the multi-linear extension of each matrix column at a given random point. The classical method [41, 58] for this task utilizes the memory method in [60] to evaluate each column's multi-linear extension. This requires approximately $4n \cdot m$ field multiplications per matrix, where n and m represent the number of rows and columns of the matrix, respectively. The major drawback of this method is that it does not leverage certain special properties of matrix elements to reduce computation as we have observed.

We propose a grouping algorithm to address this bottleneck efficiently, reducing the number of field multiplications involved in [58] to approximately half. We further accelerate our grouping algorithm by leveraging two key properties of our matrix elements: (1) the presence of many padded zeros due to sumcheck requirements, and (2) the relatively small range of values resulting from quantization. By utilizing our proposed method, the prover's bottleneck shifts to $n \cdot m$ field additions. Since field multiplications are several times more expensive than field additions, our algorithm achieves nearly a tenfold acceleration on this theoretically optimal protocol.

Efficient proof for non-linear layers. Proving non-linear layers also poses challenges in ZKP due to the involvement of ZKP-unfriendly operations such as division, square root, and exponentiation. Previous work based on floating-point circuits [63] or numerical iterations [31] incurs high overhead when proving these operations. For example, [63] requires thousands of relations to prove division and exponentiation operations. The inefficiency of existing methods stems from simulating non-arithmetic computations using arithmetic operations. To address this issue, we propose to use the computation result as advice, instead of simulating the computation in the arithmetic field. We found that, with the use of advice, division and square root operations can be proved by a single range relation. For exponentiation operations, we prepare a lookup table containing all possible evaluations of the exponentiation result. Then, each exponentiation operation can be verified with a single query to the lookup table. By leveraging

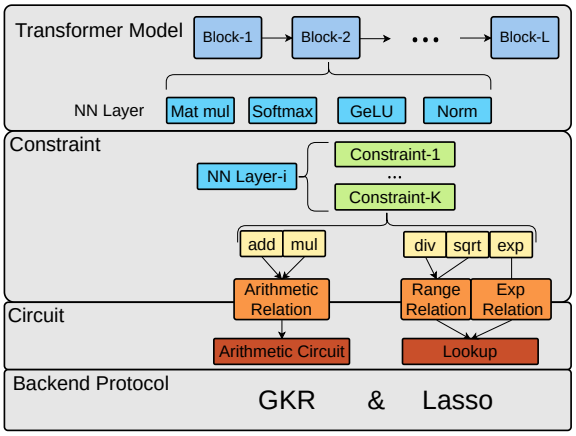


Figure 1: Workflow of the prover for LLM.

advice, our approach reduces the number of relations required for proof by several orders of magnitude, significantly lowering the prover's overhead for non-linear layers. To prove range relations and exponentiation operations, we employ the recent lookup scheme Lasso [55]. For proving the involved arithmetic relations, we adopt the GKR protocol [30]. By integrating the above proposed techniques, we have developed a proof system capable of efficiently proving different LLM layers.

Compared with existing ZKP schemes for LLMs [10, 31, 42, 57], our system offers significant advantages in the following aspects: (1) Prover efficiency. Our efficient design for proving both linear and non-linear layers greatly reduces the cost of proving LLMs. For example, [57] directly uses the protocol in [58] to prove matrix multiplication. Under the same settings, our system is nearly ten times faster than [57] in proving these layers due to the efficiency of our algorithm. Additionally, our system can prove all normalization layers in a GPT-2 model in less than 30 seconds, whereas Hao et al. [31] requires more than 4000 seconds. (2) Non-interactive and light communication. Our system can be easily transformed into a non-interactive proof using the Fiat-Shamir Transformation [20], enabling the proof to be downloaded and publicly verified offline. In contrast, previous VOLE-based solutions [31, 42] cannot be converted to non-interactive proofs. Additionally, VOLE-based solutions require gigabytes of data transfer between the prover and verifier, whereas our method is communication-efficient, with a proof size of 101 KB.

Performance improvement techniques. Although our system demonstrates several advantages, further optimizations are essential to ensure its applicability in real-world scenarios. Specifically, we aim to generate proof for GPT-2 inference within one minute on a CPU server. To achieve this, we perform comprehensive optimizations at both the constraint and circuit levels.

Constraints in ZKPs are essentially equations that define the computational relationships between variables. In our system, the constraints correspond to the computation process of the LLM. These constraints are then converted into

arithmetic and non-arithmetic relations. Non-arithmetic relations include range and exponentiation relations. Finally, our backend proves these relations. The workflow is illustrated in Figure 1. Our improvements consist of two key components: constraint fusion and circuit squeeze.

At the constraint level, we propose a generalized optimization technique called **constraint fusion**. This technique aims to combine the computation of two adjacent rounding constraints, thereby reducing the overall number of rounding constraints. In our system, range relations are introduced for checking the correctness of rounding constraints; however, the large number of range relations becomes a bottleneck during the proof generation. By reducing the number of rounding constraints, we can alleviate the associated range relations, thereby mitigating prover overhead. Importantly, our constraint fusion technique preserves the original computational precision. After applying constraint fusion, we significantly reduce the proving cost of different LLM layers.

At the circuit level, we propose a technique called **circuit squeeze** to accelerate the prover. We observed existing GKR-based systems [6, 41] cannot effectively utilize multi-thread parallelism. Profiling reveals that the low utilization stems from the cost of thread synchronization during the sumcheck at each layer. Our circuit squeeze significantly enhances parallelism by reducing circuit depth and increasing the number of gates per layer. This optimization is possible due to the topological independence of subcircuits responsible for verifying each rounding constraint. This independence arises from our design choice to store the outputs of rounding computations as advice in the input layer, rather than computing them directly. For matrix multiplication layers not connected to standard GKR-style gates, we extend the protocol from [58] to support proving multiple matrix multiplications simultaneously. Our method provides substantial acceleration for the LLM circuit by eliminating sequential dependencies in proving computations across different NN layers.

Our key contributions are summarized as follows.

1. We are the first to present a practical ZKP design for LLMs under general settings. Our system overcomes the inefficiencies and limitations of existing works, particularly those related to large communication overhead and hardware requirements. The framework is compatible with LLMs utilizing prevalent architectures such as GPT-2 [51] and is capable of proving its inference under 25 seconds.

2. We propose efficient proof constructions for linear and non-linear layers. For linear layers, we propose a new algorithm to accelerate the matrix multiplication protocol in [58], reducing its concrete computation cost by orders of magnitude. For non-linear layers, we propose efficient methods based on advice to prove different ZKP-unfriendly operations. This reduces the proving cost by orders compared with existing prevalent solutions. To further enhance our system's efficiency, we propose constraint fusion, reducing the number of range relations to prove. Additionally, We propose circuit squeeze, which offers substantial parallelism acceleration by eliminating the sequential dependency in proving different NN layers.

3. We conduct extensive experiments on real-world CPU servers to validate the efficiency of our system. Results show that our implementation achieves over 279× speedup compared to Hao et al. (USENIX Security'24) and 185× speedup over the state-of-the-art Plonk-based approach [10] (Eurosys'24). Even compared to the concurrent VOLE-based scheme [42], our non-interactive system delivers a 5.1× speedup in prover time and reduces the proof size by 23,000×.

2 Background

2.1 LLM and quantization

An LLM primarily consists of $\mathbf{L}$ stacked transformer blocks [59]. Each transformer block comprises various layers, including matrix multiplication, normalization (LLMs use layer normalization [5]), attention, and GeLU. The input to an LLM is an embedding matrix x of dimensions $s \times d$, where s is the length of text tokens, d denotes the hidden dimension size of the LLM. After x passes through the initial LLM transformer block, the output is a matrix f with dimensions $s \times d$. This matrix f is then passed to the next transformer block. The process repeats through all $\mathbf{L}$ blocks. The detailed structure and computation involved in each transformer block is illustrated in Appendix A.

Real number calculations in ML models pose significant challenges for ZKP. Most ZKP approaches for ML [10, 18, 19, 36, 41, 42, 57] utilize quantization to convert real numbers to integers, making computations easier to prove. We adopt the quantization approach from zkCNN [41], where a real number x is mapped to an integer q using the formula $x = S(q - z)$. S denotes a floating-point number representing the quantization scale, and z is an integer known as the zero-point. All values of LLM parameters and intermediate results in computation are expressed as Q-bit integers q. The scale S and zero-point z are typically shared within a NN layer or tensor.

When two numbers are added and their scales are the same, we add their integer parts: $x_1 + x_2 = S(q_{x_1} - z_1 + q_{x_2} - z_2)$. However, when multiplying two numbers, we must perform **rescaling** to avoid overflow. Specifically, when multiplying $x = S_x q_x$ and $y = S_y q_y$ to obtain $z = S_z q_z$ (in this and subsequent derivations, we neglect the zero-point for simplicity), we compute $q_z = \text{round}\left(\frac{C_1}{C_2} \cdot q_x \cdot q_y\right)$, where C_1 and C_2 are Q-bit integers that ensure $\frac{C_1}{C_2} \approx \frac{S_x \cdot S_y}{S_z}$. This approach allows floating-point addition and multiplication to be approximated through integer computation.

2.2 GKR for ML-friendly circuit

In our system, the computation steps of the quantized LLM are converted into constraints. And relations to prove are derived from constraints. These relations generally fall into two types. The first type is arithmetic relations, composed of additions and multiplications. We prove the arithmetic relations by converting them into an arithmetic circuit and using the GKR protocol [30]. The second type is non-arithmetic relations, such as range relations and exponentiation relations. They are proved by the Lasso protocol [55] introduced in Section 2.3.

The GKR protocol [30] uses the sumcheck protocol [43] as a building block to prove circuits consisting of addition and multiplication gates. Each gate takes at most two inputs from the previous layer and computes the result. More details on sumcheck and GKR protocols can be found in Appendix C.

GKR protocol is widely adopted in ZKP due to its efficiency. However, the classical GKR protocol has a significant limitation under many ML scenarios: The weight matrices of ML models are committed in the input layer. Since each gate only receives input from the preceding layer, these weights must be relayed across the circuit to reach the desired layer. This process introduces numerous additional gates, leading to high computational overhead.

Define identity function $eq(x,y)$, which takes two bit-strings x and y of the same length as input. $eq(\cdot,\cdot)$ returns 1 when $x=y$; otherwise returns 0. Suppose $\tilde{eq}(\cdot,\cdot)$ is the multi-linear extension of $eq(\cdot,\cdot)$. $\tilde{eq}(x,y)=\prod_{i=1}^{\ell}((1-x_i)(1-y_i)+x_iy_i)$.

Definition 1 (Multi-linear Extension [13]). *Let $V:\{0,1\}^{\ell}\to\mathbb{F}$ be a function. The multi-linear extension of V is the unique polynomial $\tilde{V}:\mathbb{F}^{\ell}\to\mathbb{F}$ such that $\tilde{V}(x_1,x_2,...,x_{\ell})=V(x_1,x_2,...,x_{\ell})$ for all $x_1,x_2,\ldots,x_{\ell}\in\{0,1\}$. $\tilde{V}$ can be expressed as $\tilde{V}(x_1,x_2,...,x_{\ell})=\sum_{b\in\{0,1\}^{\ell}}\tilde{eq}(x,b)\cdot V(b)$.*

To resolve the above limitation of GKR, [41,72] introduced an elegant circuit construction alongside an extension of the GKR protocol. They modified the definitions of addition and multiplication gates to enable them to receive input either from the preceding layer or from the input layer directly. This reduces the need to relay values from the input layer. For more details, please refer to [41]. This particular circuit configuration is referred to as a "machine learning-friendly circuit" (ML-friendly circuit).

Following the notations in [41], we define a function $V_i:\{0,1\}^{s_i}\to\mathbb{F}$ that takes a binary string b and returns the output of gate b in layer i. With this definition, V_0 corresponds to the input layer, and V_d corresponds to the circuit output. We denote the multi-linear extension of V_i as $\tilde{V}_i$. We also denote the subset of values in the input layer that connect to the i-th layer as $V_{i,in}$, with its multi-linear extension denoted as $\tilde{V}_{i,in}(\cdot)$.

At the i-th circuit layer, the prover and verifier invoke the sumcheck protocol to reduce two evaluations of $\tilde{V}_i$ to two evaluations of $\tilde{V}_{i-1}(\cdot)$ and two evaluations of $\tilde{V}_{i,in}(\cdot)$.

When reaching the input layer, the verifier has received two evaluations of $\tilde{V}_{i,in}(\cdot)$ for the input at each layer. To combine these evaluations into a single evaluation of the multi-linear extension of the input $\tilde{V}_{in}(\cdot)$, the prover and verifier invoke the sumcheck protocol. For the details of definition of addition gates, multiplication gates, and the extended GKR protocol, please refer to Appendix D.

2.3 Lookup protocol and range proof support

Lookup protocols are key components in ZKPs to prove non-arithmetic relations. They allow the prover to convince the verifier that a secret vector is contained within a public table. Specifically, for a vector $\mathrm{H}\in\mathbb{F}^m$, the prover proves that each element of H exists within $\mathrm{T}\in\mathbb{F}^n$, where T is a public table of size n, and m is the number of queries to the table.

In our approach, we utilize lookup protocols to prove exponentiation relations and range relations. When verifying the range relation $\ell_i\le x_i\le r_i$ for $i=0,1,\ldots,m-1$, we can transform it into proving $0\le x_i-\ell_i\le t$ and $0\le r_i-x_i\le t$. ℓ_i and r_i are integers, and t is defined as the maximum value of $r_i-\ell_i$. We then construct a lookup table T containing numbers $0,1,\ldots,t$. Using the lookup protocol, we check that all $x_i-\ell_i$ and r_i-x_i are in the table T, thus proving the m range relations. In our scenario, t may be quite large, potentially exceeding 2^{32}, resulting in a huge table size. Such large tables are infeasible for most existing lookup table approaches [16,23,69,70], as these approaches either require prover time linear in the table size n or need preprocessing time of $O(n\log n)$.

However, a recent advancement in lookup protocols, Lasso [55], addresses this problem. When the table is structured—meaning that the query to the table can be decomposed into sub-queries to much smaller subtables—the prover's cost in Lasso, for any integer parameter $c>1$, is dominated by committing to $3\cdot c\cdot m+c\cdot n^{1/c}$ field elements. The table we use in our scheme to check range relations satisfies the structured table requirement in Lasso. Moreover, the exponentiation table utilized in our scheme is relatively small. Therefore, despite the exponentiation table being unstructured, Lasso achieves an efficient complexity of $O(n+m)$. For technical details of Lasso, please refer to [55]. For technical details of integrating Lasso into GKR, please refer to Appendix E.

3 Technical Overview

An LLM consists of many linear and non-linear layers. To ensure applicability in real-world scenarios, a practical ZKP system for LLMs should be able to efficiently prove both types of layers. In this paper, we design efficient proofs for linear and non-linear layers in LLMs.

Efficient proof for linear layers. Our proof of linear layers is based on the well-known protocol in [58] to prove matrix multiplication. This protocol is adopted directly in many

existing ZKP for ML systems [6,41,57]. It proves the multiplication of matrix A with shape $n \times m$ and matrix B with shape $m \times k$ results in matrix C. This protocol achieves a theoretical complexity of $O(nm+mk)$, which is optimal. However, we identified significant opportunities to improve its practical implementation by notably reducing the field operations required by the protocol.

The time cost of [58] is dominated by computing the bookkeeping table for the input matrices A and B. We propose a grouping algorithm to accelerate the computation of the bookkeeping table. Grouping performs many dot products between input matrix elements and precomputed $\tilde{eq}$ evaluations, effectively halving field multiplications.

Further optimization opportunities arise from leveraging the properties of matrix elements in our scenario. Due to the requirement of sumcheck, the matrix dimensions are padded to powers of 2 with zero. During each dot product computation, we skip these zero elements, saving numerous field operations. Additionally, due to quantization, all matrix field elements have relatively small values. Thus we accelerate computation by precomputing field multiplication results.

As a result of these optimizations, our final implementation of bookkeeping table computation for matrix A is bottlenecked by $n \cdot m$ field additions. In contrast, the classical implementation in [41,58] consumes $2^{\lceil \log n \rceil + \lceil \log m \rceil + 1}$ field multiplications and $2^{\lceil \log n \rceil + \lceil \log m \rceil}$ field additions. In many cases, this amounts to approximately $4n \cdot m$ field multiplications. Given that field multiplications are several times more expensive than field additions [27,45], our optimization is estimated to reduce the overhead of [58]'s protocol by nearly an order of magnitude.

