

# LUT-APP: Dynamic-Precision LUT-based Approximation Unifying Non-Linear Operations in Transformers

Seokkyu Yoon, Namjoon Kim and Hyun Kim

Department of Electrical and Information Engineering and Research Center for Electrical and Information Technology  
Seoul National University of Science and Technology, Seoul 01811, Korea  
skyoon@seoultech.ac.kr, rlarla2626@seoultech.ac.kr, hyunkim@seoultech.ac.kr

**Abstract**—On-device transformer inference faces a growing bottleneck in which non-linear functions (*e.g.*, exponential (EXP), reciprocal, reciprocal square root, GeLU, and SiLU) contribute significantly to inference latency as matrix operations become highly optimized. Existing approximation methods either rely on operator-specific datapaths with poor hardware reusability or exhibit a suboptimal accuracy-resource balance with conventional look-up table (LUT)-based piecewise linear approximation (PWL) under stringent edge constraints. This work presents *LUT-APP*, a unified dynamic-precision LUT-based PWL approximation framework that reconciles accuracy and hardware efficiency across diverse non-linear operators. First, a dynamic fixed-point format (DFF) adaptively allocates bit-width based on input magnitude and parameter scaling to handle the wide dynamic range of EXP. Second, a genetic adaptive differential evolution (GADE) algorithm synthesizes non-uniform PWL segments to minimize approximation error for a given LUT budget. Third, hardware-efficient DFF processing units enable a unified INT8 multiply-add datapath, allowing a single reusable implementation across functions. Experimental results demonstrate that *LUT-APP* reduces approximation error by up to  $6.87\times$  versus state-of-the-art methods while preserving baseline accuracy in large language models and vision transformers without fine-tuning. Hardware synthesis with a 28nm technology shows  $4.19\times$  lower area and  $3.26\times$  lower power savings than existing LUT-based PWL approaches, validating *LUT-APP* as a practical, resource-constrained solution for on-device accelerators. We provide the *LUT-APP* implementation at <https://github.com/IDSL-SeoulTech/LUT-APP>

**Index Terms**—Dynamic precision format, non-linear function approximation, Transformer, look-up table, metaheuristics

## I. INTRODUCTION

Transformer architectures have achieved state-of-the-art (SOTA) performance across natural language processing and computer vision tasks [1]–[3]. Growing demand for enhanced data privacy and reduced inference latency has driven significant interest in on-device deployment [4]–[6]. However, hardware and power constraints in on-device environments

This research was partly supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2026-RS-2022-00156295) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation) and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-02304537, Development of Intelligent Home On-Device AI Matter Hub System Technology). (Corresponding author: Hyun Kim)

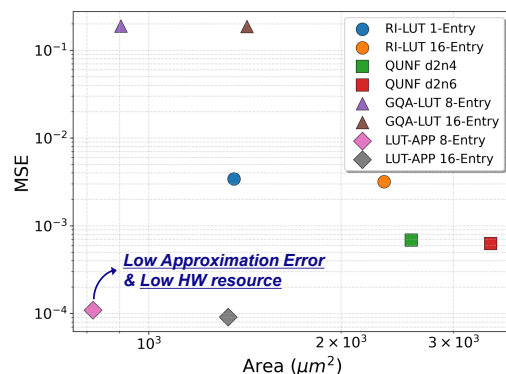


Fig. 1. Area-MSE trade-off comparison of existing PWL approximation methods. The x-axis and y-axis represent hardware area and MSE, respectively. MSE is averaged across EXP, RECI, RSQRT, GeLU, and SiLU.

make GPU-based inference impractical, necessitating specialized hardware for transformer inference [7]–[9]. Transformers comprise attention (ATTN) and feed-forward network (FFN), using matrix multiplications and non-linear functions (*e.g.*, exponential (EXP), reciprocal (RECI), reciprocal square root (RSQRT), GeLU, and SiLU). However, existing hardware research focuses on compression techniques like quantization [10]–[14] and pruning [15]–[18] for matrix multiplications, which dominate computational workload. As matrix multiplication efficiency continues to improve through these optimization techniques, non-linear functions increasingly emerge as a considerable performance bottleneck [19].

