HAAN: A Holistic Approach for Accelerating Normalization Operations in Large Language Models

Tianfan Peng[1,2†], Tianhua Xia[1†], Jiajun Qin[1,3†], Sai Qian Zhang[1*]
[1]Tandon School of Engineering, New York University, New York, USA
[2]Shenzhen Institute of Information Technology, Shenzhen, China
[3]Zhejiang University, Hangzhou, China
Email: {tianfanpeng, hobbitqia}@gmail.com, {tx856, sai.zhang}@nyu.edu

***Abstract*—Large language models (LLMs) have revolutionized natural language processing (NLP) tasks by achieving state-of-the-art performance across a range of benchmarks. Central to the success of these models is the integration of sophisticated architectural components aimed at improving training stability, convergence speed, and generalization capabilities. Among these components, normalization operation, such as layer normalization (LayerNorm), emerges as a pivotal technique, offering substantial benefits to the overall model performance. However, previous studies have indicated that normalization operations can substantially elevate processing latency and energy usage. In this work, we adopt the principles of algorithm and hardware co-design, introducing a holistic normalization accelerating method named *HAAN*. The evaluation results demonstrate that HAAN can achieve significantly better hardware performance compared to state-of-the-art solutions.**

I. Introduction

The superior performance of large language models (LLMs) is accompanied by increased computational and memory consumption. The widespread adoption of LLMs has pushed transformer sizes to unprecedented scales, often reaching multiple billions and continuing to grow. Besides the growing model size, LLM computation involves a blend of linear matrix multiplication and non-linear operations, including normalization, softmax, and GeLU. In contrast to other DNN architectures such as Convolutional Neural Networks (CNNs), where matrix multiplication dominates, transformers dedicate a considerable portion of their runtime to nonlinear operations [1]–[3].

Among the numerous normalization operations proposed, Layer Normalization (LayerNorm) [4] and Root Mean Square Normalization (RMSNorm) [5] are two most widely employed normalization techniques in LLM. LayerNorm adjusts the input vectors of hidden layers by re-centering and rescaling them, thereby producing outputs with zero mean and unit variance. This process mitigates internal covariate shift issues and greatly enhances the training efficiency [6]. Conversely, RMSNorm adopts a more efficient approach by solely rescaling the input vector using its root mean square (RMS) value, offering superior computational efficiency compared to LayerNorm.

Non-linear operations in transformers, particularly LayerNorm and softmax, are notoriously time-consuming [1], [2], [7], [8]. The recent surge in popularity of FlashAttention [9] has reduced softmax latency by 80% by integrating it with matrix multiplication, and several hardware accelerators [10]–[14] have further optimized softmax. Additionally, multiple methods have been proposed to facilitate matrix multiplication operations within LLMs by representing them with extremely low precision (e.g., FP8, INT8) [15]–[18]. However, efficient solutions for LayerNorm are still limited.

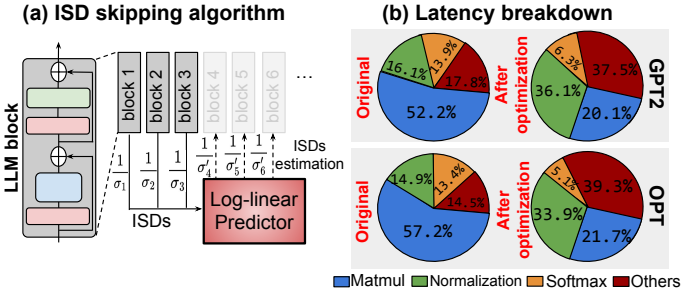

Fig. 1: (a) Standard deviation skipping for efficient normalization. (b) Runtime breakdown for GPT-2 and OPT execution with and without applying optimization techniques, using a sequence length of 2048.

To demonstrate the cost of normalization operations in LLM execution, we profile the GPT2-117M and OPT-2.7B models from Huggingface [19] using half-precision (FP16) on an A100 GPU. As illustrated in the latency breakdown in Figure 1(b), LayerNorm computation constitutes a substantial portion of model runtime (approximately 16% in the GPT2 model). Additionally, we evaluate the latency of LayerNorm by applying optimization techniques such as FlashAttention and FP8 quantization for linear layers within the LLM. The results indicate that, while these optimizations greatly reduce the processing latency of softmax and Matmul operations, LayerNorm becomes a critical bottleneck, contributing to more than 33% of the overall latency. Thus, speeding up normalization is essential. Accelerating the normalization operation poses a considerable challenge, primarily due to the complexity of hardware implementation. This complexity stems from two main factors. Firstly, the square root and division operations required for variance computation are computationally intensive and resource-demanding. Secondly, the data dependencies in normalization hinder effective pipelining and parallelism, causing high latency and energy consumption.