Efficient proof for non-linear layers. Non-linear functions in LLMs are composed of division, square root, and exponentiation operations, which are not ZKP-friendly. Previous work [31,63] proved these operations with hundreds or even thousands of relations, thus incurring high overhead. To resolve this inefficiency, inspired by the idea of checking advice correctness in [26,28], we propose to check division, square root, and exponentiation in one non-arithmetic relation.

Take division as an example. The division of quantized x, y is $z = \mathsf{round}(\frac{C_1}{C_2} \cdot \frac{q_x}{q_y})$, where C_1, C_2 are integers such that $\frac{C_1}{C_2} \approx \frac{S_x}{S_y \cdot S_z}$. We derive $C_2 q_y(2q_z - 1) \leq 2C_1 \cdot q_x < C_2 q_y(2q_z + 1)$. Similarly, square root can be transformed into **one** range relation. For exponentiation operations, we prepare a lookup table containing all possible evaluations of the exponentiation result. Then, each exponentiation operation can be verified through a single query to the lookup table. The range relations and lookup queries above are proved with the state-of-the-art Lasso [55] lookup protocol. For arithmetic relations, we compose them into an arithmetic circuit and prove them with the ML-friendly GKR protocol introduced in Section 2.2.

Now we are ready to prove each LLM layer using the previously designed building blocks. To further maximize the prover efficiency, we introduce optimization methods at both the constraint and circuit levels.

Constraint fusion. Our first optimization idea is to merge adjacent rounding constraints to reduce the number of range relations to be proved. We discovered that proving range relations is the bottleneck in our system, as it is more costly than proving arithmetic relations. To address this issue, we combine computations across adjacent rounding constraints before rounding. Under appropriate conditions, merging these constraints reduces proving overhead by saving range relations to prove introduced by rounding. We developed rules to automatically determine the optimal merging strategies for different LLM layers.

For example, during the computation of the normalization layer, given an input vector x with length n, its mean $\mu = \frac{\sum_{i=1}^{n} x_i}{n}$ and variance $\sigma = \sqrt{\frac{(\sum_{i=1}^{n}(x_i-\mu)^2)}{n}}$. Assume we want to prove σ is correct. [31,42] proves the following constraints: (1) $q_\mu = \mathsf{round}(\frac{\sum_{i=1}^{n} q_{x_i}}{n})$ (2) $q_t = \mathsf{round}(\frac{(\sum_{i=1}^{n}(q_{x_i}-q_\mu)^2)}{n})$ (3) $q_\sigma = \mathsf{round}(\sqrt{q_t})$. Based on these constraints, their systems can prove q_σ in **three** range relations.

However, leveraging the idea of merging adjacent rounding constraints, we can reduce the range relations that need to be proven to **one**. Specifically, we merge the constraints on q_μ and q_t, and substitute them into the last constraint on q_σ. We first turn q_μ and q_t into the analytic form: $q_\mu = \frac{\sum_{i=1}^{n} q_{x_i}}{n}$ and $q_t = \frac{(\sum_{i=1}^{n}(q_{x_i}-q_\mu)^2)}{n}$. By substituting them into q_σ, we have $q_\sigma = \mathsf{round}(\sqrt{\frac{(\sum_{i=1}^{n}(q_{x_i}-\frac{\sum_{i=1}^{n} q_{x_i}}{n})^2)}{n}})$. We eventually derive the following range relation: $n(2q_\sigma - 1)^2 \leq \sum_{i=1}^{n}(nq_{x_i} - \sum_{i=1}^{n} q_{x_i})^2 < n(2q_\sigma + 1)^2$.

Circuit squeeze. We observe that our circuit, like existing GKR-based ZKP systems for ML [6,39,41], cannot effectively utilize multi-thread parallelism. This inefficiency comes from two main factors: (1) the small size of each circuit layer and (2) the Fiat-Shamir Transformation [20], which restricts parallelization to be conducted within each layer. As a result, significant time is wasted on synchronization due to the small sizes handled by each sumcheck.

Our optimization design stems from the fact that the computation order of the GKR protocol follows the circuit architecture. Thus, we have to reorganize the circuit architecture to improve parallelism opportunity. Our key observation is that due to our advice design, the subcircuits checking different constraints have no topological dependence. We can squeeze different subcircuits into the same set of layers leveraging this topological independence.

The remaining challenge is how to squeeze multiple matrix multiplication layers. This challenge arises because matrix multiplications are proven through the protocol in [58] rather than GKR-style sumcheck. To address this, we design a new protocol that simultaneously checks the correctness of multiple matrix multiplications based on sumcheck. Circuit

squeezing substantially enhances the performance of the LLM circuit by boosting the utilization of multi-thread parallelism.

4 Efficient Proof for LLM

4.1 Constraints for LLM layers

Utilizing the quantization scheme introduced in Section 2.1, we derive the constraints for different LLM layers.

Matrix multiplication. For matrix a and weight W, the matrix multiplication layer is computed as $b = a \cdot W$. The result matrix b needs to be rescaled to avoid overflow. The constraint is $q_b = \mathsf{round}(\frac{S_a \cdot S_W}{S_b} a \cdot W)$.

Attention. The attention layer takes h matrix triples (Q_i, K_i, V_i) from each head as input and computes: $P = \underset{i=1,\cdots,h}{Concat}(Softmax(\frac{Q_i K_i}{\sqrt{d}})V_i)$. The computation of each head is data-independent before concatenation; for simplicity, we only write the constraints for a single head, and neglect the subscript for each head. Based on the computation process of Attention layer illustrated in Equation 7 in Appendix, our constraints are listed as follows:

$$
\begin{aligned}
q_C &= \mathsf{round}(\frac{S_Q \cdot S_K}{S_C} q_Q \cdot q_K) \\
q_E &= \mathsf{round}(\frac{S_C}{S_E \sqrt{d}} \cdot q_C) \\
q_{E_{max,i}} &= \max q_{E_{ij}}, \text{ for } i = 1, \cdots s \\
q_{e_{ij}} &= \exp(q_{E_{max,i}} - q_{E_{ij}}), \text{ for } i = 1, \cdots s, j = 1, \cdots, i \\
q_{s_{ij}} &= \mathsf{round}(\frac{q_{e_{ij}}}{S_s \sum_{j=1}^{i} q_{e_{ij}}}), \text{ for } i = 1, \cdots s, j = 1, \cdots, i \\
q_P &= \mathsf{round}(\frac{S_s \cdot S_V}{S_P} q_s \cdot q_V)
\end{aligned}
\tag{1}
$$

where $S_Q, S_K, S_V, S_C, S_E, S_s, S_P$ are the scales of q_Q, q_K, q_V, q_C, q_E, q_s and q_P, respectively. For GeLU and Normalization layers, their constraint derivations are similar to attention. Due to space limitations, we present them in Appendix G.

Due to the similarity in computational workflows, the constraints proposed in the concurrent work by Lu et al. [42] share some similarities with ours. The key distinction lies in our novel zk-friendly approximation for GeLU, referred to as z-GeLU, detailed in Appendix I. In contrast, Lu et al. [42] adopt the approximation from I-BERT [37] directly.

4.2 Efficient proof for linear layers

After deriving the LLM constraints, we now consider how to efficiently prove them. We first consider linear layers since they account for the majority of computation in plain-text LLM inference. We integrate the dedicated protocol for matrix multiplication in [58] into the ML-friendly GKR framework in a manner identical to that of [41]. Although the protocol achieves optimal prover time in theory, we identify significant opportunities to improve its concrete efficiency.

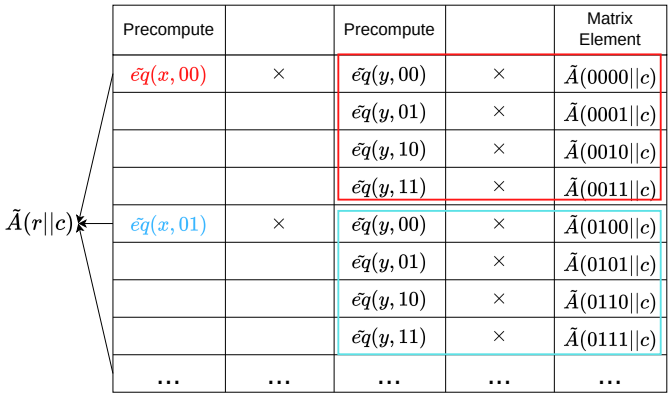


Figure 2: Sketch of how grouping works in computing each entry of bookkeeping table $\tilde{A}(r||c), c \in \{0,1\}^{\lceil \log m \rceil}$.

4.2.1 Analysis of classical implementation of [58]

The protocol of [58] is as follows: Denote the two input matrices as A, B, and the output matrix as C. Assume A, B, C have original shapes $n \times m, m \times k, n \times k$, respectively. Due to requirements of sumcheck, these matrices are all padded to powers of 2 in both width and height. [58]'s protocol uses the sumcheck protocol to reduce the claim on $\tilde{C}(z)$ to claims on $\tilde{A}(\cdot), \tilde{B}(\cdot)$. z is a random point chosen by the verifier. Denote $a||b$ as the concatenation of vectors a and b. Let $z = r||r'$, where $r \in \mathbb{F}^{\lceil log n \rceil}, r' \in \mathbb{F}^{\lceil log k \rceil}$. By the property of matrix multiplication, we have $\tilde{C}(r||r') = \sum_{c \in \{0,1\}^{\lceil log m \rceil}} \tilde{A}(r||c)\tilde{B}(c||r')$.

Before running the sumcheck protocol on this equation, the prover first computes the bookkeeping tables: $\tilde{A}(r||c)$ and $\tilde{B}(c||r')$ for all $c \in \{0,1\}^{\lceil \log m \rceil}$. Existing methods [41, 58] leverage the following equation to iteratively compute $\tilde{A}(r||c)$: $\tilde{A}(r_1, \cdots, r_i, b_{i+1}, \cdots, b_{\lceil \log n \rceil}||c) = (1 - r_i)\ \tilde{A}(r_1, \cdots, r_{i-1}, 0, b_{i+1}, \cdots, b_{\lceil \log n \rceil}||c) + r_i\ \tilde{A}(r_1, \cdots, r_{i-1}, 1, b_{i+1}, \cdots, b_{\lceil \log n \rceil}||c)$. This is identical to the "memorization" method in [60] to evaluate multi-linear extension. The method's computation cost for each c is $2 \cdot (2^{\lceil \log n \rceil - 1} + \cdots + 1) \approx 2^{\lceil \log n \rceil + 1}$ field multiplications and $2^{\lceil \log n \rceil}$ field additions. Therefore, the total cost for computing $\tilde{A}(r||\cdot)$ is $2^{\lceil \log n \rceil + \lceil \log m \rceil + 1}$ field multiplications and $2^{\lceil \log n \rceil + \lceil \log m \rceil}$ field additions.

4.2.2 Optimizations based on grouping and matrix element properties

Next, we will introduce our efficient algorithm for computing the bookkeeping table.

Efficient computation by grouping. By the definition of multi-linear extension, we have $\tilde{A}(r||c) = \sum_{b \in \{0,1\}^{\lceil log n \rceil}} \tilde{eq}(r,b)\tilde{A}(b||c)$. Without loss of generality, assume $\lceil \log n \rceil$ is even. Let $r = x||y, b = L||R$, where $x, y, L, R \in \mathbb{F}^{\frac{\lceil \log n \rceil}{2}}$. We observe that $\tilde{eq}(\cdot)$: $\tilde{eq}(r,b) = \tilde{eq}(x,L) \cdot \tilde{eq}(y,R)$. Leveraging this property, we have:

$$\tilde{A}(r||c) = \sum_{L\in\{0,1\}^{\frac{\lceil \log n\rceil}{2}}} \tilde{eq}(x,L)\cdot\Big(\sum_{R\in\{0,1\}^{\frac{\lceil \log n\rceil}{2}}} \tilde{eq}(y,R)\cdot\tilde{A}(L||R||c)\Big) \tag{2}$$

Computing using the above equation is more efficient. The procedure is as follows: All $\tilde{eq}(x,\cdot)$ and $\tilde{eq}(y,\cdot)$ are precomputed once in $O(2^{\frac{\lceil \log n\rceil}{2}})$ field operations. Imagine there are $2^{\frac{\lceil \log n\rceil}{2}}$ groups represented by different L. Computing Equation 2 could be viewed as conducting a dot product between $\tilde{eq}(y,\cdot)$ and $\tilde{A}(L,\cdot,||c)$ in each group; then the result of the dot product in each group forms a vector; at last this vector performs a dot product with $\tilde{eq}(x,\cdot)$ to obtain $\tilde{A}(r||c)$. A sketch of our grouping-based algorithm is shown in Figure 2.

Skipping computation by leveraging sparsity. The benefit of leveraging grouping is that we can efficiently exploit the properties of matrix A, since the elements of A are directly used in each inner dot product. Specifically, during the computation of each inner dot product, we can skip over elements in A that are zeros, thereby reducing the number of field operations required by our method. This opportunity arises because matrix A is padded with many zero elements to satisfy the requirement of sumcheck.

Saving field multiplication by precomputing. We also observe another important property of matrix A is that elements in A are all quantized numbers. This means that all of their value satisfy $\in [0,2^Q]$, where $2^Q << |\mathbb{F}|$. Thus, the multiplication between each $\tilde{eq}(y,\cdot)$ and different $\tilde{A}(b||c)$ only obtains at most 2^Q+1 different results. Furthermore, these possible results are $\{0,1,\cdots,2^Q\}$ times of $\tilde{eq}(y,b_{\frac{\lceil \log n\rceil}{2}+1}\cdots b_{\lceil \log n\rceil})$. We can efficiently derive all these results by only performing simple field additions on $\tilde{eq}(y,b_{\frac{\lceil \log n\rceil}{2}+1}\cdots b_{\lceil \log n\rceil})$.

Cost analysis. The full algorithm for computing the bookkeeping table for matrix A is shown in Algorithm 1. The two bottlenecks in our computation is precomputing the field multiplication results in line 10 and performing the inner dot product based on precomputed results in line 18. These two parts each correspond to $2^{\lceil \frac{\log n}{2}\rceil+Q}$ field additions and $n\cdot m$ field additions. Our optimizations can significantly mitigate computation overhead compared with existing implementation [58]. Take computing the bookkeeping table of the matrix of shape 3072×768 in the GPT-2 feedforward layer as an example. Based on previous analysis, the classical method costs 8.4×10^6 field multiplications and 4.2×10^6 field additions. Substituting $Q=16$, Our optimized method costs $n\cdot m+2^{\frac{\lceil \log n\rceil}{2}+Q}\approx 4.5\times 10^6$ field additions. Assume one field multiplication is 5 times cost of one field addition. Our method is $10.2\times$ more efficient than the classical method. The optimization techniques presented in our work could also be used to accelerate the evaluation of multi-linear extensions under other scenarios in ZKP.

4.3 Efficient proof for non-linear layers

Proving non-linear layers in the LLM also poses a significant challenge. The challenge comes from these layers invoking ZKP-unfriendly operations such as division, square root, and exponentiation. Existing approaches either simulate these operations in the circuit—using binary decomposition and floating-point emulation [41, 63] or proving numerical iterations [17, 31]. These approaches all incur a large number of relations to prove and significant overhead. For example, [63] requires thousands of relations to prove division and exponentiation operations.

Recent advances in ZKP systems [4, 26] leverage advice to efficiently prove non-arithmetic operations. [4] proves ZKP-unfriendly bitwise operations like XOR by taking the result as advice and querying it in the lookup table. This is much cheaper than simulating XOR in a binary circuit. Inspired by these works, we propose using the computation result of division, square root, and exponentiation as advice and check the correctness of the computation. This is much cheaper than simulating the computation in arithmetic relations.