To address this issue, several approaches have been proposed to approximate non-linear functions. PEANO-ViT [20] leverages dedicated datapaths for layers with non-linear functions (*e.g.*, LayerNorm, Softmax) in vision transformers (ViTs) via log and *Padè*-based approximations. Alternatively, piecewise linear approximation (PWL) methods achieve better reusability through unified datapaths with look-up table (LUT)-based segment indexing. RI-LUT [21] reduces LUT size by scaling input ranges before applying PWL, while GQA-LUT [22] optimizes non-uniform PWL segments for INT8 quantization using a genetic algorithm. QUNF [23] minimizes error and delay with quadratic approximation on uniform PWL segments, avoiding multiplication. However, existing methods

fail to simultaneously achieve both hardware efficiency and low approximation error. Fig. 1 compares mean squared error (MSE) and hardware area for non-linear function approximations. RI-LUT [21] and QUNF [23] achieve low error but incur high hardware costs due to large computational modules and LUT overhead. Conversely, GQA-LUT [22] significantly reduces hardware overhead but increases error due to power-of-two (PoT) quantization and static input range limitations.

This challenge stems from a fundamental trade-off between hardware efficiency and approximation accuracy. On one hand, on-device environments impose strict resource constraints, requiring non-linear function modules to offer high module reusability, a minimal LUT size, and lightweight computational units. Indeed, inefficient implementations of non-linear functions can become an area bottleneck, consuming more resources than even the matrix multiplication units in on-device accelerators [24], [25]. On the other hand, ensuring low approximation error requires high-precision arithmetic and support for a wide input range. An excessive focus on hardware efficiency can degrade model accuracy, necessitating model fine-tuning to compensate for the performance, which is computationally expensive [26] and especially arduous for large language models (LLMs) due to their massive number of parameters [27]. This inherent trade-off severely constrains practical deployment scenarios, where existing approaches must sacrifice either accuracy or hardware efficiency.

To address these challenges, we propose *LUT-APP*, a unified dynamic precision LUT-based PWL approximation framework that simultaneously achieves high accuracy and hardware efficiency. First, a dynamic fixed-point format (DFF) adaptively adjusts integer bit-width based on data magnitude and parameter scaling, enabling precise approximation. Second, genetic adaptive differential evolution (GADE) optimizes PWL parameters, minimizing approximation error for a given LUT size. Third, hardware-efficient DFF converter and comparator units enable unified INT8 multiply-add (MADD) datapath for non-linear function approximation. Experimental evaluation demonstrates substantial improvements across multiple performance metrics. *LUT-APP* achieves up to  $6.87\times$  lower average approximation error compared to SOTA methods across non-linear functions common in transformer models. When integrated into LLMs and ViTs, our approach preserves baseline accuracy without requiring time-consuming fine-tuning. Hardware synthesis using 28nm technology achieves  $4.19\times$  area reduction and  $3.26\times$  power savings compared to existing LUT-based PWL approaches. These results demonstrate that *LUT-APP* successfully balances accuracy with hardware efficiency for on-device deployment. Our contributions are summarized as follows:

- We propose DFF to dynamically adjust integer bit-width based on data magnitude and parameter scaling, enabling precise approximation across a wide input range for EXP.
- We introduce GADE to optimize PWL segments, minimizing approximation error for a given LUT size.
- We propose a DFF converter and DFF comparator to enable a unified INT8 MADD datapath, reducing computational module size for non-linear function approximation.

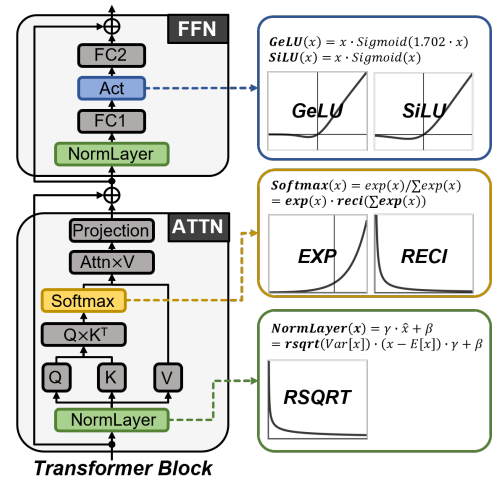


Fig. 2. Structure of the transformer block in a transformer-based model and types of non-linear functions with their equations.