In this paper, we introduce *HAAN*, a holistic approach for accelerating normalization operation within LLM. In particular,

†Equal Contribution
*Corresponding author: sai.zhang@nyu.edu

978-3-9826741-0-0/DATE25/© 2025 EDAA

our contribution can be summarized as follows:

- Our analysis reveals a strong correlation in the input statistics across consecutive attention layers within LLMs. Leveraging this insight, we introduce an efficient technique to estimate normalization statistics by utilizing the statistics of early LLM layers (Figure 1). This approach substantially mitigates the computational cost of the normalization operation with negligible accuracy degradation.
- We demonstrate that subsampling the LayerNorm input for computing input statistics leads to significant reductions in latency and power consumption during hardware implementation, with minimal impact on accuracy.
- The HAAN accelerator uses a hybrid of fixed and floating-point formats to streamline normalization computations, while offering configurability for selecting the optimal numeric format and input subsampling rate for LayerNorm.
- Evaluation results show that HAAN reduces power consumption by over 60% and latency by 20% compared to baseline methods across various LLM workloads, while maintaining superior accuracy.

II. Background and Related Work

A. Normalization Operation within LLM

Normalization operation has been widely utilized in the modern LLM architecture. Normalization can help in faster convergence during training by reducing the internal covariate shift problem [4]. It ensures that the distributions of inputs to each layer remain stable throughout training. Two common normalization operations are widely employed: LayerNorm and RMSNorm [5]. RMSNorm is utilized in LLMs such as the LLaMA series [20], [21] and Mistral [22], while LayerNorm is employed in LLMs like OPT [23], the GPT series [24], and Megatron-LM [25]. Specifically, for layer normalization function, the function takes a vector $\boldsymbol{z} = [z_1, \cdots, z_N]^T$ and generates an output $\boldsymbol{s} = [s_1, \cdots, s_N]^T$, both have a length of N. It is defined as follows:

$$\boldsymbol{s} = \alpha \frac{\boldsymbol{z} - \mu_z}{\sigma_z} + \beta \tag{1}$$

where $\mu_z = \frac{\sum_i z_i}{N}$ is the mean of the vector $\boldsymbol{z}$, $\sigma_z = \sqrt{\frac{\sum_i (z_i - \mu_z)^2}{N}}$. α and β are learnable vectors which has the identical shape as $\boldsymbol{z}$. α and β will both be fixed during the LLM execution. To apply LayerNorm on the input with a dimension $B \times L \times E$, the LayerNorm will apply to the $B \times L$ vectors, with each has a length of E, and B and L denote the batch size and token length, respectively. The intermediate results will then proceed with affine transformation by multiplying with a $E \times 1$ vector α and sum with a $E \times 1$ vector β.

RMSNorm adopts a more efficient strategy by rescaling the input vectors using only the RMS value of the input vector. Instead of re-centering the data, it directly normalizes the activations based on their RMS value. This approach provides higher computational efficiency compared to LayerNorm. Specifically, RMSNorm can be expressed as:

$$RMSNorm(\boldsymbol{z}) = \alpha \frac{\boldsymbol{z}}{r_z} + \beta \tag{2}$$

where $r_z = \sqrt{\frac{\sum_i (z_i)^2}{N}}$ is the RMS value of $\boldsymbol{z}$.

B. Efficient Normalization within LLM

Multiple studies have been conducted to accelerate the normalization operation within the transformer architecture. The previous methods including DFX [2], [26] and [27] modified the way for variance computation, which further improves the parallelism and reduce the processing latency of the layernorm computation. In [7], the intermediate results are dynamically compressed for the variance computation with low precision, leading to a reduction on energy and latency consumption. In [28], the parameters of LayerNorm is quantized using 8 bits. However, this will produce a degradation on the accuracy performance. All the previous mentioned work are proposed for conventional transformer acceleration, which has entirely different scale and data distribution from LLM. These methods often involve rigid quantization and approximate computation, which can be costly for LLMs due to their large size and the high expense of fine-tuning. HAAN selects skipped normalization layers offline with minimal accuracy impact and can be reconfigured to optimize precision and input subsampling for LayerNorm without additional fine-tuning costs.

III. HAAN Algorithm for Efficient Normalization

In this section, we first present a statistical study on input distribution of the normalization layers (Section III-A). We then demonstrate that variance computation within certain normalization operations can be skipped and propose methods to estimate the variance for these layers (Section III-B). Finally, we introduce quantization over the operands to facilitate normalization computation (Section III-C).

A. A Study on Input Activation Statistics

To motivate our variance skipping algorithm, we examine the trend on the variance of the input activations across all the normalization layers within the LLM. Conducting a statistical analysis in this regard could provide us with additional insights into the correlation between input variance across different layers, thereby enriching our understanding and enabling the generation of more effective optimization strategies. Specifically, rather than variance, we focus on the **inverse of the standard deviation (ISD)**, namely $\frac{1}{\sigma}$, which is directly proportional to the normalization results, as shown in equation 1 and equation 2. More importantly, our profiling results of the normalization operation runtime on GPU reveal that ISD computation accounts for more than 90% of the overall normalization runtime. Figure 2 illustrates the distribution of $\frac{1}{\sigma}$ for each of 64 normalization layers within the LLaMA-7B across different tokens. The x-axis represents the layer index, while the y-axis indicates the $\frac{1}{\sigma}$ on a logarithmic scale. We notice that:

- In general, the ISD values decrease with layer depth, dramatically in the initial layers, then flatten in later layers.
- The logarithm of ISD values in the deeper layers of the LLM exhibits a pronounced negative linear relationship.