For the division operation $z=\frac{x}{y}$, the actual constraint is $q_z=\mathsf{round}(\frac{C_1}{C_2}\cdot\frac{q_x}{q_y})$. We approximate $\frac{S_x}{S_y\cdot S_z}$ using $\frac{C_1}{C_2}$, C_1 and C_2 are integers. C_1, C_2 are obtained by searching in an interval $[1,2^Q]$ and finding the nearest fraction to $\frac{S_x}{S_y\cdot S_z}$. By the property of rounding, we have $q_z-\frac{1}{2}\le\frac{C_1}{C_2}\cdot\frac{q_x}{q_y}<q_z+\frac{1}{2}$. Thus, $C_2q_y(2q_z-1)\le 2C_1\cdot q_x\le C_2q_y(2q_z+1)-1$. Similarly, we can convert proving square root operation $q_y=\mathsf{round}(\sqrt{q_x})$ into a range relation: $(2q_y-1)^2\le 4q_x\le(2q_y+1)^2-1$. Therefore, for division and square root operations, we can transform checking their correctness into a single range relation. For proving exponentiation operations, we construct a lookup table containing pairs of input and output $(q_t,\mathsf{round}(\frac{e^{S_t\cdot q_t}}{S_z}))$. Exponentiation operation $q_z=\mathsf{round}(\frac{e^{S_t\cdot q_t}}{S_z})$ can be proved by querying (q_t,q_z) in this table, checking whether the pair belongs to the table. By leveraging the Lasso lookup protocol, we can efficiently prove these non-arithmetic relations. Experiment results show that our designs save the time cost of proving non-linear layers by several orders.

Proving non-linear layers in the LLM also involves proving some arithmetic relations. We assemble these relations into an arithmetic circuit and then prove the circuit using the ML-friendly GKR protocol.

4.4 Putting everything together

With the building blocks presented in the previous sections, we construct a ZKP protocol for LLM. The values in the LLM circuit's input layer consist of public input and secret witnesses. Following previous works [42, 57], our LLM circuit takes the input embedding matrix as public input. The witness includes model weight, intermediate results obtained

by rounding, and queries to every lookup table. The prover commits to all witness values. The commitment to weight is done once in preprocessing and reused across all users (see Appendix J). The intermediate results and lookup query values depend on user input. Thus, their commitment are computed for each user separately. The verifier runs the Lasso protocol with the prover to check whether the non-arithmetic relations are satisfied. Then, the prover and verifier run the GKR protocol on the arithmetic circuit, checking all the arithmetic relations between the variables. For the specified matrix multiplication layer, it is proven by the dedicated protocol in [58] leveraging our optimized algorithm in Section 4.2. After the GKR protocol, the prover and verifier reduce the claim to the correctness of a claim at the input layer. They open the commitment to the input layer values to determine whether the protocol passes. The compliance of all the relations ensures the correctness of the whole LLM computation. The entire protocol is detailed in Protocol 4 in Appendix C. The proofs of soundness and completeness for the zkGPT protocol are provided in Appendix F. The zero-knowledge property can be incorporated into the protocol by using masking polynomials in a manner similar to that described in [72].

5 Constraint level optimization

In the previous sections we have built a non-interactive ZKP system for LLM. To ensure better applicability in real-world scenarios, we illustrate how we optimize our system at the constraint level in this section.

Previous works most related to our research [36, 41] introduced a technique to fuse the rescaling division after convolution with the subsequent ReLU constraint, reducing the number of relations to prove. However, they did not analyze constraint fusion opportunities in general cases, limiting the method's applicability to other NNs.

5.1 Constraint level optimization opportunity

In Section 4.1, we derive the constraints for each LLM layer. Rounding is incorporated in these constraints to ensure that intermediate results remain within the integer domain. However, the mapping of LLM layers to constraint sets is not unique; different approximations can lead to multiple sets of constraints, each with varying proving costs. These costs are largely influenced by the number of roundings, as rounding introduces range relations to prove, which are more computationally expensive than proving arithmetic relations.

Constraint fusion draws inspiration from operator fusion [11, 47], a well-known method for accelerating NN executions. The core idea of constraint fusion is to consolidate various constraints into a more concise structure. In particular, the approach involves merging computations within two adjacent rounding operations, resulting in one merged rounding constraint, thereby decreasing the overhead of proving range relations.

Table 1: Merging different types of constraints. Whether merging constraint i with constraint j is profitable (denoted as P), potentially beneficial (denoted as PB), or unprofitable (denoted as UP).

First const. \ Second const.	I	II	III	IV
I	PB	PB	PB	UP
II	PB	P	P	UP
III	PB	P	P	UP
IV	UP	UP	UP	UP

However, assessing the benefits of merging constraints is challenging. Some merges can reduce the number of roundings at free cost. Some may even significantly increase overhead. Additionally, some merges might not immediately reduce roundings but could pave the way for more effective merges later. The diverse and varying forms of constraints across computations further complicate manual analysis.

5.2 Constraint merging rules

Constraint classification. To better analyze the benefit of merging constraints inside the constraint, we classify the existing constraints with rounding into several types:

- Type-I: constraints that only contain arithmetic computation (Add: $q_y = q_{x_1} + q_{x_2}$, Mul: $q_y = q_{x_1} * q_{x_2}$).
- Type-II: constraints that contain division mixed with arithmetic computation. A typical representation is $q_y = round(\frac{S_{x_1} S_{x_2}}{S_y} q_{x_1} q_{x_2})$ (we omit the addition for simplicity).
- Type-III: constraints that contain square root mixed with division and arithmetic computation. The simplified representation is $y = round(\sqrt{t})$, where t is Type-I or Type-II constraint.
- Type-IV: constraints that contain exponentiations.

Type-I and Type-II constraints are used in all LLM layers. Type-III constraint is used in Normalization. Type-IV is employed by softmax function in attention layer.

Table 1 presents all potential constraint merging combinations. A profitable merge results in performance gains by reducing the number of rounding constraints while introducing minimal or no overhead on the arithmetic circuit side. A potentially beneficial merging offers opportunities for improvement, though it may not directly reduce rounding or increase proving costs. An unprofitable merge harms performance as the introduced overhead outweighs the savings from reducing the number of rounding constraints. We perform constraint merging only for profitable or potentially beneficial cases, leaving unprofitable ones untouched. The constraint

type may change with fusion, but it remains one of the four fundamental types. In the following, we elaborate on profit analysis of representative combinations in Table 1.

Merging Type-II with itself is profitable. Merging Type-II constraints reduces the number of range relations to prove, and does not introduce extra overhead on the arithmetic circuit side. A detailed analysis of the impact of constraint fusion on the number of required multiplication and addition gates can be found in Lemma K.1 in Appendix K.

Merging Type-II with Type-III is profitable. For instance, assume the first constraint is $q_y = \text{round}\left(\frac{C_{x_1x_2}}{C_y} q_{x_1} q_{x_2}\right)$, where $C_{x_1x_2}$ and C_y are integers. And the second constraint is $q_z = \text{round}(\sqrt{q_y})$. After merging, the rounding constraint derives one range relations forming $(2q_z-1)^2 C_y \leq 4C_{x_1x_2} q_{x_1} q_{x_2} \leq (2q_z+1)^2 C_y$. Although merging introduces small overhead due to the arithmetic circuit (square), it is minimal compared to the rounding saved.

Merging Type-III with Type-II or Type-III is profitable. Consider the first constraint $q_y = \text{round}(\sqrt{q_x})$ and the second constraint $q_z = \text{round}\left(\frac{C_{yy'}}{C_z q_y} q_{y'}\right)$, where $C_{yy'}$ and C_z are integers and $\frac{C_{yy'}}{C_z} \approx \frac{S_y S_{y'}}{S_z}$. The rounding constraint derives to $(2q_z-1)^2 C_z^2 q_x \leq 4C_{yy'}^2 q_{y'}^2 \leq (2q_z+1)^2 C_z^2 q_x$. Although extra square computation introduce arithmetic overhead, it is still minimal compared to the rounding saved. Similar with merging Type-III with Type-II, merging Type-III with Type-III is profitable too.

Merging Type-I with others (except Type-IV) or merging others (except Type-IV) with Type-I is potentially beneficial. It is easy to see that merging Type-I with other constraints does not introduce extra gate or change the rounding constraints since Type-I constraint only contain arithmetic computation. While merging does not directly eliminate rounding, it creates opportunities for further merging with adjacent constraints, potentially reducing overall complexity and overhead.

Merging with Type-IV or Merging Type-IV with others is unprofitable. For the first case, since the query for the lookup table must be an integer, rounding can not be eliminated for this case. For the second case, it prevents reusing the same exponentiation table across different LLM layers, leading to a high commitment cost for multiple extra tables and diluting the final performance.

Sequential merging strategy. Given a group of constraints, we sequentially merge adjacent constraints when the merging is either profitable (P) or potentially beneficial (PB). This merging process is repeated until no further profitable merges are possible. The final set of constraints remains consistent regardless of the merging order.

Error analysis. Assume we have constraint $q_y = \text{round}(F(q_x))$, $q_z = \text{round}(G(q_y))$, F, G are arbitrary real number functions. Denote the merged constraint as $q_{z'} = \text{round}(G(F(q_x)))$. Denote $p = G(F(q_x))$. We find that $|q_{z'} - p| \leq |q_z - p|$. This is because of, $q_{z'}$ equals $\text{round}(p)$. Thus it is the integer that minimizes $|q_{z'} - p|$. Since q_z is also belongs to the integer domain, $|q_{z'} - p| \leq |q_z - p|$ must hold. This means that constraint fusion never increases the numerical error of quantization. In contrast, it often reduces this error.

5.3 Optimized constraint after merging

After conducting sequential merging for each LLM layer, we can get appropriate constraint groups.

Attention. After merging q_C with q_E and $q_{s_{ij}}$ with $q_{P_{ij}}$, we are able to effectively reduce the number of rounding constraints of attention from $\frac{3}{2}hs(s+1)+sd$ to $\frac{1}{2}hs(s+1)+sd$. This reduction results in a nearly 30% decrease in the total number of rounding constraints required compared to the original attention.

$$
\begin{aligned}
q_E &= \text{round}\left(\frac{S_Q \cdot S_K}{S_E \sqrt{d}} \cdot q_Q \cdot q_K\right) \\
q_{E_{max,i}} &= \max q_{E_{ij}}, \text{ for } i = 1, \cdots s \\
q_{e_{ij}} &= \exp(q_{E_{max,i}} - q_{E_{ij}}), \text{ for } i = 1, \cdots s, j = 1, \cdots, i \\
q_{P_{ij}} &= \text{round}\left(\frac{S_V (q_e \cdot q_V)_{ij}}{S_P \sum_{k=1}^{i} q_{eik}}\right)
\end{aligned}
\tag{3}
$$

Constraint merging is performed similarly for GeLU and Normalization layers, reducing their number of rounding constraints by around 50%. We present the details in Appendix H.

For a detailed explanation of the rounding savings effect from constraint fusion, readers are referred to Table 10 in the Appendix.

6 Circuit Optimization

A persistent challenge in ZKP is maximizing the utilization of computational resources during proof generation. ZKP parallelization [40, 64, 68] offers a promising solution. However, existing ZKP systems for ML [6, 39, 41, 42, 63] fail to fully exploit parallelism. We address this limitation by leveraging the insight that the LLM execution trace is already known during proving. Consequently, proving LLM execution does not need to follow the sequential order of LLM computation.

6.1 Circuit level optimization opportunity

We discover that existing ZKP for ML approaches [6, 31, 39, 41, 42] achieve poor speedup under multi-thread parallelism. For sVOLE-based systems [31, 42], high communication overhead undermines the parallelism opportunity. As profiled in emp-zk [12], doubling computational resources yields less than 10% speedup under limited network bandwidth.

Although GKR-based systems [6, 39, 41] do not require network communication, they still suffer from low parallelism utilization. In the GKR protocol, parallelization is conducted

within the sumcheck protocol of each layer. Different entries of the bookkeeping table are computed using multi-threaded parallelism by leveraging the "distributed sumcheck" paradigm proposed in [68]. The parallelization is limited within the sumcheck of each layer due to the Fiat-Shamir Transformation. Note that NNs typically consist of many layers, each with several constraints. Thus, for proving an NN circuit, the computation required for sumcheck in each layer is relatively small. Consequently, the synchronization process occupies a large proportion of computation time, leading to poor speedup.

To demonstrate our findings, we profiled the sumcheck protocol using different thread counts and various input sizes in Figure 5 in Appendix. Figure 5a shows that the sumcheck protocol has a relatively low acceleration ratio when the input size is small. Figure 5b shows that as the size of the sumcheck increases, the ratio of resource utilization improves significantly. From these results, we infer that increasing the number of gates per layer is necessary to better utilize parallelism for GKR circuits.

Based on previous observations, our optimization is to squeeze the circuit into a wider and shallower shape. However, circuit squeezing faces the challenge that GKR circuits naturally impose layer-wise dependency. If variable b is computed based on variable a, then b has to be placed in subsequent layers of a. This makes it seem impossible to squeeze the circuit without making large changes to the circuit first.

6.2 Acceleration by circuit squeeze

The core idea of circuit squeeze is to break the layer-wise dependency based on commitment. Then, multiple subcircuits are topologically independent and could be squeezed into the same set of layers. Circuit squeezing helps reduce the circuit depth and increase the number of gates per layer, offering much better opportunities for parallelism.

Zero overhead circuit squeeze. Inspired by the construction of the Plonk [24] proof system, which has no sequential dependency in proving different constraints, we discover that committing intermediate results in the input layer can eliminate the topological dependency in the GKR circuit.

Since each subcircuit checks constraints using advice to reduce overhead, no additional commitment effort is required to squeeze these subcircuits. Moreover, as each subcircuit directly takes input from the input layer, they are topologically independent. Leveraging this independence, multiple subcircuits can be combined into a single set of layers, thereby restructuring the circuit into a wider and shallower form.

In general, we can apply circuit squeezing as follows: for each circuit layer that checks the correctness of a rounding constraint, starting from the i-th layer ($i > 1$), we move each gate originally in layer j to the $(j-i+1)$-th layer, while maintaining the connections between these gates. For our LLM circuit, since intermediate results are already committed as advice, squeezing incurs zero overhead.

Circuit squeeze for matrix multiplication. However, the above mentioned technique to merge layers comprised of normal GKR gates is not applicable to matrix multiplication layers. Because these layers are proved using the dedicated protocol [58]. And the protocol in [58] only supports proving one matrix multiplication in each invocation.

To solve this problem, we design a new protocol generalizing [58] to support proving multiple matrix multiplication. Assume the circuit layer we consider is layer 1, and the number of matrix multiplications is M. After execution of GKR on previous layers, we have claims $\tilde{V}_1(r_1)$ and $\tilde{V}_1(r_2)$.

Inspired by the method in [6] proving multi-channel convolution in sumcheck, we can check multiple matrix multiplications by concatenating the matrices and checking the "batched" matrix multiplication relation using sumcheck. Denote the two matrices involving in the i-th matrix multiplication as A_i, B_i, the output of the $i-$th matrix multiplication as C_i. Assume all A_i are padded to the same shape, with each dimension to the power of 2. B_i is padded in the same manner. Denote A_i has shape $n \times m$, B_i has shape $m \times k$, C_i has shape $n \times k$. Assume the number of matrix multiplications M is also padded to the power of 2. Denote the concatenation of A_i as $A \in \mathbb{F}^{M \times n \times m}$, $B \in \mathbb{F}^{M \times m \times k}$ is defined similarly. $V_1 \in \mathbb{F}^{M \times n \times k}$ is the flattened form of the concatenation of C_i. For $\forall i \in \{0,1\}^{\log M}, j \in \{0,1\}^{\log n}, p \in \{0,1\}^{\log k}$, we have $V_1(i||j||p) = \sum_{c \in \{0,1\}^{\log m}} A(i||j||c) \cdot B(i||c||p)$.

The above equation is well-suited for proving within the sumcheck protocol. Let $r_1 = r'||r'_x||r'_y, r_2 = r''||r''_x||r''_y$, where $r', r'' \in \mathbb{F}^{\log M}, r'_x, r''_x \in \mathbb{F}^{\log n}, r'_y, r''_y \in \mathbb{F}^{\log k}$. We have the following:

$$\begin{aligned} &\alpha \cdot \tilde{V}_1(r_1) + \beta \cdot \tilde{V}_1(r_2) \\ &= \sum_{c \in \{0,1\}^{\log m}} \alpha \tilde{A}(r'||r'_x||c) \cdot \tilde{B}(r'||c||r'_y) + \beta \tilde{A}(r''||r''_x||c) \cdot \tilde{B}(r''||c||r''_y) \end{aligned} \tag{4}$$

Running sumcheck on the above equation, we can reduce claims on $\tilde{V}_1(r_1)$ and $\tilde{V}_1(r_2)$ to $\tilde{A}(\cdot)$ and $\tilde{B}(\cdot)$. The remaining challenge is whether we can generalize the efficient method to run the protocol in [58] in Section 4.2 to our new protocol. Essentially, we need to efficiently compute $\tilde{A}(r'||r'_x||c), \tilde{B}(r'||c||r'_y), \tilde{A}(r''||r''_x||c), \tilde{B}(r''||c||r''_y)$ for each $c \in \{0,1\}^{\log m}$. By symmetry, we only need to consider how to efficiently compute $\tilde{A}(r'||r'_x||c)$. By the property of multi-linear extension, we have $\tilde{A}(r'||r'_x||c) = \sum_{b' \in \{0,1\}^{\log M + \log n}} \tilde{eq}(r'||r'_x, b') \cdot \tilde{A}(b'||c)$.