## II. PRELIMINARIES AND RELATED WORK

### A. Non-Linear Functions in Transformer-based Models

Transformer models employ diverse non-linear functions across different computational blocks, as illustrated in Fig. 2. FFN utilizes activation functions such as GeLU and SiLU for complex feature learning, while attention mechanisms rely on EXP and RECI within softmax. Additionally, normalization layers, including LayerNorm and RMSNorm, require RSQRT operations for numerical stability. The diversity and complexity of these functions present significant challenges for hardware implementation. Each function exhibits distinct computational characteristics, input ranges, and precision requirements, making unified approximation difficult. This heterogeneity has motivated operator-specific design, such as PEANO-ViT [20], which implements dedicated approximation datapaths for individual operations to meet per-function requirements. However, dedicated datapaths sacrifice hardware efficiency for precision. In contrast, on-device environments demand unified architectures that maximize reusability while minimizing area and power. A unified approximation framework offers significant advantages in hardware efficiency, particularly for intensive functions like GeLU and SiLU.

### B. LUT-based Approximation

LUT-based methods are widely studied for hardware acceleration of non-linear functions due to their high modularity and efficient datapath integration [21]–[23]. PWL effectively reduces hardware complexity by dividing non-linear functions into segments and approximating them into linear operations. This approach enables a uniform datapath for diverse non-linear functions, significantly improving hardware efficiency. The LUT-based PWL is typically implemented as follows:

$$PWL(x) = \begin{cases} k_0 \cdot x + b_0, & x \in (R_n, p_0], \\ k_1 \cdot x + b_1, & x \in (p_0, p_1], \\ \vdots & \\ k_{N-1} \cdot x + b_{N-1}, & x \in (p_{N-2}, R_p], \end{cases} \quad (1)$$

where  $p$  represents segment breakpoints,  $R_n$  and  $R_p$  define input boundaries, and the parameters  $k$  and  $b$  represent the slopes and intercepts, respectively. LUT is used to store PWL parameters. This approach presents a fundamental trade-off between approximation accuracy and hardware cost. Finer segmentation with more breakpoints improves the accuracy but increases LUT size. To address this trade-off, existing methods employ either uniform or non-uniform segment strategies. Among existing methods, uniform segment approaches (RI-LUT [21], QUNF [23]) require a substantial LUT size or utilize high-precision arithmetic units to achieve acceptable accuracy. In contrast, the non-uniform approach (GQA-LUT [22]) has succeeded in effectively searching for segments, but it suffers from significant performance degradation due to PoT quantization, necessitating fine-tuning to recover performance.

### C. Metaheuristic Optimizations

Existing PWL methods rely on uniform segmentation or simple heuristics. Conventional optimization techniques [28], [29] struggle with PWL segment optimization due to the high-dimensional continuous search space and inter-dependencies between adjacent breakpoints. Metaheuristics [30] address these challenges by employing population-based search strategies that explore the search space more effectively. For PWL segment optimization, population-based metaheuristics adapt to a function's global and local characteristics. By evaluating multiple breakpoints and applying mutation, these methods assign wider segments to regions with low change rates and denser segments to areas with rapid changes, creating non-uniform PWL segments.

Representative population-based metaheuristics include genetic algorithm (GA) [31] and differential evolution (DE) [32], which are updated through operators such as mutation, crossover, and selection. GA excels in local convergence but risks local optima traps due to the initial population distribution. In contrast, DE, less sensitive to initial distribution, is strong in global exploration but can still fall into local optima with poor hyper-parameter tuning. Thus, an algorithm capable of consistently deriving effective solutions for PWL is required.

## III. PIECE-WISE LINEAR APPROXIMATION WITH DYNAMIC FIXED-POINT FORMAT

### A. Dynamic Fixed-point Format

Fig. 3 shows the wasted bits when using fixed-point format (FiP) for EXP inputs. Consequently, a unified PWL datapath with FiP inherently wastes bits, sacrificing precision for high hardware efficiency. Conversely, floating-point format (FP) minimizes wasted bits through dynamic scaling, achieving high precision. However, FP operations demand significant hardware resources for complex tasks like exponent handling and normalization. To tackle these challenges, we propose DFF, blending FP's adaptive precision with FiP's hardware simplicity. As depicted in Fig. 4, a DFF value,  $X_{DFF}$ , comprises an 8-bit value,  $V_X$ , and a 3-bit scale,  $S_X$ , expressed as follows:

$$X_{DFF} = V_X^{frac} \cdot 2^{S_X}, \quad (2)$$

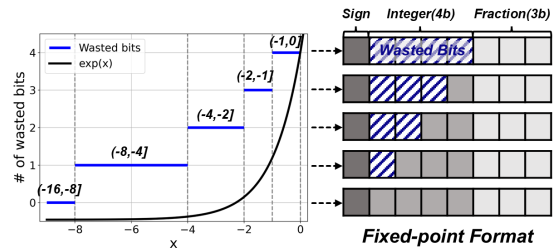


Fig. 3. Wasted bits in a fixed-point format that uses a 4-bit integer width to represent the EXP over the input range of  $[-9, 0]$ .