Furthermore, we have replicated the experiment across several other language models, such as GPT-2, OPT, and others,

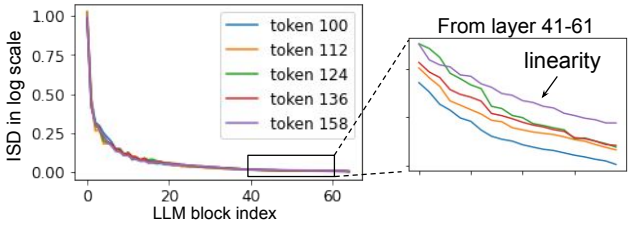

Fig. 2: ISD values across different normalization layers within the LLaMA-7B model. Give tokens are chosen randomly. We notice that the ISD values of the later layers reveals a linearity in logarithm domain.

all of which demonstrate a consistent pattern. The second observation suggests that ISD for the later layers is **highly predictable**. This trend in ISD reflects the natural evolution of feature representation within the LLM across various layers: weak token correlation in the initial LLM layer results in pronounced changes in ISD; in the middle layers, as the attention mechanism becomes more intricate, the feature representation stabilizes, leading to linearized changes in ISD; for the deeper layers that are near the end of the LLM outputs, adjustments in feature levels induce significant fluctuations in ISD, a phenomenon associated with the softmax function's capacity to enhance discriminative features [29]. Consequently, we can dynamically skip certain ISD computations and use predicted values, significantly reducing computational costs.

B. Variance Prediction across Layers

In Section III-A, we demonstrate that the ISD exhibits an almost linear relationship when plotted on a logarithmic scale for the latter layers of LLM. This pattern presents a great opportunity for bypassing certain ISD calculations and substituting them with predicted values. Nonetheless, three key questions arise:

- Which ISD can be skipped?
- How can the skipped ISDs be estimated?
- Will the method exhibit generality across different input samples and datasets?

To address the first question, we propose the *ISD skipping algorithm* described in Algorithm 1. A calibration set with S input samples are first fed into the LLM, and their corresponding ISDs are gathered for training the coefficient of the *ISD predictor*. To pinpoint the ISDs to be skipped, we compute the *Pearson correlation coefficients* across various layer ranges. Subsequently, we identify the range exhibiting the most negative correlation, signifying a significant degree of negative linearity among the ISD values, thus indicating the potential candidates for skipping. Specifically, Algorithm 1 iterates over the Pearson correlation coefficients for each pair of ISD values between a pair layer whose gap is larger than M, and returning a pairs of layer ranges whose ISD values yield a smallest Pearson correlation coefficient. The ISD computation within this range can be safely skipped and replaced with the predicted values. Moreover, the algorithm also computes the linear gradient of changes in ISD with respect to the layer index gaps utilizing the function *calDecay*.

To estimate the ISD value, when the layer index k falls within

Algorithm 1 ISD Skipping Algorithm

Input: Calibration set **S**, LLM model with L layers, $F_l(.)$, skip range M
Output: Skipping range (i_f, j_f), coefficient e

Initialize $ISDLists \leftarrow \emptyset$, $minCor \leftarrow 1$
for all $s \in S$ **do**
  for all $l \in L$ **do**
    Compute $F_l(s)$, record $\log(v_l)$ in $ISDLists$
for $i \leftarrow 1$ to $L - M$ **do**
  $j \leftarrow i + M$
  if $Pearson(ISDLists[i:j], [i:j]) < minCor$ **then**
    $minCor \leftarrow Pearson(ISDLists[i:j], [i:j])$
    $(i_f, j_f) \leftarrow (i, j)$
    $e \leftarrow$ calDecay$(ISDLists[i:j])$
return (i_f, j_f), e

the skip range (i, j), the ISD value of the layer k $(i \leq k \leq j)$ can be estimated using the following equation:

$$log(\text{ISD}_k) = log(\text{ISD}_i) + \text{e}_{ij} \times (k - i) \tag{3}$$

where i, j represent the start and end layer indices within the range. ISD_i and ISD_i are the ISDs of layer i and j. e_{ij} denotes the gradient returned by the $calDecay$ function. Finally, we demonstrate that the ISD predictor exhibits high generalizability across different datasets. Specifically, the ISD predictor trained using the calibration set from one dataset can effectively perform well on other downstream tasks. The evaluation results are detailed in Section V-A.