The equation above is similar in form to the bookkeeping table used in the matrix multiplication protocol of [58]. Leveraging techniques similar to Algorithm 1 described in Section 4.2, we can compute the above equation efficiently. Due to space constraints, we omit the details.

7 Experiment

In this section, we present a comprehensive evaluation of our ZKP framework for LLM.

7.1 Implementations

Software. Our work is mainly implemented in C++. Our implementations of sum-check and ML-friendly GKR are based on the open source implementation of zkCNN [41]. We use Hyrax [61] as our polynomial commitment scheme because of its efficient prover time, reasonable proof size, and transparency. The prover time is $O(N)$ and the proof size and the verifier time are $O(\sqrt{N})$ for a polynomial of size N. For Lasso, we migrate the code based on its official implementation[1]. We utilize the mcl library[2] to implement operations on finite field and BN254 elliptic curve adopted in [7, 40]. BN254 provides around 100 bits of security.

Hardware. Our evaluations are performed on a server equipped with an Intel Xeon 6126 2.60GHz 16-Core CPU and 200GB of memory. We leverage 32 thread parallelization by default.

Implementation acceleration. We accelerate the prover's commitment of the circuit's input values using multi-thread parallelism. In the final step of the ML-friendly GKR protocol, sumcheck consolidates multiple input claims into one, with multi-threading used for further acceleration. Details are provided in Appendix J.

Quantization details. In our experiments, we use quantization level $Q = 16$. This means that all quantized values q belongs to range $[0, 65536]$. For ensuring value bound, the prover proves that the committed quantized weight lies within the range $[0, 2^Q]$, while the quantized input embedding is public and easy to verify. Rescaling with appropriate scaling factor is performed at each layer to ensure the computation results always fall the range $[0, 2^Q]$.

Baselines. In this paper, we compare the performance of our system with existing ZKP systems for LLM [10, 31, 42, 57]. [10] and [42] natively supports multi-thread parallelism thus we directly use their implementation for comparison. The proof system of [57] is purely based on GPU parallelism implemented by CUDA. Thus, we run it on a server with NVIDIA A100 GPU. For the VOLE-based schemes [31, 42], we evaluate performance under a standard network environment with 500 Mbps bandwidth.

7.2 Performance of zkGPT

7.2.1 Performance of linear layers

We first benchmark the performance of our efficient algorithm for accelerating proof of matrix multiplications under single thread. The results can be found in Table 2.

Table 2: Acceleration of matrix optimizations.

	$(32 \times 768) \cdot (768 \times 3072)$	Optimized	GPT-2 matrices	Optimized
Time	0.25s	0.037s	9.8s	1.5s

We can see that our algorithm does achieve very significant optimization effect compared with the original algorithm in [58]. For the case of proving matrix multiplication happening in the GPT-2 feed-forward network, our method achieves an acceleration ratio of $6.8\times$.

For a single transformer block in GPT-2, it contains 4 matrix multiplications, each weight matrix has shape $768 \times 2304, 768 \times 768, 768 \times 3024, 3024 \times 768$, respectively. We also profile the overhead of proving all of these matrix multiplications happening in GPT-2 inference in Table 2. We can see that, proving these multiplications utilizing the classical method in [58] requires about 10 seconds, which is quite considerable compared with our goal of squeezing our prover time of GPT-2 inference into one minute. Leveraging our optimizations, the time cost of proving all multiplications in GPT-2 can be reduced by $6.5\times$. This highlights the significance of our efficient method for proving matrix multiplications.

7.2.2 Overall performance

Next, we compare our system with existing ZKP systems for LLM. We also measure the LLM functionality loss caused by quantization. From Table 3, we can see that even under single thread setting and without our optimizations, our system achieves quite impressive performance compared to existing work. Compared with one of the state-of-the-art non-interactive proof schemes ZKML [10] which requires more than 1 hour to generate a proof for GPT-2, our system is $13\times$ more efficient, consuming approximately 5 minutes. The major reason is that our system is based on GKR and avoids the expensive FFT operations in Plonk-based systems. Comparing with state-of-the-art VOLE based scheme [42], though our system's prover time is a few times longer without multi-thread parallelism and optimizations, our scheme is non-interactive and the proof size is smaller by two orders of magnitude. This reduces the need of heavy communication burden for the verifier, also enabling the proof to be verified by the public.

After applying our proposed optimizations and multi-thread parallelism, the advantage of our system is amplified. Table 3 shows that our fully optimized system is $185\times$ more efficient than the state-of-the-art non-interactive scheme ZKML, and $5.1\times$ more efficient than the state-of-the-art VOLE-based scheme Lu et al [42], while our communication is $23000\times$ more efficient. Note that in MLaaS settings, the verifier normally owns much less computation power. Thus our system is much more practical in real world for verifying the integrity of MLaaS computation.

[1] https://github.com/a16z/jolt/tree/lasso

[2] https://github.com/herumi/mcl

Table 3: Time comparison of different proof systems for GPT-2 model.

Scheme	Prover time	Verifier Time	Proof size
Hao et al [31][3]	6096s		6.85 GB
ZKML [10] (32 thread)	4026s	12.1s	7.8 K
Lu et al [42][4] (32 thread)	112.3s	31.4s	2.24 GB
zkLLM [57] (6912 CUDA cores)	15.8s	0.54s	126K
Ours (Single thread) wo all opt	319.6s	0.46s	154K
Ours (32 thread) wo all opt	52.3s	0.46s	154K
Ours (32 thread) w all opt	21.8s	0.35s	101K

Table 4: Generate text quality of GPT-2 and our quantization on various datasets.

Dataset	WikiText2	PTB	LAMBADA
FP32	29.3	41.3	48.4
Our quantization	29.5	41.7	48.7

When comparing with zkLLM [57], though our concrete prover performance is 30% slower, we mainly attribute it to different hardware configurations. CPUs generally have inferior parallel computing power compared with GPUs. For the 16 core CPU we use in our experiments, it has 10.2 TFLOPS computing power in total. While for A100 GPU, its computing power is 312 TFLOPS when fully utilized [48], which is 30× stronger. This indicates that our system utilizes computation power in a much more effective way. This roots in multiple reasons (1) zkLLM proves each transformer block sequentially; Due to our circuit squeezing technique, our system has better parallelization opportunity than zkLLM. (2) zkLLM directly adopts the matrix multiplication protocol in [58], while our optimized method consumes much less field operations. (3) zkLLM did not propose techniques to reduce the number of non-arithmetic relations to prove from the constraint level. Thus our system is very likely to outperform them under the same hardware.

Since our system adopts quantization to make the LLM friendly to prove like most ZKP systems for ML, it is important to measure the functionality loss caused by quantization. Following previous works in quantization [15, 66], we leverage the perplexity (PPL) [35] metric computed on multiple datasets. This metric measures the quality of LLM generated text, lower PPL indicated higher quality. We employ WikiText-2 [46], PTB corpus [44], and LAMBADA dataset [49], which are all commonly used benchmarks in LLM quantization [14, 37, 66]. On the three datasets, we can see that, the increase on PPL are all less than 0.5. This indicates that the LLM functionality loss introduced by our quantization is very small.

[3] [31] is not open-source and only supports proving separate neural network layers. The complex synchronization in its implementation makes it difficult to timekeep the prover and the verifier separately. We sum and report the numbers provided in their paper.

[4] For VOLE based systems we include the communication time into the prover time for fair comparison.

Table 5: Acceleration ratio of constraint fusion and circuit squeeze.

		Prover Time (wo Cons-Opt)	Prover Time (wo Circ-Opt)	Prover Time (all Opt)
Commit advice		1.5s	0.8s	0.8s
GKR	Layer SC	6.0s	20.3s	5.7s
	Combine SC	3.3s	3.1s	3.2s
Lookup		22.4s	12.3s	12.1s
Total		33.2s	36.5s	21.8s

Table 6: Prover time on LLMs of different sizes.

GPT-2	125M	355M	774M	1.5B
Prover Time (s)	21.8	54.1	88.9	165.3
Llama2	7B		13B	
Prover Time (s)	634.2		1154.6	

7.3 Effect of optimization at different levels

We also study the effect of our proposed two optimizations, constraint fusion, circuit compression, respectively.

From Table 5 we can see the comparison between our fully optimized system and our optimized system only disabling constraint and circuit optimization, respectively. The overall acceleration ratio of constraint optimization is 1.5×, its mainly composed of reducing the lookup procedure runtime by 1.8× and reducing the commitment time on auxiliary inputs by 1.9×. The lookup runtime is reduced due to constraint optimization reduces the number of range relations the system has to check. The commitment time is reduced mainly because the number of committed intermediate rounding results is reduced. For circuit optimization, we can see that for the per-layer sumcheck procedure in GKR protocol, by adopting circuit optimization, we can achieve an acceleration ratio of 1.7× compared with not adopting circuit optimization. Before adopting circuit optimization, the layer-wise sumcheck in GKR protocol is actually quite time consuming, occupying about 56% of the prover time. After optimizing this part, the time ratio of GKR layer-wise sumcheck is reduced to 26%.

From Table 3 we can see that, compared with our system without optimization implemented in single thread, further jointly adopting constraint optimization and circuit optimization accelerates the end-to-end performance by 14.7×. Even compared with the strong baseline of adopting 32-thread parallelism for acceleration, our optimizations accelerate the system's end-to-end performance by 2.4×, unlocking great optimization opportunities aside from trivially accelerating everything by multi-thread parallelism.

7.4 Extension to different LLMs

To show the generalizability of our design on different LLMs, we evaluate our system on 6 different models: GPT-2 125M, 355M, 774M, 1.5B and Llama2 7B, 13B. The main architec-

tural difference between GPT-2 and Llama2 is that, Llama2 employs pre-layer normalization instead of post-layer normalization, which means the normalization is applied before each sub-layer. However, this difference could be simply resolved by adjusting the layer orders in our system. The experimental results are shown in Table 6. From the results we can see that, our system scales well with different sizes of LLMs, even under the most challenging Llama2-13B case, our system can prove its inference in less than 20 minutes. This is even faster than Hao et al (Usenix'24) proving the small GPT-2 model (shown in Table 3), showing the strong scalability of our zkGPT.

8 Related Work

Zero-Knowledge Proof for machine learning. There has been significant research on developing ZKPs for ML. Pioneering work [71] introduced methods to prove decision trees. Subsequent efforts focused on verifying the correctness of CNN executions, as seen in [18], [38], [63] and [36]. To address inefficiencies in previous approaches, zkCNN [41] developed a specialized protocol to prove convolution and built a system for verifying CNNs. Further improvements were made in [6], which proved convolution with lower complexity compared with zkCNN [41].

Recently, there has been a surge in research on building ZKPs for LLMs. ZKML [10] generated proofs for GPT-2 [51] inference, requiring over an hour to generate a proof for GPT-2. Concurrently, Hao et al. [31] employed digit decomposition to prove non-linear layers in neural networks. However, their approach does not support end-to-end proof generation for LLMs. Lu et al. [42] proposed optimizations to [31], achieving significantly better performance. Another concurrent work, zkLLM [57], focused on proving the attention layer using digit decomposition. Their system is implemented purely with GPU parallelism. However, it is incompatible with provers lacking advanced GPUs.

Compared to proving inference, proving training processes is far more challenging. Several studies [1, 25, 56, 62] have explored the ZKP of training various ML models. Current approaches still involve significant overhead.

Proof aggregation. Our constraint merging technique is very different from proof-aggregation techniques like Bulletproofs [8]. They focus on reducing proof size, but cannot reduce the prover's computation. Our constraint merging technique decreases the prover time.

9 Discussion

9.1 Real world passing probability

According to Appendix F, in real world, the passing probability of an adversary is dominated by $O(\frac{m_1+N_1^{\frac{1}{c}}+m_2+N_2}{|\mathbb{F}|})$.

Table 7: Q resembles the value of quantization level.

Q	8	16	24	32
Prover Time (s)	20.9	21.8	102.3	26362.5
Perplexity	35.6	29.5	29.3	29.3

With $|\mathbb{F}| \approx 2^{256}$ in our system, the soundness error is negligible ($\approx 2^{-234}$). The real-world cheating probability is strictly bounded by the soundness error as long as the attacker has bounded (polynomial) computational power.

9.2 Tradeoffs of quantization precision

We first analyze the impact of quantization to our system's privacy. Quantization levels does not impact the privacy of our system, because our underlying GKR proof system satisfies perfect zero-knowledge. In perfect zero-knowledge, a malicious verifier cannot distinguish between the real world and the ideal world with probability better than random guessing. This property holds because the messages observed by the verifier in the ZKP protocol are independently and uniformly distributed (detailed derivations are provided in [67]). The "real world" refers to the world where our actual circuit is runned. The "ideal world" refers to a world where all circuit values are random. Thus, adopting different levels of quantization does not impact the privacy of our system.

Then we evaluate and analyze the impact of different quantization precisions to prover time and numerical error. We show the results in Table 7. For prover time, increasing Q from 8 to 16 does not have significant impact, but increasing from 16 to 32 increases it a lot. The reason is that, our exponentiation table has size $N_2 = 2^Q$. When Q increases from 16 to 32, N_2 significantly increases and the increased prover time mainly comes from the increased lookup overhead.

For numerical error, we can see from Table 7 that, using $Q = 8$ results in large perplexity loss. In contrast, for $Q \geq 16$, the numerical precision are all close to the FP32 inference perplexity in Table 4. From the above results and analysis, we can conclude that our choice of quantization level $Q = 16$ achieves a good balance between prover time and numerical error.

9.3 Impact of different merging levels

We also explore the impact of different constraint merging levels. (A) No merging, which is the slowest case. (B) Our proposed method. (C) Fastest merging, which is conducting all beneficial mergings to the constraints. (D) Full merging, which is conducting all possible mergings to the constraints. The constraints for level (A) are introduced in Section 4.1 and Appendix G. The constraints for level (B) are introduced in Section 5.3 and Appendix H. Our proposed method (B) theoretically achieves nearly ideal performance.

Table 8: Merging level (A) resembles no merging, which is the slowest case, (B) resembles our proposed method, (C) resembles fastest merging, (D) resembles fully (most aggressive) merging. Fully merged means all possible merges are conducted, partially merged means some merges are not conducted.

Merging level	Level (A)	Level (B)	Level (C)	Level (D)
Attention	Not merged	Partially merged	Partially merged	Fully merged
Layer Norm	Not merged	Partially merged	Fully merged	Fully merged
GeLU	Not merged	Fully merged	Fully merged	Fully merged
Prover Time (s)	33.4	21.81	21.76	$10^{154}(Estimated)$
Lookup query	4099680	2141184	2140416	2069760

How merging levels impact privacy. Due to the perfect zero-knowledge as elaborated in Section 9.2, the privacy property of our system is the same across different merging levels.

Next, we analyze the possible mergings that could be conducted on top of level (B).

Attention. In Equation 3, merging the computations of $q_{e_{ij}}$ and $q_{P_{ij}}$ is not feasible.This is because ensuring the correctness of $q_{P_{ij}}$ requires lookup queries that must be fully computable within the GKR circuit. Thus, all variables used that cannot be directly arithmetically computed ($q_{e_{ij}}$) has to be checked with additional lookup constraints that cannot be merged. However, merging $q_{E_{max,i}}$ and $q_{e_{i,j}}$ together is possible. The merged constraint is as follows:

$$q_{e_{ij}} = \exp(\max_j q_{E_{ij}} - q_{E_{ij}}), \text{ for } i = 1, \cdots s, j = 1, \cdots, i \quad (5)$$

Level (D) adopts this merging. However, it introduces an infeasible computational cost. Specifically, checking the computation of $q_{e_{i,j}}$ using a lookup table can be modeled as a non-linear function with q_{E_i} as its input. Based on our quantization, q_{E_i} is a vector with length 32 each element is 16 bit. Consequently, the total number of possible inputs to this lookup table is $2^{16\times 32} = 2^{512}$, resulting in a prohibitively computational cost. Both level (B) and level (C) avoid this merging since such aggressive merging significantly harms performance.