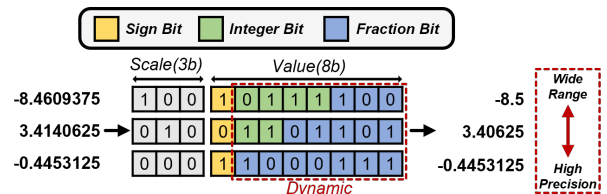


Fig. 4. Dynamic fixed-point format overview

$V_X^{frac}$  is the  $V_X$  expressed as  $Q1.7$ , referring to the case when the scale is 0. DFF dynamically adjusts integer and fractional bit-widths using  $V_X$  and  $S_X$  based on data magnitude. For large values, it allocates more bits to the integer to expand the range. For small values near zero, it prioritizes fractional bits to enhance precision. This approach suits non-linear functions, which change rapidly near zero and more gradually at extreme values. Nevertheless, when approximating EXP with PWL using DFF, PWL parameters in certain input ranges (e.g.,  $[-9, -5]$ ) become too small for reliable representation with DFF. This causes significant performance degradation in models like LLaMA2 [33] that require a wide EXP input range.

To mitigate this problem, we propose parameter scaling that approximates a function scaled by  $2^k$  in segments that have parameters difficult to represent with DFF. The segments where scaling is applied are determined based on the point at which the DFF representation of the EXP's derivative becomes zero. Accordingly, the EXP used for the approximation can be defined as follows:

$$EXP(x) = \begin{cases} e^x, & x \in (-5.5625, 0] \\ 2^k \cdot e^x, & x \in (-9, -5.5625] \end{cases} \quad (3)$$

The original EXP can then be approximated by subtracting  $k$  from the DFF scale corresponding to the PWL output. This approach extends the input range with minimal hardware cost and enhances EXP approximation accuracy. Moreover, it is also applicable to other non-linear functions (e.g., RECI, RSQRT) that require high precision for PWL parameters.

### B. Genetic Adaptive Differential Evolution for Piece-Wise Linear Approximation

Population-based metaheuristics effectively optimize PWL segments by balancing LUT size and approximation error. However, existing population-based methods typically excel in either local convergence or global exploration, necessitating hyperparameter tuning tailored to the specific problem. To address these challenges, we propose GADE, a PWL segment

### Algorithm 1: Genetic Adaptive Differential Evolution Piece-Wise Linear Approximation

**Input:** Function  $f(\cdot)$ , search range  $(R_n, R_p)$ , breakpoint size  $N_b$ , population size  $N_p$ , mutant probability  $\theta_m$ , iteration size  $T$ , strategy probability range  $(SP_{min}, SP_{max})$