C. Efficient Normalization with Subsampling and Quantization

In addition to the ISD skipping method outlined in Section III-B, two additional approaches are introduced in this section to improve the efficiency of normalization. First, for the remaining ISD values that cannot be skipped, their values can be estimated by using a subsampled version of the input:

$$\text{ISD}_{sub} = \frac{1}{\sqrt{\frac{1}{N_{sub}} \sum_{a=1}^{N_{sub}} z_a^2}} \tag{4}$$

where ISD_{sub} represents the estimated version of the ISD using the subsampled inputs, N_{sub} denotes the number of input samples used for ISD estimation, and z_a signifies the a-th input $\boldsymbol{z}$. Empirically, a minimal N_{sub} is chosen to maintain a negligible impact on perplexity (PPL). It is important to note that the optimal value of N_{sub} varies between different LLM models, as discussed in Section V-A. To implement the subsampling operation on the input, we simply truncate the first N_{sub} elements within the input.

Moreover, subsampling can also aid in streamlining mean computation within LayerNorm. As demonstrated in Section V-A, a substantial portion of the input can be discarded to estimate of both mean and ISD with negligible impact on LLM accuracy. In addition to subsampling, proper quantization of operands during normalization is also applied to reduce implementation costs.

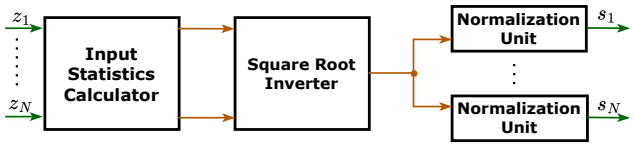

Fig. 3: Data path of HAAN where N denotes the size of embedding dimensions. In this figure, the floating-point and fixed-point data paths are highlighted in green and orange, respectively. The control signals are highlighted in black.

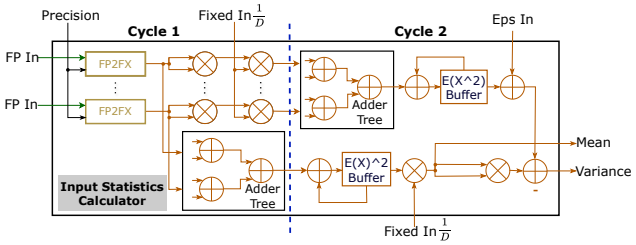

Fig. 4: Hardware design for Input Statistics Calculator.

Our normalization hardware accelerator, as detailed in Section IV, can effectively accommodate all of the optimization techniques described above, including ISD skipping, subsampling, and quantization [30]. Additionally, it is reconfigurable in the sense that it can also execute standard normalization computation without employing any optimization techniques, this guarantees the robustness of the accelerator performance.

IV. HAAN Hardware Accelerator

This section details the hardware design for accelerating the normalization operation. The overall architecture of the HAAN accelerator is shown in Figure 3. The system consists of three main elements: an *input statistic calculator*, a *square root inverter*, and *normalization units*. The input statistic calculator computes the mean and variance of the input tensor. The square root inverter uses these intermediate results to produce the ISD, and the normalization units generate the normalized input and perform the affine transformation. Our hardware design is highly configurable, supporting input and output in either fixed-point or floating-point (FP16 or FP32) formats while maintaining intermediate computational results in fixed-point representation. To reduce computational costs, the input statistics calculator and normalization units allow statistics to be collected from a subset of the original input. Combined with the techniques in Section III-C, this approach significantly reduces hardware costs with minimal impact on accuracy.

A. Input Statistics Calculator

The purpose of the Input statistic calculator is to generate the mean and variance for a D-dimensional vector $\boldsymbol{z}$, the hardware design is shown in Figure 4. Specifically, to reduce the processing latency, the variance of $\boldsymbol{z}$ can be expressed with the following equation:

$$Var(\boldsymbol{z}) = E(\boldsymbol{z}^2) - (E(\boldsymbol{z}))^2 = \sum_i \frac{z_i^2}{N} - (\frac{1}{N}\sum_i z_i)^2 \quad (5)$$

where N is the input dimension of $\boldsymbol{z}$. The input statistics calculator will compute $\sum_i \frac{z_i^2}{N}$ and $(\frac{1}{N}\sum_i z_i)^2$ in parallel. During execution, the inputs $\boldsymbol{z}$ in FP format are first converted to fixed-point format using FP2FX units. If the inputs are already in fixed-point format (INT8), the FP2FX units will bypass the conversion. Two multiplication units then process each element z_i of $\boldsymbol{z}$ to generate $\frac{z_i^2}{N}$. Since N is predetermined, $\frac{1}{N}$ can be precomputed and stored in memory. If N is a power of 2, division is implemented with a shift operation. An adder tree is then used to calculate $\sum_i \frac{z_i^2}{N}$. Concurrently, another adder tree concurrently computes $\sum_i z_i$ and the mean. The multiplication unit then calculates $(\frac{1}{N}\sum_i z_i)^2$, followed by a subtractor to produce the variance of $\boldsymbol{z}$. Due to the high dimensionality of $\boldsymbol{z}$ in LLMs, the mean and variance computation occurs over multiple passes, with interim results stored in local registers. For RMSNorm, the mean is not required and can be omitted. Using the subsampling techniques from Section III reduces passes and cycles, further decreasing processing latency.