Gelu. GeLU cannot be further merged, this is due to checking Equation 14 requires checking the range relation $2q_g - 1 \leq q_x + |q_x| - \mathbb{1}_{|q_x|\geq q_{\text{th}}} \cdot |q_x| \left(a \cdot S_x^2 q_x^2 + b \cdot S_x |q_x| + c\right) < 2q_g + 1$. In the relation, $\mathbb{1}_{|q_x|\geq q_{\text{th}}}$ cannot be arithmetically computed. Thus, s has to be provided in the input layer as witness. And an additional lookup query is required to ensure the correctness of s. In conclusion, GeLU is already fully merged in level (B) and cannot be further merged.

Normalization. The layer normalization constraint for level (B) is depicted in Equation 3. Level (C) and (D) further merges q_{σ_i} with $q_{y_{ij}}$. However, it only reduces the number of rounding constraints from $sd + s$ to sd. Thus, this merge is marginally beneficial.

We now evaluate the impact of different merging levels on prover time. Compared with level (A), level (B) significantly reduces prover time by decreasing the total number of lookup queries by approximately $1.91\times$, as shown in Table 8. This results in an overall prover time reduction of $1.53\times$. Additionally, Table 8 indicates that the prover time for level (C) is only marginally faster than that of level (B). This is because level (C) offers minimal constraint-saving benefits for the layer normalization layer compared with level (B). In contrast, level (D) leads to an impractical prover time due to the need to construct an unrealistically large lookup table for the Attention layer. This demonstrates that the highest merging level does not necessarily result in the most efficient prover time. Consequently, we adopted level (B) in our implementation, as the expected performance gain from level (C) was too marginal to justify the additional engineering effort.

9.4 Generalizability of our optimizations for matrix multiplication proof

Our efficient construction for proving matrix multiplication in Section 4.2 consists of three techniques. The first technique grouping reduces costs in general scenarios. Because grouping imposes no assumptions on the density of matrices. The second technique skipping computations based on sparsity is effective only if the padded (power of 2) matrix is sparse. However this is often the case in real-world matrix multiplications, since the width or height of the matrix often does not equal the power of 2. The third technique saving field multiplication by precomputing requires the properties of quantization to the input matrices. Thus it is mainly generalizable to most cases of zero-knowledge machine learning.

9.5 Limitations of ZK-GPT

The primary limitation of ZK-GPT's design lies in its reliance on quantization, rendering it unsuitable for directly proving the floating-point inference of neural networks. However, as previously noted, most existing ZKP systems for machine learning [10,18,38,41,42,57] share this limitation due to their dependence on quantization. Extending ZK-GPT to floating-point leads to significantly higher prover overhead. Future work that could be done includes (1) Exploring the proof generation under floating-point computation. (2) Exploring proof generation for LLM training. (3) Exploring domain-specific distributed proof generation system tailored for LLM inference/training.

10 Conclusion

In this paper, we introduce a highly efficient ZKP framework for large language models (LLMs). Our efficient methods for proving both linear and non-linear layers in LLMs offer huge performance improvements by orders of magnitude.

Additionally, we propose optimization techniques at both the constraint and circuit levels. Finally, we implement our scheme on GPT-2, and the evaluation results demonstrate a significant speedup compared to existing methods.

Discussion: Research Ethics

Our research focuses on enhancing the transparency of existing LLM cloud services. Our method enables LLM service providers to ensure that customers' outputs are generated correctly without compromising the privacy of the underlying model. We adhere strictly to ethical guidelines, avoiding deceptive practices, unauthorized disclosures, live system experiments, or any actions that could jeopardize the well-being of our team. Consequently, our work seeks to improve the transparency in LLM services without introducing new ethical concerns beyond those already associated with LLMs.

Discussion: Open Science Policy

We provide our source code in the anonymous repository https://github.com/security-Anonymous/zkTransformer.

References

[1] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. *Cryptology ePrint Archive*, 2024.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Eleni Adamopoulou and Lefteris Moussiades. An overview of chatbot technology. In *IFIP international conference on artificial intelligence applications and innovations*, pages 373–383. Springer, 2020.

[4] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2024.

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

[6] David Balbás, Dario Fiore, Maria Isabel González Vasco, Damien Robissout, and Claudio Soriente. Modular sum-check proofs with applications to machine learning and image processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1437–1451, 2023.

[7] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.9.0, February 2023.

[8] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[9] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: improvements, extensions and applications to zero-knowledge decision trees. In *IACR International Conference on Public-Key Cryptography*, pages 337–369. Springer, 2024.

[10] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 560–574, 2024.

[11] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.

[12] Xiao Wang Chenkai Weng. Emp-zk toolkit, 2024. URL: https://github.com/emp-toolkit/emp-zk?tab=readme-ov-file#throughput-of-circuit-based-zk-protocol.

[13] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112, 2012.

[14] 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.

[15] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115, 2023.

[16] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. *Cryptology ePrint Archive*, 2022.

[17] Jens Ernstberger, Chengru Zhang, Luca Ciprian, Philipp Jovanovic, and Sebastian Steinhorst. Zero-knowledge location privacy via accurate floating point snarks. *arXiv preprint arXiv:2404.14983*, 2024.

[18] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *Cryptology ePrint Archive*, 2021.

[19] Boyuan Feng, Zheng Wang, Yuke Wang, Shu Yang, and Yufei Ding. Zeno: A type-based optimization framework for zero knowledge neural network inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 450–464, 2024.

[20] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[21] OpenAI Forum. Gpt4-turbo more "stupid/lazy" - it's not a gpt4, 2024. URL: https://community.openai.com/t/gpt4-turbo-more-stupid-lazy-its-not-a-gpt4/608008.

[22] OpenAI Forum. Openai did made gpt3.5 more stupid?, 2024. URL: https://community.openai.com/t/openai-did-made-gpt3-5-more-stupid/262979.

[23] Ariel Gabizon and Zachary J Williamson. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive*, 2020.

[24] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[25] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1880–1894, 2023.

[26] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. Succinct zero knowledge for floating point computations. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1203–1216, 2022.

[27] Sergey B Gashkov and Igor S Sergeev. Complexity of computation in finite fields. *Journal of Mathematical Sciences*, 191:661–685, 2013.

[28] Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo–a turing-complete stark-friendly cpu architecture. *Cryptology ePrint Archive*, 2021.

[29] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, 1985.

[30] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

[31] Meng Hao, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang, Haomiao Yang, and Tianwei Zhang. Scalable zero-knowledge proofs for non-linear functions in machine learning. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3819–3836, 2024.

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

[33] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[34] Bernard J Jansen and Udo Pooch. A review of web searching studies and a framework for future research. *journal of the American Society for Information Science and Technology*, 52(3):235–246, 2001.

[35] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

[36] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Scaling up trustless dnn inference with zero-knowledge proofs. *arXiv preprint arXiv:2210.08674*, 2022.

[37] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

[38] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vcnn: Verifiable convolutional neural network based on zk-snarks. *Cryptology ePrint Archive*, 2020.

[39] Alan Li, Qingkai Liang, and Mo Dong. Sparsity-aware protocol for zk-friendly ml models: Shedding lights on practical zkml. *Cryptology ePrint Archive*, 2024.

[40] T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 39–39, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

[41] Tianyi Liu, Xiang Xie, and Yupeng Zhang. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2968–2985, 2021.

[42] Tao Lu, Haoyu Wang, Wenjie Qu, Zonghui Wang, Jinye He, Tianyang Tao, Wenzhi Chen, and Jiaheng Zhang. An efficient and extensible zero-knowledge proof framework for neural networks. *Cryptology ePrint Archive*, 2024.

[43] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

[44] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[45] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[46] Stephen Merity. The wikitext long term dependency language modeling dataset. *Salesforce Metamind*, 9, 2016.

[47] Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 883–898, 2021.

[48] NVIDIA. Nvidia a100 datasheet, 2024. URL: https://www.nvidia.com/en-us/data-center/a100/.

[49] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[50] Nicholas Pippenger. On the evaluation of powers and related problems. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 258–263. IEEE Computer Society, 1976.

[51] 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.

[52] Alex Saltanov. Openai keeps dumbing down chatgpt, 2024. URL: https://pub.aimind.so/openai-keeps-dumbing-down-chatgpt-6a6e4a173237.

[53] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

[54] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

[55] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 180–209. Springer, 2024.

[56] Ali Shahin Shamsabadi, Sierra Calanda Wyllie, Nicholas Franzese, Natalie Dullerud, Sébastien Gambs, Nicolas Papernot, Xiao Wang, and Adrian Weller. Confidential-profitt: confidential proof of fair training of trees. In *The Eleventh International Conference on Learning Representations*, 2022.

[57] Haochen Sun, Jason Li, and Hongyang Zhang. zkllm: Zero knowledge proofs for large language models. 2024.

[58] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Annual Cryptology Conference*, pages 71–89. Springer, 2013.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[60] Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 223–237. IEEE, 2013.

[61] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018.

[62] Suppakit Waiwitlikhit, Ion Stoica, Yi Sun, Tatsunori Hashimoto, and Daniel Kang. Trustless audits without revealing data or models. In *Forty-first International Conference on Machine Learning*, 2024.

[63] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518, 2021.

[64] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 675–692, Baltimore, MD, August 2018. USENIX Association.

[65] Min-You Wu and Daniel D Gajski. Hypertool: A programming aid for message-passing systems. *IEEE transactions on parallel and distributed systems*, 1(3):330–343, 1990.

[66] 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*, pages 38087–38099. PMLR, 2023.

[67] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 733–764. Springer, 2019.

[68] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3003–3017, 2022.

[69] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3121–3134, 2022.

[70] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Rafols. Baloo: Nearly optimal lookup arguments. *Cryptology ePrint Archive*, 2022.

[71] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2039–2053, 2020.

[72] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 159–177, 2021.

[73] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 859–876. IEEE, 2020.

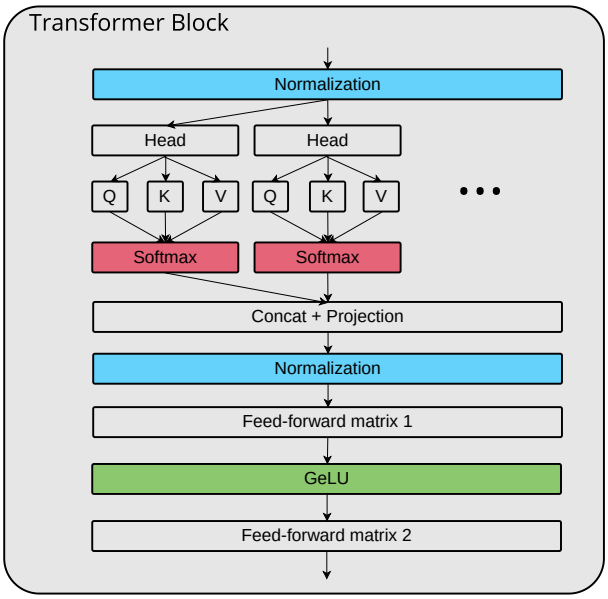


Figure 3: Structure of each transformer block in GPT-2 [51], the structure of other LLMs are similar. An LLM is composed of L concatenated blocks.

A Transformer Block Structure

Normalization is the first NN layer of the transformer block, which is computed as follows:

$$y_{ij} = \gamma_j \cdot \frac{x_{ij} - \mu_i}{\sigma_i} + \beta_j \tag{6}$$

where y is the output with same shape of x, $\mu_i = \frac{\sum_{j=1}^{d} x_{ij}}{d}$ is the mean of row i of x, σ_i is the standard deviation of row i of x, γ, β are vectors parameterizing the affine transform. Then matrix y is feeded into the multi-headed attention.

The output of multi-headed attention O is computed as follows:

$$
\begin{aligned}
Q_i &= y \cdot W_Q^i \\
K_i &= y \cdot W_K^i \\
V_i &= y \cdot W_V^i \\
P &= \underset{i=1,\cdots,h}{Concat}(Softmax(\frac{Q_i \cdot K_i}{\sqrt{d}})V_i) \\
O &= P \cdot W_O
\end{aligned} \tag{7}
$$

where W_Q^i, W_K^i, W_V^i are weights for each attention head i, W_O is the linear weight used to project the output at the end of the multi-head attention, *Concat* defines the function that concatenates different matrices with the same shape.

O is then feeded into another normalization layer, obtaining matrix o. o is sent into the part of transformer block named feed-forward network (FFN). o is first mapped to z which has shape $s \times D$ where D is the dimension of FFN, typically $D = k \cdot d, k > 1$.

Then z is feeded into the Gaussian Error Linear Unit (GeLU) activation function. It is computed as follows:

$$
g_{ij} = z_{ij} \cdot \Phi(z_{ij}) \tag{8}
$$

where g is the output matrix of GeLU, with the same shape of z. g is then sent into the second matrix of FFN, projecting it to f with shape of $s \times d$. Then f is feeded into the subsequent transformer block, so on and so force, until all L transformer blocks are computed.

Then the matrix f' obtained at the final transformer block, is linearly projected to S with shape of $s \times |\mathcal{V}|$, where $\mathcal{V}$ is the set of vocabulary for the LLM. Then we randomly sample the generated token from the vocabulary set leveraging the probability computed by the softmax of the last row of S. Following previous work [10, 42, 57], we omit proving the final linear projection and the token generation procedure.

B Definitions

We introduce definitions of zero knowledge arguments and zero knowledge polynomial commitments before presenting the formal protocols.

Zero knowledge arguments. An argument system for an NP relationship $\mathcal{R}$ is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness w such that $(x;w) \in R$ for some input x. We focus on arguments of knowledge which have the stronger property that if the prover convinces the verifier of the statement validity, then the prover must know w. We use $\mathcal{G}$ to represent the generation phase of the public parameters pp. Formally, consider the definition below, where we assume R is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition 2. *Let $\mathcal{R}$ be an NP relation. A tuple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a zero knowledge argument of knowledge for $\mathcal{R}$ if the following holds.*

- ***Correctness***. *For every* pp *output by* $\mathcal{G}(1^\lambda)$ *and* $(x,w) \in R$,
$$\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}) \rangle(x) = 1$$
- ***Knowledge Soundness***. *For any PPT prover $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that given the access to the entire executing process and the randomness of $\mathcal{P}^*$, $\mathcal{E}$ can extract a witness w such that* $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $\pi^* \leftarrow \mathcal{P}^*(x, \mathsf{pp})$ *and* $w \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, x, \pi^*)$, *the following probability is* $\mathsf{negl}(\lambda)$*:*
$$\Pr[(x;w) \notin \mathcal{R} \wedge \mathcal{V}(x, \pi^*, \mathsf{pp}) = 1]$$
- ***Zero knowledge***. *There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, auxiliary input $z \in \{0,1\}^*$, $(x;w) \in \mathcal{R}$,* pp *output by* $\mathcal{G}(1^\lambda)$, *it holds that*
$$\mathsf{View}(\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}^*(z, \mathsf{pp}) \rangle(x)) \approx \mathcal{S}^{\mathcal{V}^*}(x, z)$$

We say that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a ***succinct*** *argument system if the total communication between $\mathcal{P}$ and $\mathcal{V}$ (proof size) are* $\mathsf{poly}(\lambda, |x|, \log |w|)$.

In the definition of zero knowledge, $\mathcal{S}^{\mathcal{V}^*}$ denotes that the simulator $\mathcal{S}$ is given the randomness of $\mathcal{V}^*$ sampled from polynomial-size space. This definition is commonly used in existing transparent zero knowledge proof schemes [8, 61, 73].

Zero knowledge polynomial commitment. Let $\mathbb{F}$ be a finite field, $\mathcal{F}$ be a family of ℓ-variate polynomial over $\mathbb{F}$. A zero knowledge verifiable polynomial commitment (zkPC) for $f \in \mathcal{F}$ and $t \in \mathbb{F}^\ell$ consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$,
- $\mathsf{com} \leftarrow \mathsf{zkPC.Commit}(f, r_f, \mathsf{pp})$,
- $((y, \pi); \{0,1\}) \leftarrow \langle \mathsf{zkPC.Open}(f, r_f), \mathsf{zkPC.Verify}(\mathsf{com}) \rangle(t, \mathsf{pp})$

C Protocols

C.1 Sumcheck protocol

Sumcheck protocol is one of the most impactful interactive proofs in previous literature. The sumcheck problem is to sum a multivariate polynomial $f : \mathbb{F}^l \to \mathbb{F}$ on all binary inputs: $\sum_{b_1, b_2, \ldots, b_\ell \in \{0,1\}} f(b_1, b_2, \ldots, b_\ell)$. Directly computing the sum requires exponential time in ℓ, as there are 2^ℓ combinations of $b_1, \ldots, b_\ell$. [43] proposed a sumcheck protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, who can convince $\mathcal{V}$ that H is the correct sum. We provide a description of the sumcheck protocol in Protocol 1.