**Output:** Sets of slopes  $\mathcal{K}$ , intercepts  $\mathcal{B}$  and breakpoints  $\mathcal{P}$

- 1: Initialize breakpoint population  $\mathcal{O} = \{\mathcal{P}_0, \dots, \mathcal{P}_{N_p-1}\}$ ,
- 2: each  $\mathcal{P}_i \in [0, N_p-1]$  is a set of  $N_b$  random FP32 values in  $[R_n, R_p]$
- 3: Initialize success set of scaling factor  $\mathcal{F}$  and crossover rate  $\mathcal{CR}$
- 4: **for**  $n = 1$  **to**  $T$  **do**
- 5:     **for**  $\mathcal{P}_i$  in  $\mathcal{O}_n$  **do**
- 6:         /\* Determine  $SP$  based on progress \*/
- 7:          $SP = SP_{min} + \frac{1}{2}(SP_{max} - SP_{min})(1 + \cos(\frac{n}{T}\pi))$
- 8:         /\* Adaptive Hyperparameter Determination \*/
- 9:          $\mathcal{F} \leftarrow$  Random in *Cauchy*(mean( $\mathcal{F}_{success}$ ),  $SP$ )
- 10:          $\mathcal{CR} \leftarrow$  Random in *Gaussian*(mean( $\mathcal{CR}_{success}$ ),  $SP$ )
- 11:         /\* Dual Mutation Operation using DE and GA \*/
- 12:         Initialize mutant breakpoint population  $\mathcal{P}_m$
- 13:          $rand_m \leftarrow$  Random in  $[0,1]$
- 14:         **if**  $rand_m < SP$  **then**
- 15:             Randomly select two donor  $\mathcal{P}_{r1}, \mathcal{P}_{r2}$
- 16:              $\mathcal{P}_m = \mathcal{P}_{best} + \mathcal{F}(\mathcal{P}_{r1} - \mathcal{P}_{r2})$
- 17:         **else**
- 18:              $\mathcal{P}_m = \mathcal{P}_i +$  Random in *Gaussian*(0,  $\theta_m$ )
- 19:         **end**
- 20:         /\* Crossover Operation \*/
- 21:         Copy  $\mathcal{P}_i$  to initialize trial breakpoint population  $\mathcal{P}_t$
- 22:         **for**  $j = 0$  **to**  $N_b - 1$  **do**
- 23:              $rand_{CR} \leftarrow$  Random in  $[0,1]$
- 24:             **if**  $rand_{CR} < \mathcal{CR}$  **then**
- 25:                  $\mathcal{P}_t[j] = \mathcal{P}_m[j]$
- 26:             **end**
- 27:         **end**
- 28:         /\* Selection Operation \*/
- 29:         Initialize Mean Absolute Error (MAE):  $E_i = 0$
- 30:         Create PWL( $\cdot$ ) based on  $\mathcal{P}_t$  using DFF
- 31:         **for**  $x = R_n$  **to**  $R_p$  **with step**  $0.01$  **do**
- 32:             Update  $E_i = E_i + \frac{|PWL(x) - f(x)|}{(R_p - R_n)/0.01}$
- 33:         **end**
- 34:         **if**  $E_i < E_{i-1}$  **then**
- 35:              $\mathcal{O}_{n+1}[i] = \mathcal{P}_t$
- 36:             Append  $\mathcal{F}, \mathcal{CR}$  to  $\mathcal{F}_{success}, \mathcal{CR}_{success}$
- 37:         **else**
- 38:              $\mathcal{O}_{n+1}[i] = \mathcal{P}_i$
- 39:         **end**
- 40:     **end**
- 41: **end**
- 42:  $\mathcal{P}^* \leftarrow$  Select the best breakpoint set from  $\mathcal{O}_T$
- 43:  $\mathcal{K}, \mathcal{B} \leftarrow$  Derived from  $\mathcal{P}^*$  using DFF
- 44:  $\mathcal{P} = \frac{\lfloor \mathcal{P}^* \cdot 2^4 \rfloor}{2^4}$

TABLE I  
CONFIGURATIONS OF GADE

Hyper-parameters	EXP	RECI	RSQRT	GeLU	SiLU
$(R_n, R_p)$	(-9,0)	(1,2)	(1,4)	(-4,4)	(-4,4)
$N_b$			7 / 15		
$N_p, \theta_m, T, C_P$			50, 0.2, 500, 0.6		
$(SP_{min}, SP_{max})$			(0.2, 0.6)		

are generated based on the current generation's breakpoints and converted to DFF. The approximation error is measured, and if a lower error is achieved compared to the previous generation, it is used in the next generation. Additionally, hyperparameters that reduce the error are added to the success set (Algorithm 1, Lines 29-39).

#### IV. HARDWARE DESIGN OF LUT-APP

Fig. 5 shows the module architecture for LUT-APP. Although this module uses FP for its input and output, it internally converts it to the DFF to leverage a hardware-efficient INT8 MADD datapath. The module contains four primary components: a ① DFF Converter, a ② DFF Comparator, a ③ DFF multiplier, and a ④ DFF adder.