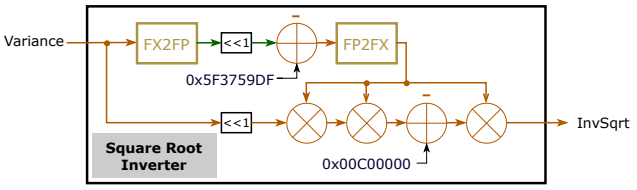

Fig. 5: Hardware design of Square Root Inverter. In the figure $0x00C00000$ equals 1.5 in fixed-point format.

B. Square Root Inverter

The variance produced from the input statistics calculator will be further processed by the square root inverter, which will accept FP number x and produce the output $y = \frac{1}{\sqrt{x}}$. Before the processing start, the FX2FP units will first convert the variance into FP format. Denote the E_x and M_x and exponent and mantissa fields of x. For positive x, we have $x = 2^{E_x - Q}(1 + M_x/2^L)$, where Q is the bias and L is the bit length of mantissa field. Then $\log_2 x = E_x - Q + \log_2(1 + M_x/2^L) \approx E_x - Q + M_x/2^L + \sigma_x = \frac{2^L(E_x - Q) + M_x}{2^L} + \sigma_x$. Where σ_x can be approximated as a constant 0.450465 when $M_x/2^L \in [0, 1]$ with negligible error [31]. To compute $y = \frac{1}{\sqrt{x}}$, we first calculate the $\log_2(y)$, which can be derived as follows:

$$\log_2(y) = -\frac{1}{2}\log_2(x) \quad (6)$$

$$\frac{2^L(E_y - Q) + M_y}{2^L} + \sigma_y = -\frac{1}{2}(\frac{2^L(E_x - Q) + M_x}{2^L} + \sigma_x) \quad (7)$$

$$\begin{aligned} M_y + 2^L E_y &= -\frac{1}{2}(M_x + 2^L E_x) + 2^L(\frac{3}{2}Q - (\sigma_y + \frac{1}{2}\sigma_x)) \\ &\approx -\frac{1}{2}(M_x + 2^L E_x) + \frac{3}{2}2^{23}(127 - 0.450465) \\ &\approx 0 \times 5f3759df - \frac{1}{2}(M_x + 2^L E_x) \end{aligned} \quad (8)$$

where $M_y + 2^L E_y$ is the bit representation of the FP format of y. To enhance the accuracy of the inverted square root operation, we first convert the y from the FP format to fixed-point format, and utilize Newton's method to refine the initial approximation obtained from equation 8. Let $f(y) = \frac{1}{y^2} - x$, and denote y_0 as the initial value returned from equation 8. Following Newton's method, the subsequent value can be computed as follows:

$$y_1 = y_0 - \frac{f(y_0)}{f'(y_0)} = y_0 - \frac{1/y_0^2 - x}{-2/y_0^3} = y_0(1.5 - xy_0^2) \quad (9)$$

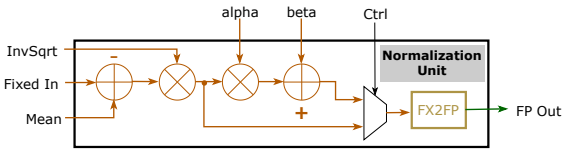

Fig. 6: Hardware design of the Normalization Unit

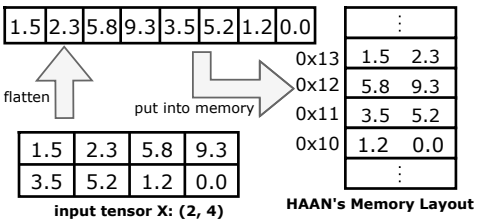

Fig. 7: HAAN memory layout for one sample $\boldsymbol{x}$ with a dimension of 2×4.

Based on the evaluation results, we observe that a single iteration is adequate to achieve accurate results. The architecture of Square Root Inverter is depicted in Figure 5.

To support the layer skipping methods from Section III-B, we design a custom unit to calculate predicted ISD using previous statistics. It employs the coefficient e of the ISD predictor and ISD values from early layers, leveraging the Xilinx Floating-point IP Core for linear prediction in the logarithm domain. The prediction is sent to the normalization unit. The ISD predictor is a scalar processor with minimal hardware cost.

C. Normalization Unit and Memory Layout

The normalization unit receives external inputs and outputs from the input statistics calculator and square root inverter to generate normalized results. For certain normalization layers, the ISD skipping algorithm in Algorithm 1 allows direct prediction of ISD values, bypassing the square root inverter. The normalized input then undergoes affine transformation by multiplying by α and adding β. When quantization is enabled, outputs remain in fixed-point format, skipping conversion in the FX2FP units. The input statistics calculator, square root inverter, and normalization unit operate in a pipelined manner across multiple input samples.