Lemma C.1. *Protocol 1 is an interactive proof for* $H = \sum_{b_1,b_2,\ldots,b_\ell \in \{0,1\}} f(b_1,b_2,\ldots,b_\ell)$ *that is complete and with soundness error bounded by* $\frac{\ell}{|\mathbb{F}|}$.

The proof size of the sumcheck protocol is $O(\ell)$ if the variable degree of f is constant, which is the case in our protocols. This is because in each round, $\mathcal{P}$ sends a univariate polynomial of one variable in f, which is of constant size. The verifier time of the protocol is $O(\ell)$. The prover time depends on the degree and the sparsity of f.

C.2 GKR protocol

Let C be a layered arithmetic circuit with depth d over a finite field $\mathbb{F}$. Each gate in the i-th layer takes inputs from two gates in the $(i-1)$-th layer; layer 0 is the input layer and layer d is the output layer. Following the convention in prior work [41, 67] we denote the number of gates in the i-th layer as S_i and let $s_i = \lceil \log S_i \rceil$. (For simplicity, we assume S_i is a power of 2, and we can pad the layer with dummy gates otherwise.) We then define a function $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ that takes a binary string $b \in \{0,1\}^{s_i}$ and returns the output of gate b in layer i, where b is called the gate label. With this definition, V_0 corresponds to the output of the circuit, and V_d corresponds to the input layer. Finally, we define two additional functions $add_i, mult_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$, referred to as wiring predicates in the literature. $add_i(mult_i)$ takes one gate label $z \in \{0,1\}^{s_{i-1}}$ in layer $i-1$ and two gate labels $x, y \in \{0,1\}^{s_i}$ in layer i, and outputs 1 if and only if gate z is an addition (multiplication) gate that takes the output of gate x, y as input. Taking the multi-linear extensions of V_i, add_i and $mult_i$, define $f_i(g,x,y) = \tilde{add}_i(g,x,y)(\tilde{V}_{i-1}(x) + \tilde{V}_{i-1}(y)) + \tilde{mult}_i(g,x,y)\tilde{V}_{i-1}(x) \cdot \tilde{V}_{i-1}(y)$. for any $g \in \mathbb{F}^{s_i}$, $\tilde{V}_i(g) = \sum_{x,y \in \{0,1\}^{s_{i-1}}} f(g,x,y)$.

The GKR protocol proceeds as follows. The prover $\mathcal{P}$ first sends the claimed output of the circuit to $\mathcal{V}$. From the claimed output, $\mathcal{V}$ defines polynomial $\tilde{V}_d$ and computes $\tilde{V}_d(g)$ for a random $g \in \mathbb{F}^{s_d}$. $\mathcal{V}$ and $\mathcal{P}$ then invoke a sumcheck protocol on $\tilde{V}_d(g)$. At the end of the sumcheck, $\mathcal{V}$ needs an oracle access to $f_d(g,u,v)$, where u, v are randomly selected in $\mathbb{F}^{s_{i-1}}$. To compute $f_d(g,u,v)$, $\mathcal{V}$ computes wiring pattern of the circuit, and asks $\mathcal{P}$ to send $\tilde{V}_{d-1}(u)$ and $\tilde{V}_{d-1}(v)$ and computes $f_d(g,u,v)$ to complete the sumcheck protocol. In this way, $\mathcal{V}$ and $\mathcal{P}$ reduce a claim about the output to two claims about values in layer $d-1$. $\mathcal{V}$ and $\mathcal{P}$ then combines the two claims into one through a random linear combination, and run a sumcheck protocol on $\tilde{V}_{i-1}(g)$, and then recursively all the way to the input layer. The formal GKR protocol and its properties are presented in Protocol 2.

Protocol 1 (Sumcheck). *The protocol proceeds in ℓ rounds.*

• *In the first round, $\mathcal{P}$ sends a univariate polynomial*

$$f_1(x_1) \stackrel{def}{=} \sum_{b_2,\ldots,b_\ell \in \{0,1\}} f(x_1,b_2,\ldots,b_\ell),$$

$\mathcal{V}$ checks $H = f_1(0) + f_1(1)$. Then $\mathcal{V}$ sends a random challenge $r_1 \in \mathbb{F}$ to $\mathcal{P}$.

• *In the i-th round, where $2 \le i \le \ell - 1$, $\mathcal{P}$ sends a univariate polynomial*

$$f_i(x_i) \stackrel{def}{=} \sum_{b_{i+1},\ldots,b_\ell \in \{0,1\}} f(r_1,\ldots,r_{i-1},x_i,b_{i+1},\ldots,b_\ell),$$

$\mathcal{V}$ checks $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$, and sends a random challenge $r_i \in \mathbb{F}$ to $\mathcal{P}$.

• *In the ℓ-th round, $\mathcal{P}$ sends a univariate polynomial*

$$f_\ell(x_\ell) \stackrel{def}{=} f(r_1,r_2,\ldots,r_{l-1},x_\ell),$$

$\mathcal{V}$ checks $f_{\ell-1}(r_{\ell-1}) = f_\ell(0) + f_\ell(1)$. The verifier generates a random challenge $r_\ell \in \mathbb{F}$. Given oracle access to an evaluation $f(r_1,r_2,\ldots,r_\ell)$ of f, $\mathcal{V}$ will accept if and only if $f_\ell(r_\ell) = f(r_1,r_2,\ldots,r_\ell)$. The instantiation of the oracle access depends on the application of the sumcheck protocol.

D ML-friendly circuit and extended GKR protocol

For the ML-friendly circuit, we denote the addition gate between the i-th and the $(i-1)$-th, the i-th and the input layer as $\tilde{add}_i(z,x)$, $\tilde{add}_{in}(z,x)$. $\tilde{add}_i(z,x)$ returns 1 if $V_{i-1}(x)$ is added to $V_i(z)$, otherwise return 0. $\tilde{add}_{in}(z,x)$ returns 1 if $V_{in}(x)$ is added to $V_i(z)$, otherwise return 0. We define the multiplication gate taking inputs both from the previous layer, both from the input layer and one from the previous layer one from input as $\tilde{mult}_i(z,x,y)$, $\tilde{mult}_{in}(z,x,y)$, $\tilde{mult}_{i,in}(z,x,y)$. Therefore, we have:

$$\begin{aligned}
\tilde{V}_i(z) = &\sum_{x \in \{0,1\}^{s_{i-1}}} \tilde{add}_i(z,x) \cdot \tilde{V}_{i-1}(x) \\
&+ \sum_{x \in \{0,1\}^{s_{in}}} \tilde{add}_{in}(z,x) \cdot \tilde{V}_{i,in}(x) \\
&+ \sum_{x,y \in \{0,1\}^{s_{i-1}}} \tilde{mult}_i(z,x,y) \cdot \tilde{V}_{i-1}(x)\tilde{V}_{i-1}(y) \\
&+ \sum_{x,y \in \{0,1\}^{s_{i,in}}} \tilde{mult}_{in}(z,x,y) \cdot \tilde{V}_{i,in}(x)\tilde{V}_{i,in}(y) \\
&+ \sum_{\substack{x \in \{0,1\}^{s_{i,in}}, \\ y \in \{0,1\}^{s_{i-1}}}} \tilde{mult}_{i,in}(z,x,y) \cdot \tilde{V}_{i-1}(x)\tilde{V}_{i,in}(y)
\end{aligned} \tag{9}$$

Protocol 2 (GKR). *Let $\mathbb{F}$ be a finite field. Let C: $\mathbb{F}^n \to \mathbb{F}^k$ be a d-depth layered arithmetic circuit. $\mathcal{P}$ wants to convince that **out** $= C($**in**$)$ where **in** is the input from $\mathcal{V}$, and **out** is the output. Without loss of generality, assume n and k are both powers of 2 and we can pad them if not.*

1. *Define the multi-linear extension of array **out** as $\tilde{V}_d$. $\mathcal{V}$ chooses a random $g \in \mathbb{F}^{s_d}$ and sends it to $\mathcal{P}$. Both parties compute $\tilde{V}_d(g)$.*
2. *$\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol on*

$$\tilde{V}_d(g^{(d)}) = \sum_{x,y\in\{0,1\}^{s_{d-1}}} (\tilde{add}_d(g^{(d)},x,y)(\tilde{V}_{d-1}(x)+\tilde{V}_{d-1}(y)) + \tilde{mult}_d(g^{(d)},x,y)\tilde{V}_{d-1}(x)\tilde{V}_{d-1}(y))$$

   At the end of the protocol, $\mathcal{V}$ receives $\tilde{V}_{d-1}(u^{(d-1)})$ and $\tilde{V}_{d-1}(v^{(d-1)})$. $\mathcal{V}$ computes $\tilde{mult}_d(g^{(d)},u^{(d-1)},v^{(d-1)})$, $\tilde{add}_d(g^{(d)},u^{(d-1)},v^{(d-1)})$ and checks that $\tilde{add}_d(g^{(d)},u^{(d-1)},v^{(d-1)})$ $(\tilde{V}_{d-1}(u^{(d-1)}) + \tilde{V}_{d-1}(v^{(d-1)})) + \tilde{mult}_d(g^{(d)},u^{(d-1)},v^{(d-1)})\,\tilde{V}_{d-1}(u^{(d-1)})\tilde{V}_{d-1}(v^{(d-1)})$ equals to the last message of the sumcheck.
3. *For $i = d-1,...,1$:*
   - *$\mathcal{V}$ randomly selects $\alpha_{i,1},\alpha_{i,2} \in \mathbb{F}$ and sends them to $\mathcal{P}$.*
   - *$\mathcal{P}$ and $\mathcal{V}$ run the sumcheck on the equation*

$$\alpha_{i,1}\tilde{V}_i(u^{(i)}) + \alpha_{i,2}\tilde{V}_i(v^{(i)}) =$$
$$\sum_{x,y\in\{0,1\}^{s_{i-1}}} ((\alpha_{i,1}\tilde{add}_i(u^{(i)},x,y) + \alpha_{i,2}\tilde{add}_i(v^{(i)},x,y))(\tilde{V}_{i-1}(x)+\tilde{V}_{i-1}(y))$$
$$+(\alpha_{i,1}\tilde{mult}_i(u^{(i)},x,y) + \alpha_{i,2}\tilde{mult}_i(v^{(i)},x,y))\tilde{V}_{i-1}(x)\tilde{V}_{i-1}(y))$$

   - *At the end of the sumcheck protocol, $\mathcal{P}$ sends $\mathcal{V}$ $\tilde{V}_{i-1}(u^{(i-1)})$ and $\tilde{V}_{i-1}(v^{(i-1)})$.*
   - *$\mathcal{V}$ computes the following and checks if it equals to the last message of the sumcheck. For simplicity, let $\tilde{mult}_i(x) = \tilde{mult}_i(x,u^{(i-1)},v^{(i-1)})$ and $\tilde{add}_i(x) = \tilde{add}_i(x,u^{(i-1)},v^{(i-1)})$.*

$$(\alpha_{i,1}\tilde{mult}_i(u^{(i)}) + \alpha_{i,2}\tilde{mult}_i(v^{(i)}))\tilde{V}_{i-1}(u^{(i-1)})\tilde{V}_{i-1}(v^{(i-1)}) +$$
$$(\alpha_{i,1}\tilde{add}_i(u^{(i)}) + \alpha_{i,2}\tilde{add}_i(v^{(i)}))(\tilde{V}_{i-1}(u^{(i-1)}) + \tilde{V}_{i-1}(v^{(i-1)}))$$

   If all checks in the sumcheck pass, V uses $\tilde{V}_{i-1}(u^{(i-1)})$ and $\tilde{V}_{i-1}(v^{(i-1)})$ to proceed to the $(i-1)$-th layer. Otherwise, $\mathcal{V}$ outputs `0` *and aborts.*
4. *At the input layer d, $\mathcal{V}$ has two claims $\tilde{V}_0(u^{(0)})$ and $\tilde{V}_0(v^{(0)})$. $\mathcal{V}$ evaluates $\tilde{V}_0$ at $u^{(0)}$ and $v^{(0)}$ using the input and checks that they are the same as the two claims. If yes, output* `1`*; otherwise, output* `0`*.*

By executing the sumcheck protocol on $\alpha\tilde{V}_i(z_1) + \beta\tilde{V}_i(z_2)$ at the i−th layer, the verifier and the prover can directly reduce two evaluations of $\tilde{V}_i(\cdot)$ to two evaluations of $\tilde{V}_{i-1}(\cdot)$ and two evaluations of $\tilde{V}_{i,\text{ in}}(\cdot)$ [41]. The prover time is $O(S_i + S_{i-1} + S_{i,in})$.

When they arrive at the input layer, suppose the evaluations received from layer i are $\tilde{V}_{i,in}(z_{i,0})$ and $\tilde{V}_{i,in}(z_{i,1})$. The verifier generates randomness $r_{i,0}, r_{i,1} \in \mathbb{F}$ for claims on layer i and combines all the claims through a random linear combination:

$$\sum_i \left(r_{i,0}\tilde{V}_{i,\text{in}}(z_{i,0}) + r_{i,1}\tilde{V}_{i,\text{in}}(z_{i,1})\right) =$$
$$\sum_{z\in\{0,1\}^{s_{\text{in}}}} \tilde{V}_{\text{in}}(z)\left(\sum_i \left(r_{i,0}C_i(z_{i,0},z) + r_{i,1}C_i(z_{i,1},z)\right)\right) \tag{10}$$

where $C_i(z_i,z)$ returns 1 if the z_i-th value in $V_{i,in}$ is the z-th value in V_{in}, otherwise return 0.

By running the sumcheck protocol on Equation 10, the verifier reduces multiple claims on $\tilde{V}_{i,\text{in}}(\cdot)$ to a single evaluation of $\tilde{V}_{\text{in}}(\cdot)$. The prover time is linear in the input size.

Protocol 3 (Lasso). *T is an SOS lookup table of size N, meaning there are $\alpha = kc$ tables $T_1, \ldots, T_\alpha$, each of size $N^{1/c}$, such that for any $r \in \{0,1\}^{\log N}, T[r] = g(T_1[r_1], \ldots, T_k[r_1], T_{k+1}[r_2], \ldots, T_{2k}[r_2], \ldots, T_{\alpha-k+1}[r_c], \ldots, T_\alpha[r_c])$. During the commit phase, $\mathcal{P}$ commits to c multi-linear polynomials $\mathrm{dim}_1, \ldots, \mathrm{dim}_c$, each over $\log m$ variables. dim_i is purported to provide the indices of $T_{(i-1)k+1}, \ldots, T_{ik}$ the natural algorithm computing $\sum_{i \in \{0,1\}^{\log m}} eq(i,r) \cdot T[\mathrm{nz}[i]]$.*

$//\mathcal{V}$ requests $\langle u, t \rangle$, where the i th entry of t is $T[i]$ and the y th entry of u is $\widetilde{M}(r,y)$.

1. $\mathcal{P} \to \mathcal{V}$: 2α different $(\log m)$-variate multi-linear polynomials $E_1, \ldots, E_\alpha$, $\mathsf{read\_ts}_1, \ldots \mathsf{read\_ts}_\alpha$ and α different $(\log(N)/c)$-variate multi-linear polynomials $\mathsf{final\_cts}_1, \ldots, \mathsf{final\_cts}_\alpha$.

$//E_i$ is purported to specify the values of each of the m reads into T_i. sub-tables T_i.

$//\mathsf{read\_ts}_1, \ldots \mathsf{read\_ts}_\alpha$ and $\mathsf{final\_cts}_1, \ldots \mathsf{final\_cts}_\alpha$, are "counter polynomials" for each of the α sub-tables T_i.

2. $\mathcal{V}$ and $\mathcal{P}$ apply the sum-check protocol to the polynomial $h(k) := \tilde{\mathrm{eq}}(r,k) \cdot g(E_1(k), \ldots, E_\alpha(k))$, which reduces the check that $v = \sum_{k \in \{0,1\}^{\log m}} g(E_1(k), \ldots, E_\alpha(k))$ to checking that the following equations hold, where $r_z \in \mathbb{F}^{\log m}$ chosen at random by the verifier over the course of the sum-check protocol:

- *$E_i(r_z) \stackrel{?}{=} v_{E_i}$ for $i = 1, \ldots, \alpha$. Here, $v_{E_1}, \ldots, v_{E_\alpha}$ are values provided by the prover at the end of the sum-check protocol.*

3. $\mathcal{V}$: check if the above equalities hold with one oracle query to each E_i.

4. // The following checks if E_i is well-formed, i.e., that $E_i(j)$ equals $T_i[\mathrm{dim}_i(j)]$ for all $j \in \{0,1\}^{\log m}$.

5. $\mathcal{V} \to \mathcal{P} : \tau, \gamma \in_H \mathbb{F}$.

// In practice, one would apply a single sum-check protocol to a random linear combination of the below polynomials. For brevity, we describe the protocol as invoking c independent instances of sum-check.

6. $\mathcal{V} \leftrightarrow \mathcal{P}$: For $i = 1, \ldots, \alpha$, run a sum-check-based protocol for "grand products" (See [54] Section 5) to reduce the check that $\mathcal{H}_{\tau,\gamma}(\mathrm{WS}) = \mathcal{H}_{\tau,\gamma}(\mathrm{RS}) \cdot \mathcal{H}_{\tau,\gamma}(S)$, where RS, WS, S are as defined in [55] Claim 3 and $\mathcal{H}$ is defined in [55] Claim 4 to checking if the following hold, where $r_i'' \in \mathbb{F}^\ell, r_i''' \in \mathbb{F}^{\log m}$ are chosen at random by the verifier over the course of the sum-check protocol:

- *$E_i(r_i''') \stackrel{?}{=} v_{E_i}$*
- *$\mathrm{dim}_i(r_i''') \stackrel{?}{=} v_i; \mathsf{read\_ts}_i(r_i''') \stackrel{?}{=} v_{\mathsf{read\_ts}_i}; add\ \mathsf{final\_cts}_i(r_i'') \stackrel{?}{=} v_{\mathsf{final\_cts}_i}$*

7. $\mathcal{V}$: Check the equations hold with an oracle query to each of E_i, dim_i, $\mathsf{read\_ts}_i$, $\mathsf{final\_cts}_i$.

E Integrate Lasso into GKR protocol

Assume our system invokes t lookup tables. For table j, we have to check the consistency of lookup vector A_j (A_j is part of the circuit's witness and committed together with the input). We can leverage random combination to check the claim $\tilde{A}_j(r_j)$ as the following:

$$\sum_{j=1}^{t} r_j'' \tilde{A}_j(r_j) = \sum_{z \in \{0,1\}^{s_{in}}} \tilde{V}_{in}(z) \sum_{j=1}^{t} r_j'' \tilde{C}_j'(r_j, z) \tag{11}$$

where r_j'' are random points send by the verifier. $C_j'(\cdot)$ returns 1 if the x-th value in A_j is the z-th value in V_{in}, otherwise returns 0. The sumcheck protocol reduces claims to $\tilde{A}_j$ to a single claim on $\tilde{V}_{in}(\cdot)$.

After invoking the Lasso protocol, we can check the consistency of lookup vectors leveraging the above method. For the details of the entire zkGPT protocol combining Lasso and GKR protocol, please refer to Protocol 4. For the security analysis of the zkGPT protocol, please refer to Section F.

F Security Analysis of zkGPT protocol

Proof. **Completeness**. The completeness is straightforward by the completeness of the GKR protocol and Lasso protocol. **Soundness**. For the analysis of soundness, our protocol could be viewed as the combination of GKR protocol and Lasso protocol. Suppose the prediction sent by $\mathcal{P}$ is not correctly computed, i.e., $y \neq pred(W,X)$, but still passes the verification. This event can be divided into four cases:

- Case 1: all the values in aux, L are computed correctly. Let $\tilde{X}_m^*()$ be the multi-linear extension of $X_m^* = y \neq X_m$, by the Schwartz-Zippel lemma [53], in Step (4) of Protocol 4, $\tilde{X}_m^*\left(r^{(m)}\right) \neq \tilde{X}_m\left(r^{(m)}\right)$ with all but probability $\frac{1}{|\mathbb{F}|}$. Since with high probability $\tilde{X}_m^*\left(r^{(m)}\right)$ is incorrect, and in is correct, due to the soundness of GKR, the checks in Step (5) and Step (7) of Protocol 4 can only pass with negligible probability.
- Case 2: some values in aux is incorrect, L is fully computed using this aux. In this case, due to the equivalence between our defined constraints and corresponding

Protocol 4 (zkGPT Protocol). *Let C be a circuit with m layers computing an LLM with model parameters $\mathbf{W}$. Let $X_1 = \mathbf{X}$ be the input embedding, $X_m = y$ be the model's output embedding $C(\mathbf{W}, \mathbf{X})$, and X_i be the output of the i-th layer of C.*

• $\mathsf{zkGPT.KeyGen}(1^\lambda)$*: Run* $\mathsf{pp} \leftarrow \mathsf{zkPC.KeyGen}(1^\lambda)$ *and outputs pp.*

• $\mathsf{zkGPT.Commit}(\mathbf{W}, \mathsf{pp}, r)$*: Define the multi-linear extension of* $\mathbf{W}$*, viewed as an array by concatenating the parameters of each layer, as* $\tilde{W}$*.* $\mathcal{P}$ *commits to* $\tilde{W}$ *by running* $\mathrm{com}_W \leftarrow \mathsf{zkGPT.Commit}\left(\tilde{W}, \mathsf{pp}, r_W\right)$ *and sends* $\mathcal{V}$ com_W*.*

• $\langle \mathsf{zkGPT.Prove}\left(\mathbf{W}, r_W\right), \mathsf{zkGPT.Verify}\left(\mathrm{com}_W\right)\rangle\left(\mathrm{X}, \mathsf{pp}\right)$ *:*

(1) Upon receiving $\mathbf{X}$ *from* $\mathcal{V}$*,* $\mathcal{P}$ *evaluates the LLM to compute the prediction* $y = X_m$*, the values in each layer* X_i*, the intermediate values aux, the lookup queries L. L is composed of* $L_1 \| L_2$*, where* L_1 *is the lookup values for the range check table,* L_2 *is the lookup values for the exponentiation table. P commits to the multi-linear extension of* aux, L *by running* $com_{aux} \leftarrow \mathsf{zkPC.Commit}(\tilde{aux}, pp, r_{aux}), com_L \leftarrow \mathsf{zkPC.Commit}(\tilde{L}, pp, r_L)$ *and sends* $\mathcal{V}$ com_{aux}, com_L *and* $y = X_m$*. Without loss of generality, we pad* $\mathbf{X}, \mathbf{W}$*,* $\mathbf{aux}$*,* $\mathbf{L}$ *to the maximum length of the four, denoted as* N*. We arrange the input of the circuit as* $\mathbf{X}\|\mathbf{W}\|\mathbf{aux}\|\mathbf{L}$ *of size* $4N$*.*

(2) For the range check table and exponentiation table, $\mathcal{P}$ *sends the Lasso-commitment to the multi-linear extension of matrix* $M_1 \in \{0,1\}^{m_1 \times N_1}, M_2 \in \{0,1\}^{m_2 \times N_2}$*, each consisting of* c *different* $\log(m)$ *)-variate multi-linear polynomials* $\dim_1, \ldots, \dim_c$*, respectively.* $\mathcal{V}$ *picks random* $r_1 \in \mathbb{F}^{\log m_1}, r_2 \in \mathbb{F}^{\log m_2}$ *and sends* r_1, r_2 *to* $\mathcal{P}$*.* $\mathcal{P}$ *claims to* $\mathcal{V}$ *on* $\widetilde{L_1}(r_1), \widetilde{L_2}(r_2)$*.*

(3) $\mathcal{P}$ *and* $\mathcal{V}$ *apply Protocol 3 on range check table and exponentiation table respectively.* $\mathcal{P}$ *proves that* $\sum_{y \in \{0,1\}^{\log N_1}} \widetilde{M_1}(r_1, y) T_1[y] = \widetilde{L_1}(r_1)$ *and* $\sum_{y \in \{0,1\}^{\log N_2}} \widetilde{M_2}(r_2, y) T_2[y] = \widetilde{L_2}(r_2)$*.*

(4) $\mathcal{V}$ *defines the multi-linear extension of* $y = X_m$ *as* $\tilde{X}_m$*.* $\mathcal{V}$ *chooses a random* $r^{(m)}$ *and sends it to* $\mathcal{P}$*. Both parties compute* $\tilde{X}_m\left(r^{(m)}\right)$*.*

(5) For $i = m, m-1, \ldots, 2$*, with* $\tilde{X}_i\left(r^{(i)}\right)$*,*

- *If layer* i *is a matrix multiplication layer,* $\mathcal{P}$ *and* $\mathcal{V}$ *run a sumcheck protocol on*

$$\tilde{X}_i(x, y) = \sum_z \tilde{X}_{i-1}(x, z) \cdot \tilde{W}_{i-1}(z, y)$$

  using the sumcheck algorithm for matrix multiplication proposed in [58]. At the end of the sumcheck, $\mathcal{V}$ *receives* $\tilde{X}_{i-1}\left(r^{(i-1)}\right)$ *and* $\tilde{W}_{i-1}\left(r^{(i-1)}\right)$*, where* $\tilde{W}_{i-1}$ *denotes the multi-linear extension defined by the model parameters* W_i *used in layer* i*.*

- *If layer* i *belongs to other constraint layers,* $\mathcal{P}$ *and* $\mathcal{V}$ *run the GKR protocol on Equation 9. At the end of the sumcheck,* $\mathcal{V}$ *receives* $\tilde{X}_{i-1}\left(r^{(i-1)}\right)$ *and* $\tilde{aux}_{i-1}\left(r^{(i-1)}\right)$*, where* aux_{i-1} *denotes the multi-linear extension defined by auxiliary input* aux_i *used in layer* i*.*

(6) At the input layer, $\mathcal{V}$ *has received* $\tilde{X}_1\left(r^{(1)}\right)$*,* $\tilde{W}_{i-1}\left(r^{(i-1)}\right)$*,* $\widetilde{aux}_{i-1}\left(r^{(i-1)}\right)$ *for* $i = m, \ldots, 1$*, and* $\widetilde{L_1}(r_1), \widetilde{L_2}(r_2)$*.* $\mathcal{P}$ *and* $\mathcal{V}$ *run a sumcheck protocol on Equation 10 to combine them into a single evaluation. At the end of the sumcheck,* $\mathcal{V}$ *receives* $\tilde{in}(r)$*, where* $\tilde{in}$ *denotes the multi-linear extension defined by the entire input of the circuit.*

(7) $\mathcal{V}$ *validates in* (r_{in}) *by opening the polynomial commitments. In particular, as the size of the input is* $4N$*, let* $r = \left(r_1, \ldots, r_{\log N+2}\right)$ *and* $r^- = \left(r_1, \ldots, r_{\log N}\right)$*,* $\mathcal{P}$ *and* $\mathcal{V}$ *run* $\langle \mathsf{zkPC.Open}(\tilde{W}, r_W), \mathsf{zkPC.Verify}(\mathrm{com}_W)\rangle(r^-, \mathsf{pp})$ *and* $\langle \mathsf{zkPC.Open}(aux, r_{aux}), \mathsf{zkPC.Verify}(\mathrm{com}_{aux})\rangle\left(r^-, \mathsf{pp}\right)$*.* $\mathcal{V}$ *receives* $\tilde{W}\left(r^-\right)$ *and* $\tilde{aux}\left(r^-\right)$*, and evaluates* $\tilde{X}_1\left(r^-\right)$ *locally.* $\mathcal{V}$ *checks that* $\tilde{in}(r) = \tilde{X}_1\left(r^-\right) \cdot \left(1 - r_{\log N+1}\right)\left(1 - r_{\log N+2}\right) + \tilde{W}\left(r^-\right) \cdot \left(1 - r_{\log N+1}\right) r_{\log N+2} + \tilde{aux}\left(r^-\right) \cdot r_{\log N+1}\left(1 - r_{\log N+2}\right) + \tilde{L}\left(r^-\right) \cdot \left(1 - r_{\log N+1}\right)\left(1 - r_{\log N+2}\right)$*.*

If all the checks pass, $\mathcal{V}$ *outputs* 1*; otherwise,* $\mathcal{V}$ *outputs 0.*

lookup relations, there must be some values in L inconsistent with the relations in two lookup tables. Due to the soundness of Lasso protocol, the checks in Protocol 3 can only pass with probability at most $O(\frac{m_1 + N_1^{\frac{1}{c}} + m_2 + N_2}{|\mathbb{F}|})$ which is negligible.

- Case 3: some values in *aux* is correct, L is inconsistent with this *aux*. Then some arithmetic relations checking L and *aux* are not satisfied. Due to the soundness of GKR, the checks in Step (5) and Step (7) of Protocol 4 can only pass with negligible probability.
- Case 4: some values in *aux* is incorrect, L is inconsistent with this *aux*. Similarly, some arithmetic relations check-

ing L and *aux* are not satisfied. Due to the soundness of GKR, the checks in Step (5) and Step (7) of Protocol 4 can only pass with negligible probability.

By the union bound, the probability of $\mathcal{V}$ outputting 1 when $y \neq pred(W,X)$ is negligible.

G Constraints for GeLU and Normalization

GeLU. GeLU is computed as $G = g \cdot \Phi(g)$. Directly proving it in ZKP is very challenging. Multiple approximations have been proposed but are not enough ZKP-friendly. We design a better approximation as $G = \frac{g+|g|}{2} - \mathbb{1}_{|g|>=\mathsf{th}} \cdot g \cdot (a \cdot |g|^2 + b \cdot |g| + c)$. For design details and comparisons, please see Appendix I. The constraints corresponding to our approximation are listed as follows:

$$\begin{aligned} s &= \mathbb{1}_{|q_g| \geq q_{\mathsf{th}}} \\ q_p &= \mathsf{round}(\frac{aS_g^2 q_g^2 + bS_g|q_g| + c}{S_p}) \\ q_D &= \mathsf{round}(S_p \cdot q_g \cdot q_p) \\ q_G &= \frac{q_g + Q_g}{2} - s \cdot q_D \end{aligned} \tag{12}$$

where $Q_g = |q_g|$, q_{th}, q_G share scale S_g with q_g, $q_{\mathsf{th}} = \mathsf{round}(\mathsf{th}/S_g)$. Comparison constraints like $y = \mathbb{1}_{A \geq B}$ could be checked by relations: $y(y-1) = 0, y(A-B) + (1-y)(B-A-1) \geq 0$.

Normalization. For matrix x, the normalization layer is computed as $y_{ij} = \gamma_j \cdot \frac{x_{ij} - \mu_i}{\sigma_i} + \beta_j$, where μ_i, σ_i is the mean and variance of row i of x, γ, β are vectors parameterizing the affine transform. The constraints are listed as follows:

$$\begin{aligned} q_{\mu_i} &= \mathsf{round}(\sum_{j=1}^{d} q_{x_{ij}}/d) \\ q_{t_i} &= \mathsf{round}(\sum_{j=1}^{d} (q_{x_{ij}} - q_{\mu_i})^2/d) \\ q_{\sigma_i} &= \mathsf{round}(\sqrt{q_{t_i}}) \\ q_{s_i} &= \mathsf{round}(\frac{q_{x_{ij}} - q_{\mu_i}}{S_s \cdot q_{\sigma_i}}) \\ q_{y_{ij}} &= \mathsf{round}(\frac{S_s S_\gamma}{S_y} \cdot q_{\gamma_j} \cdot q_{s_i} + \frac{S_\gamma}{S_y} \cdot q_{\beta_j}) \end{aligned} \tag{13}$$

where μ, σ share scale S_x with q_x, β share scale S_γ with γ.