To support HAAN execution, we design an efficient storage format shown in Figure 7. The input tensor is flattened into a vector, with each memory entry containing a chunk equal to the input bandwidth of the HAAN accelerator. This allows the accelerator to access and process one memory entry per cycle. Figure 7 shows storing an input sample with dimensions 2×4, where the accelerator processes a vector of length two per cycle. In subsampling mode, only the initial portion of memory entries is accessed for computing input statistics.

V. EXPERIMENTAL RESULTS

In this section, we first evaluate the HAAN algorithm in Section V-A. Next, we present the hardware system evaluation in Section V-B.

A. Accuracy Evaluation

1) Accuracy evaluation over different LLMs and tasks

We run simulations of the HAAN algorithm, as detailed in Section III, using the Huggingface library. All pretrained

model	method	WG	PQ	HS	A-e	A-c
LLaMA-7B	Original	0.7017	0.7867	0.5694	0.7517	0.4198
	HAAN	0.7016	0.7818	0.5696	0.7567	0.4163
OPT-2.7B	Original	0.6093	0.7367	0.4581	0.6073	0.2696
	HAAN	0.6085	0.7318	0.4582	0.5997	0.2713
GPT2-1.5B	Original	0.5833	0.7084	0.4004	0.5829	0.2500
	HAAN	0.5801	0.7065	0.3997	0.5779	0.2554

TABLE I: Accuracy evaluation of HAAN on PIQA (PQ), WinoGrande (WG), HellaSwag (HS), Arc-Easy (A-e), Arc-Challenge (A-c) tasks.

Method	Config	WG	PQ	HS	A-e	A-c
Subsample length	128	0.5722	0.6654	0.4067	0.4520	0.2432
	256	0.7016	0.7818	0.5696	0.7567	0.4163
	512	0.7015	0.7828	0.5691	0.7513	0.4168
Data format	INT8	0.7016	0.7818	0.5696	0.7567	0.4163
	FP16	0.7016	0.7826	0.5691	0.7545	0.3963
	FP32	0.7017	0.7862	0.5691	0.7511	0.4198
Skip range	$(10, 20)$	0.5018	0.5818	0.3496	0.5032	0.2512
	$(30, 40)$	0.6218	0.7018	0.4896	0.6767	0.2675
	$(50, 60)$	0.7016	0.7818	0.5696	0.7567	0.4163

TABLE II: LLaMA-7B accuracy across different configurations.

LLMs are downloaded from Huggingface's official repository. To assess LLMs' accuracy, we consider five tasks: PIQA [32], WinoGrande [33], HellaSwag [34], and Arc (Easy and Challenge) [35]. We employ the evaluation [36] with default parameters for our experiments on NVIDIA RTX 3090Ti GPUs.

For the ISD skipping algorithm outlined in Algorithm 1, we random select 100 samples from the Wikitext dataset and use it as the calibration set. Subsequently, we execute the ISD algorithm across various LLMs and record the skip range for each. This recorded range will be applied to streamline the normalization computation across different downstream tasks. Specifically, for the LLaMA-7B model, we utilize the first $N_{sub} = 256$ input sample with a skip range of $(i_f, j_f) = (50, 60)$ to estimate the ISD. Furthermore, we apply INT8 quantization over the input. For OPT-2.7B model, we utilize the first $N_{sub} = 1280$, with the skip range adjusted to $(i_f, j_f) = (55, 62)$, and FP16 precision was employed. The GPT2-1.5B model is configured with a $N_{sub} = 800$ and a skip range of (85, 92), also utilizing FP16 precision.

Each LLM's performance is compared with the original setting in Table I, where the original denotes the FP32 LLM without ISD skipping and subsampling. The results show HAAN achieves nearly the same accuracies ($< 1\%$ degradation) as the original LLM across three models and five tasks. This demonstrates that HAAN is highly reliable.

2) Ablation studies

In this section, we investigate the impact of skip range (i_f, j_f), subsampling length (N_{sub}) and data format over the LLM accuracy. We choose LLaMA to study and evaluate its accuracy with different configurations. The evaluation results are shown in Table II. We have the following observation: first, N_{sub} has a significant impact over the accuracy. Particularly, $N_{sub} = 256$ achieves a great balance between efficiency and precision. Furthermore, in terms of data precision, INT8, FP16 and FP32 all achieve comparable results. Finally, the selection of the skip range is critical: skipping ISD computation in the early part of normalization layers of the model severely harm the model accuracy, leading to a failure on A-c task. Skipping in the middle also causes a notable accuracy drop.

B. Hardware Evaluation

In this section, we describe the hardware performance of the HAAN accelerator. We implement HAAN on the Xilinx Alveo U280 board and synthesize our design using the Xilinx Vivado

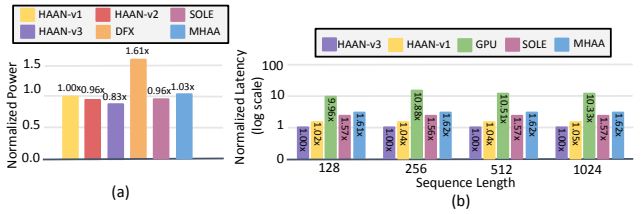

Fig. 8: (a) Normalized power comparison. (b) Normalized latency of HAAN compared to GPU on OPT-2.7B. The results of *HAAN-v2* are excluded from our evaluation since its configuration is incompatible with GPT2 model.

Suite. The HAAN accelerator operates at a 100MHz clock frequency. The HAAN accelerator is highly reconfigurable, supporting different input data widths for both the input statistics calculator and the normalization unit, denoted by p_d and p_n respectively. When subsampling is enabled, the parallelism of the input statistics calculator can be reduced without increasing processing latency, freeing up hardware resources (e.g., DSP) for more normalization units with more pipeline levels. The HAAN accelerator also supports various input data formats, including FP32, FP16, and INT8.

1) Self-Evaluation Under Different Configurations

We provide a comprehensive evaluation by varying the data formats, token length and subsampling rate over the normalization layer input. We report the corresponding hardware resource usage and power consumption. The corresponding resource and power costs under different settings are reported in Table III. We notice that performing the normalization in FP32 consumes $1.29\times$ more power than FP16 processing on average, and the power consumption increases moderately as the input sequence length grows. Overall, the normalization in INT8 achieves the least hardware and power consumption.

2) Performance comparison with other systems

We assess the hardware performance of the HAAN accelerator against other accelerators for normalization operation over GPT-2 and OPT. Most previous accelerators for LayerNorm [7] [37] [38] are either designed and evaluated with ASIC or not revealed the detailed architecture design [39] [40]. To ensure a fair comparison, we compare HAAN accelerator with DFX [2] by extracting the latency of LayerNorm from the overall latency reported in their study. We also reproduce SOLE [7] and MHAA [40] while aligning with HAAN's settings to ensure a fair comparison.

With the ISD skipping algorithm, ten normalization layers in GPT-2 can be skipped, and the input can be subsampled by half of its length. This modification does not cause noticeable degradation ($\leq 0.5\%$) in LLM accuracy on five downstream tasks described in Section V-A. For a comprehensive evaluation, we include two configurations for our HAAN accelerator, denoted as *HAAN-v1* and *HAAN-v2*. Both HAAN-v1 and HAAN-v2 adopt a single pipeline and accept normalization input in FP16. The (p_d, p_n) are set to (128, 128) and (80, 160) for HAAN-v1 and HAAN-v2, respectively.

Figure 8 (a) and Figure 9 compare processing latency and power consumption for normalization layers in GPT-2. HAAN-v1 and HAAN-v2 show an average latency reduction of $11.7\times$, $10.5\times$, $1.25\times$ and $2.42\times$ compared to DFX, GPU, SOLE and MHAA, respectively, across different input sequence lengths. Additionally, HAAN-v1 and HAAN-v2 achieve average power usage reductions of 61% and 64% compared to DFX while using slightly less power than SOLE and MHAA. HAAN's superior performance primarily stems from its ISD skipping and subsampling mechanism, leading to significant computational cost reduction. Additionally, HAAN optimizes the latency of each individual blocks by dynamically configuring the precision, enabling it to support deep pipelining for enhanced throughput. Also, HAAN utilizes inter-sample parallelism and pipelining and by setting particular p_d, p_n, the time of the different stages of the pipeline is evenly distributed, so that we can maximize the utilization rate of hardware units and lower latency when input size becomes huge.

Input Format	(p_d, p_n)	FPGA resource consumption			Power (W)
		LUT	FF	DSP	
FP32	(128, 128)	84K/4.9%	17K/0.5%	1536/12.5%	6.362
	(32, 128)	99K/5.7%	21K/0.6%	1036/8.4%	6.136
FP16	(128, 128)	55K/3.2%	11K/0.4%	1536/12.5%	4.868
	(32, 128)	76K/4.5%	15K/0.4%	1036/8.4%	4.790
INT8	(256, 256)	58K/3.4%	21K/0.6%	1536/12.5%	3.458
	(32, 512)	86K/5.0%	25K/0.7%	1025/8.3%	6.382

TABLE III: Hardware cost of HAAN accelerator with FP32/FP16/INT8 inputs. The left/right numbers represent absolute values and percentage of overall resources. Power was measured as the average at input lengths of 16, 128, and 256.

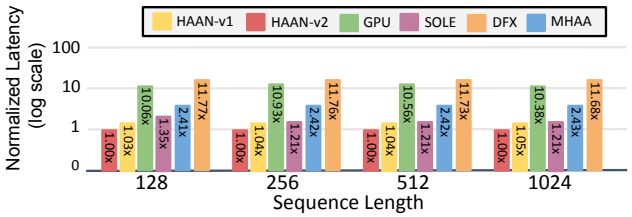

Fig. 9: Normalized latency of HAAN compared to DFX and GPU on GPT2-1.5B.