H Constraints for Normalization and GeLU after fusion

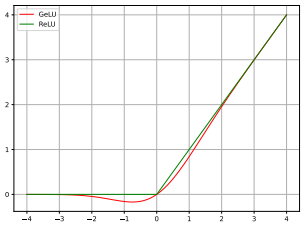
GELU After merging q_D with q_G, we are able to effectively reduce the number of GeLU rounding constraints from $2sd$ to sd. This reduction results in a nearly 50% decrease in the total number of rounding constraints required compared to the original GeLU.

$$\begin{aligned} s &= \mathbb{1}_{|q_x| \geq q_{\mathsf{th}}} \\ q_g &= \mathsf{round}(\frac{q_x + Q_x}{2} - s \cdot (Q_x(a \cdot S_x^2 q_x^2 + b \cdot S_x Q_x + c))) \end{aligned} \tag{14}$$

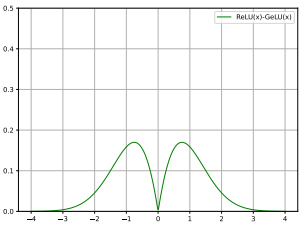
Normalization. The merging process for Normalization differs slightly from the usual strategy; After merging q_{μ_i} with q_{t_i} and q_{s_i}, we did not merge q_{σ_i} with $q_{y_{ij}}$ to prevent increasing the arithmetic and commitment overhead by a factor of N. As a result, we reduce the number of rounding constraints for normalization from $3s + 2sd$ to $s + sd$. This leads to nearly a 50% decrease in the total number of rounding constraints compared to the original normalization.

$$\begin{aligned} q_{\sigma_i} &= \mathsf{round}(\sqrt{\frac{\sum_{j=1}^{d}(q_{x_{ij}} \cdot d - \sum_{j=1}^{d} q_{x_{ij}})^2}{d^2}}) \\ q_{y_{ij}} &= \mathsf{round}(\frac{S_\gamma}{S_y} \cdot q_{\gamma_j} \cdot \frac{q_{x_{ij}} \cdot d - \sum_{j=1}^{d} q_{x_{ij}}}{q_{\sigma_i} \cdot d} + \frac{S_\gamma}{S_y} \cdot q_{\beta_j}) \end{aligned} \tag{15}$$

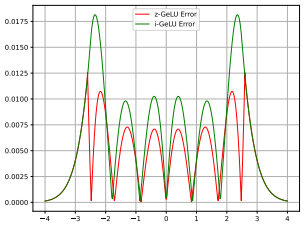
I GeLU approximations



(a) Comparison of GeLU and ReLU function.



(b) The difference function of ReLU and GeLU.



(c) Comparison of numerical error between our $z-GeLU$ and $i-GeLU$ [37].

Figure 4: GeLU numerical analysis.

For example, [32] suggests using Sigmoid function to approximate erf:

$$\mathrm{GeLU}(x) \approx x\sigma(1.702x) \tag{16}$$

where $\sigma(\cdot)$ is the Sigmoid function. This approximation, though adopted in zk-LLM [57], however, is sub-optimal for efficient ZKP, as the Sigmoid itself is another non-linear function.

[33] replaced sigmoid with a "hard" version obtaining another approximation function for GELU:

$$\text{h-GeLU}(x) := x\frac{\text{ReLU6}(1.702x+3)}{6} \approx \text{GELU}(x). \quad (17)$$

However as pointed out in previous literature [37], the accuracy drop of replacing GeLU with h-GeLU is high, due to the large gap between the two functions.

[37] proposed an approximation function for GeLU based on quadratic approximation and piece-wise function, as following:

$$\begin{aligned} \text{L}(x) &= \text{sgn}(x)\left[a(\text{clip}(|x|, \text{max} = -b) + b)^2 + 1\right] \\ \text{i-GeLU}(x) &= x \cdot \frac{1}{2}\left[1 + \text{L}\left(\frac{x}{\sqrt{2}}\right)\right] \end{aligned} \quad (18)$$

where $a = -0.2888$ and $b = -1.769$, and sgn denotes the sign function. Their method preserves decent accuracy and has been adopted in ZKP systems like Lu et al [42]. However, directly integrating i-GeLU into our circuit will introduce large overhead. The reason is that, their method invokes a large number of non-arithmetic operations, including division, sign function, absolute function, clipping. This will lead towards a very complicated circuit. This motivates the need of deriving a more ZK-friendly approximation for GeLU.

We observe that $y = GeLU(x)$ converges to $y = x$ for $x \to +\infty$, and converges to $y = 0$ for $x \to -\infty$. Therefore it converges to $y = ReLU(x)$ for $x \to +\infty$ and $x \to -\infty$. A natural idea occurs: could we use directly use ReLU to replace GeLU when the input absolute value is large, and use a low order polynomial to handle the case when the input absolute value is small.

From Figure 4b we can see that the difference function $D(x) = ReLU(x) - GeLU(x)$ is an even function. And we can see that $D(x)$ is in the small range of $[0, 0.2)$, and converges to 0 when $|x| \to +\infty$. When designing a simple function $D'(x)$ to approximate $D(x)$ on $[0, +\infty)$(we only consider approximation on the positive domain due to $D(x)$ is even), we can set $D'(x) = 0$ for $x \geq t$, where t is a threshold. On the range $x \in [0, t)$, we can fit $D(x)$ with a low-order polynomial. Considering $D(0) = 0$, we can set $D'(x) = x(a \cdot x^2 + b \cdot x + c), (x < t)$. Then the problem becomes:

$$\min_{a,b,c,t} \max_{x>0} |D'(x) - D(x)| \quad (19)$$

Considering the monotonicity of $D(x)$ when $x \in [t, +\infty)$, optimizing the above target is equivalent to the following problem:

$$\min_{a,b,c,t} \max\{\max_{x\in[0,t)} |x(a \cdot x^2 + b \cdot x + c) - D(x)|, D(t)\} \quad (20)$$

	z-GeLU	$x\sigma(1.702x)$	h-GeLU	i-GeLU
L_2 dist	**0.0054**	0.012	0.031	0.0082
L_∞ dist	**0.012**	0.020	0.068	0.018

Table 9: Comparison of different approximation methods for GELU. As metrics for approximation error, we report L_2 and L_∞ distance from GELU across the range of $[-4, 4]$.

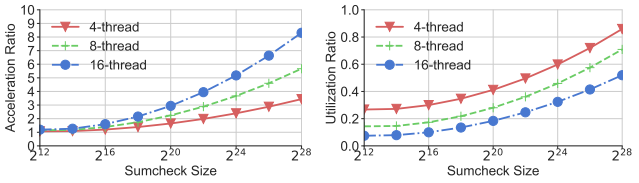
Table 10: Comparison of before and after fusion.

Rounding Numbers	Normalization	Attention	GeLU
Before Fusion	$3s + 2sd$	$\frac{3}{2}hs(s+1) + sd$	$2Ds$
	49224	43584	196608
After Fusion	$s + sd$	$\frac{1}{2}hs(s+1) + sd$	Ds
	24608	30912	98304

The inner max of the above function only needs to be computed on a small interval, making the optimization much more efficient.

We compare z-GELU along with existing approximations in Tab 9, where z-GELU has an average error of 5.4×10^{-3} and a maximum error of 1.2×10^{-2}. This is $1.7\times$ more accurate than current state-of-the-art i-GELU whose average and maximum errors are 8.2×10^{-2} and 1.8×10^{-2}, respectively.

J Details of implementation acceleration



(a) Acceleration ratio. (b) Utilization ratio.

Figure 5: Acceleration ratio and utilization ratio measured for sumcheck under different sizes and thread numbers. Utilization ratio is computed as $\frac{T_{\text{single thread}}}{T_{t\ \text{thread}} \cdot t}$.

The commitment at the input layer includes committing to input data, weight, and auxiliary input (advice). This procedure is quite time-consuming; the main bottleneck is multi-scalar multiplication (MSM). We leverage Pippenger algorithm [50] and multi-thread parallelism in our implementation to accelerate the commitment phase. Also, we observe that the commitment to weight data only needs to be done once for a single model. Thus, the prover can compute the commitment to the model weight during preprocessing. The weight commitment could be reused when serving different user requests.

For GKR protocol on ML-friendly circuit, the last step is to reduce claims on different layers and subsets of the input to one random claim on the input (see sumcheck on Equation 10 in Section 2.2). This step is also quite time-consuming without acceleration. We can leverage multi-thread parallelism to accelerate its computation.

K Proof of constraint fusion cost

Lemma K.1. *Representing the fused constraint of constraint 1 and constraint 2 in arithmetic circuit does not introduce extra multiplication and addition gates.*

Proof. The fused constraint is:

$$y = \mathsf{round}(\frac{(D_1 + D_2 \cdot D_3) \cdot D_4}{D_2 \cdot D_5} \cdot x + D_6) \tag{21}$$

Proving constraint 1 requires computing Δ_1 in the arithmetic circuit. Similarly, proving constraint 2 requires computing Δ_2. They are defined as follows:

$$\begin{aligned} \Delta_1 &= ((2x+1) \cdot D_2 - 2(D_1 + D_2 \cdot D_3) - 1) \cdot \\ &\quad (2(D_1 + D_2 \cdot D_3) - (2x-1) \cdot D_2) \\ \Delta_2 &= ((2y+1) \cdot D_5 - 2(D_4 \cdot x + D_6 \cdot D_5) - 1) \cdot \\ &\quad (2(D_4 \cdot x + D_6 \cdot D_5) - (2y-1) \cdot D_5) \end{aligned} \tag{22}$$

$$\begin{aligned} \Delta_3 &= ((2y+1)D_2 \cdot D_5 - 2((D_1 + D_2 \cdot D_3)D_4 \cdot x + D_2 \cdot D_5 \cdot D_6) - 1) \cdot \\ &\quad (2((D_1 + D_2 \cdot D_3)D_4 \cdot x + D_2 \cdot D_5 \cdot D_6) - (2y-1)D_2 \cdot D_5) \\ &= ((2y + 1 + 2D_6)D_2 \cdot D_5 - 2(D_1 + D_2 \cdot D_3)D_4 \cdot x - 1) \cdot \\ &\quad (2(D_1 + D_2 \cdot D_3)D_4 \cdot x - (2y - 1 - 2D_6)D_2 \cdot D_5) \end{aligned} \tag{23}$$

Define $C_*(P)$ as the number of multiplication gates required to compute multi-variate polynomial P from its input. For Δ_3 we have:

$$\begin{aligned} C_*(2(D_1 + D_2 \cdot D_3)D_4 \cdot x) = C_*(D_2) + C_*(D_3) + \\ C_*(D_1) + C_*(D_4) + 4 \end{aligned} \tag{24}$$

After computing the previous formula, D_2 is already computed, we have:

$$C_*((2y + 1 + 2D_6)D_2 \cdot D_5) = C_*(D_5) + C_*(D_6) + 4 \tag{25}$$

After computing D_5,$D_2 \cdot D_5$, $2y$ and $2D_6$, we have:

$$C_*((2y - 1 - 2D_6)D_2 \cdot D_5) = 2 \tag{26}$$

Thus, we have:

$$\begin{aligned} C_*(\Delta_3) &= C_*(2(D_1 + D_2 \cdot D_3)D_4 \cdot x) + C_*((2y + 1 + 2D_6)D_2 \cdot D_5) \\ &\quad + C_*((2y - 1 - 2D_6)D_2 \cdot D_5) + 1 \\ &= C_*(D_1) + \cdots + C_*(D_6) + 11 \end{aligned} \tag{27}$$

$$\begin{aligned} C_*(2(D_1 + D_2 \cdot D_3)) &= C_*(D_1) + C_*(D_2) + C_*(D_3) + 2 \\ C_*(\Delta_1) &= C_*(2(D_1 + D_2 \cdot D_3)) + 2 + 1 \\ &= C_*(D_1) + C_*(D_2) + C_*(D_3) + 5 \end{aligned} \tag{28}$$

Similarly , for Δ_2 we have:

$$C_*(\Delta_2) = C_*(D_4) + C_*(D_5) + C_*(D_6) + 6 \tag{29}$$

Thus we have $C_*(\Delta_3) \leq C_*(\Delta_1) + C_*(\Delta_2)$.
For addition gates we can perform similar analysis.

More importantly, when D_2 equal D_4 or D_2 and D_4 share common factors, Equation 21 can be further simplified, reducing $C_*(\Delta_3)$. Therefore, in practice, fusing Type-II constraints does not introduce extra overhead on computing the arithmetic circuit, and improves the overall performance by mitigating invocations to the lookup table. Constraint fusion also reduces the number of advice and lookup values committed in the input, thus accelerating the commitment procedure.

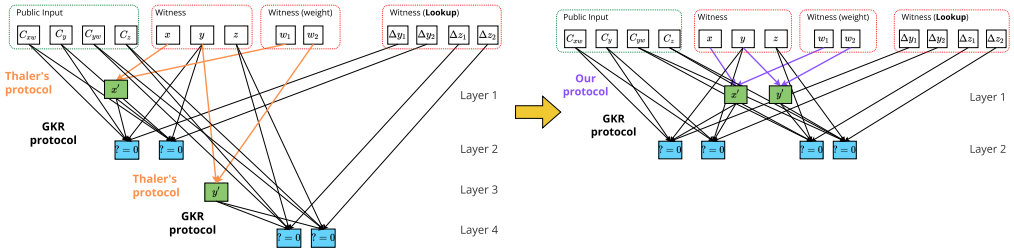
L Efficient matrix multiplication protocol

M Example for circuit squeezing

Example of circuit squeeze. At last, we offer an example of circuit squeeze. Assume that the constraints corresponding to the computation of $x \to y \to z$ are as follows: (1) $q_{x'} = x \cdot w_1$. (2) $q_y = \mathsf{round}(\frac{C_{xw}}{C_y} q_{x'})$. (4) $q_{y'} = y \cdot w_2$. (5) $q_z = \mathsf{round}(\frac{C_{yw}}{C_z} q_{y'})$, where x, w_1, y, w_2, z are matrices, C_{xw}, C_y, C_{yw}, C_z are integers. Checking $x \to y \to z$ is correct is transformed to checking the relations in Figure 6a. And the subcircuit corresponding to these relations is shown in Figure 6b. Figure 6b shows that the subcircuit checking $y \to z$ does not rely on x'. Thus we can squeeze the subcircuits checking $x \to y$ and $y \to z$ in Figure 6b into Figure 6c. Note that without our designed protocol, it is impossible to merge y' into the same layer of x' since the original protocol in [58] only supports checking a single matrix multiplication.

$$x' = x \cdot w_1$$
$$\Delta y_1 = C_y(2q_y + 1) - 2C_{xw} \cdot q_{x'} - 1 \geq 0$$
$$\Delta y_2 = 2C_{xw} \cdot q_{x'} - C_y(2q_y - 1) \geq 0$$
$$y' = y \cdot w_2$$
$$\Delta z_1 = (2q_z + 1)C_z - 2C_{yw} \cdot q_{y'} - 1 \geq 0$$
$$\Delta z_2 = 2C_{yw} \cdot q_{y'} - (2q_z - 1)C_z \geq 0$$

(a) Relations to check



(b) Before circuit squeeze (c) After circuit squeeze

Figure 6: Toy example of circuit squeeze.

Algorithm 1 *Efficient algorithm for bookkeeping table* $\tilde{A}(r||\cdot)$

Input: Matrix A, number of rows before padding n, number of rows before padding m, random evaluation point r, quantization bits Q (quantization range: $[0, 2^Q]$).
Output: array $\mathsf{BK}[2^{\lceil \log m \rceil}]$, $\mathsf{BK}[c] = \tilde{A}(r||c)$
Initialize array BK as zeros
Partition $r = x||y$
Compute $\tilde{eq}(x,\cdot)$, $\tilde{eq}(y,\cdot)$ using the memorization method in [60], stored in table $\mathsf{EQ_x}[2^{\lceil \frac{\log n}{2} \rceil}], \mathsf{EQ_y}[2^{\lceil \frac{\log n}{2} \rceil}]$
Initialize array $\mathsf{FM}[2^{\lceil \frac{\log n}{2} \rceil}][2^Q + 1]$ with zeros
Compute $\mathsf{FM}[R][t] = \tilde{eq}(y,R) \cdot t$:
for $R \in \{0,1\}^{\lceil \frac{\log n}{2} \rceil}$ **do**
  for $t \in \{1, \cdots, 2^Q\}$ **do**
    $\mathsf{FM}[R][t] = \mathsf{FM}[R][t-1] + \mathsf{EQ_y}[R]$
  end for
end for
for $c \in [0, m-1]$ **do**
  Initialize array $\mathsf{AC}[2^{\lceil \frac{\log n}{2} \rceil}]$ with zeros
  Compute the inner dot product:
  for $b \in [0, n-1]$ **do**
    Partition $bin(b) = b_L||b_R$
    $\mathsf{AC}[b_L] += \mathsf{FM}[b_R][A[b][c]]$
  end for
  for $L \in \{0,1\}^{\lceil \frac{\log n}{2} \rceil}$ **do**
    $\mathsf{BK}[c] += \mathsf{EQ_x}[L] \cdot \mathsf{AC}[L]$
  end for
end for
return BK