Figure 8 (b) illustrates the comparative analysis of processing latency of the normalization layers within OPT-2.7B, where 7 out of 65 ISD operations can be skipped and the input can be further truncated with a length of 1280. Here we introduce *HAAN-v3*, another HAAN configuration with a single pipeline, FP16 input and (p_d, p_n) set to (64,128). We also observe that HAAN-v1 and HAAN-v3 achieve a notable reduction in processing latency compared to the baseline hardware platforms.

To evaluate end-to-end performance, we use the experimental settings and results from previous work [41]. We test the GPT-2 model with 355M parameters and 24 layers, experimenting with input lengths of 128, 256, and 512. Our results show that incorporating HAAN will enable an average end-to-end speedup of approximately $1.11\times$ across different input lengths.

VI. Conclusion

This study introduces HAAN, a holistic normalization acceleration approach for LLMs. HAAN exploits the correlation in normalization statistics among adjacent layers to bypass normalization computation by estimating statistics from preceding layers. This evaluation results demonstrate that HAAN can achieve great improvement over other baselines algorithms.

REFERENCES

[1] H. Khan, A. Khan, Z. Khan, L. B. Huang, K. Wang, and L. He, "Npe: An fpga-based overlay processor for natural language processing," *arXiv preprint arXiv:2104.06535*, 2021.

[2] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 616–630.

[3] H. Wang *et al.*, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," 2021.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[5] B. Zhang and R. Sennrich, "Root mean square layer normalization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[6] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin, "Understanding and improving layer normalization," *Advances in neural information processing systems*, vol. 32, 2019.

[7] W. Wang, S. Zhou, W. Sun, P. Sun, and Y. Liu, "Sole: Hardware-software co-design of softmax and layernorm for efficient transformer inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[8] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "Nn-lut: neural approximation of non-linear operations for efficient transformer inference," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 577–582.

[9] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[10] Y. Gao *et al.*, "Design and implementation of an approximate softmax layer for deep neural networks," in *2020 ISCAS*, 2020, pp. 1–5.

[11] M. Wang *et al.*, "A high-speed and low-complexity architecture for softmax function in deep learning," in *2018 APCCAS*, 2018, pp. 223–226.

[12] J. R. Stevens *et al.*, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *2021 DAC*. IEEE Press, 2021, p. 469–474.

[13] Y. Zhang *et al.*, "Base-2 softmax function: Suitability for training and efficient hardware implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3605–3618, 2022.

[14] T. Xia and S. Q. Zhang, "Softmax acceleration with adaptive numeric format for both training and inference," *arXiv preprint arXiv:2311.13290*, 2023.

[15] A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. D. Sa, "Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks," 2024.

[16] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: Outlier-free 4-bit inference in rotated llms," 2024.

[17] E. Frantar and D. Alistarh, "Optimal brain compression: A framework for accurate post-training quantization and pruning," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=ksVGCOlOEba

[18] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," 2023.

[19] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," 2020.

[20] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," 2023.

[21] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[22] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[23] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[24] T. B. Brown *et al.*, "Language models are few-shot learners," 2020.

[25] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[26] S. Jeong, M. Seo, X. T. Nguyen, and H.-J. Lee, "A low-latency and lightweight fpga-based engine for softmax and layer normalization acceleration," in *2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, 2023, pp. 1–3.

[27] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," in *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*. IEEE, 2020, pp. 84–89.

[28] Z. Liu, G. Li, and J. Cheng, "Hardware acceleration of fully quantized bert for efficient natural language processing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 513–516.

[29] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille, "Normface: L2 hypersphere embedding for face verification," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1041–1049.

[30] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018, pp. 2704–2713.

[31] C. Lomont, "Fast inverse square root," *Tech-315 nical Report*, vol. 32, 2003.

[32] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[33] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[34] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[35] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[36] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "A framework for few-shot language model evaluation," 12 2023. [Online]. Available: https://zenodo.org/records/10256836

[37] S. Jeong, M. Seo, X. T. Nguyen, and H.-J. Lee, "A low-latency and lightweight fpga-based engine for softmax and layer normalization acceleration," in *2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, 2023, pp. 1–3.

[38] G. Tzanos, C. Kachris, and D. Soudris, "Hardware acceleration of transformer networks using fpgas," in *2022 Panhellenic Conference on Electronics Telecommunications (PACET)*, 2022, pp. 1–5.

[39] M. Huang, J. Luo, C. Ding, Z. Wei, S. Huang, and H. Yu, "An integer-only and group-vector systolic accelerator for efficiently mapping vision transformer on edge," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 12, pp. 5289–5301, 2023.

[40] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," 2020. [Online]. Available: https://arxiv.org/abs/2009.08605

[41] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of fpga-based spatial acceleration for large language model inference," *ACM Transactions on Reconfigurable Technology and Systems*, Apr. 2024. [Online]. Available: http://dx.doi.org/10.1145/3656177