① **DFF Converter** transforms an FP input into DFF. The FP input comprises a sign  $Sign$ , exponent  $Exp$ , and mantissa  $Manti$ . The conversion parameters are determined as follows:

$$\begin{aligned} UnExp &= Exp - (bias - 1), \\ Manti_S &= 1.Manti \gg -\min(UnExp, 0), \end{aligned} \quad (4)$$

where the unbiased exponent  $UnExp$  is computed using a bias of  $bias - 1$ , ensuring the shifted mantissa  $Manti_S$  lies within  $[0, 1]$ . If  $UnExp$  is negative,  $Manti$  that includes implicit 1 is right-shifted to obtain  $Manti_S$ . The values  $S_q$  and  $V_q$  are then determined as follows:

$$\begin{aligned} S_q &= \min(\max(UnExp, 0), 7) \\ V_q &= \begin{cases} Saturate, & UnExp > 7, \\ 2's\ comp\{Sign, Manti_S\} & UnExp \leq 7, \end{cases} \end{aligned} \quad (5)$$

If  $UnExp > 7$ ,  $S_q$  is saturated to 7, the maximum for 3 bits, and  $V_q$  is set to -128 or 127 based on  $Sign$ . If  $UnExp \leq 7$ ,  $S_q$  equals  $UnExp$ .  $V_q$  is formed in 8-bit 2's complement format, combining  $Sign$  with the upper 7 bits of  $Manti_S$ .

② **DFF Comparator** determines the index of the PWL segment by comparing the converted DFF data with the FiP breakpoints. The DFF data is converted to FiP as follows:

$$q_{FiP} = V_q \ll S_q \quad (6)$$

Subsequently,  $q_{FiP}$  is compared in parallel with all 8-bit FiP breakpoints. The comparator outputs are concatenated and fed into a priority encoder, which selects the final index by identifying the first true result, starting from the highest index  $N_b - 1$ .

③, ④ **DFF Multiplier & DFF Adder** processes multiplication and addition using the  $V$  and  $S$  components of DFF, enabling the INT8 MADD datapath for PWL. The DFF multiplier computes the product of the input  $q$  and the PWL slope  $k$  as follows:

$$V_m = V_q \cdot V_k, \quad S_m = S_q + S_k, \quad (7)$$

optimization algorithm based on adaptive cauchy differential evolution [34], which dynamically optimizes global exploration and local convergence while adaptively tuning hyperparameters based on algorithmic progress. Details of the algorithm are provided in Algorithm 1, with configurations listed in Table I.

GADE employs cosine scheduling [35] to adjust the strategy probability  $SP$ , promoting global exploration early and local convergence later based on algorithmic progress (Algorithm 1, Line 7). For adaptive hyperparameter tuning, hyperparameters  $\mathcal{F}$  and  $\mathcal{CR}$  are determined by obtaining random numbers from *Cauchy* and *Gaussian* distributions, respectively. The success set means of  $\mathcal{F}$  and  $\mathcal{CR}$  are used as the mean and  $SP$  as the standard deviation (Algorithm 1, Lines 9-10). In the mutation operation, DE-based mutation utilizing differences between populations and GA-based mutation using noise are employed in parallel, with proportions determined by  $SP$  (Algorithm 1, Lines 12-19). Subsequently, the current generation's breakpoints are probabilistically replaced with the mutation (Algorithm 1, Lines 21-27). Finally, PWL parameters

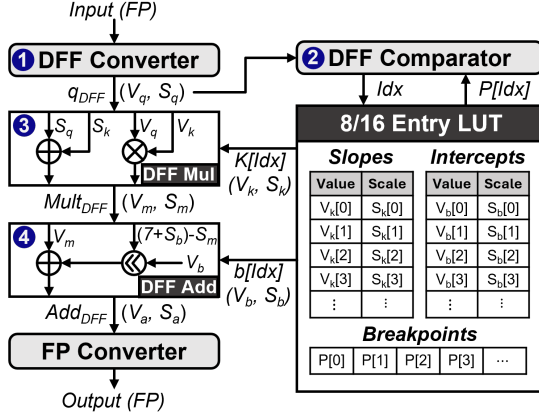


TABLE II  
MSE COMPARISON OF LUT-BASED METHODS

Method	Setting	EXP (-9,0)	RECI (0.01,128)	RSQRT (0.01,128)	GeLU (-6,6)	SiLU (-6,6)	Avg.
RI-LUT [21]	1 Entry	1.67e-5	9.53e-4	1.91e-4	2.55e-3	1.34e-2	3.42e-3
	16 Entry	5.56e-9	7.49e-8	2.23e-8	2.54e-3	1.33e-2	3.17e-3
QUNF [23]	d=2, n=4	1.82e-5	9.23e-5	2.08e-6	1.41e-3	1.90e-3	6.85e-4
	d=2, n=6	1.19e-6	1.45e-5	1.55e-7	1.31e-3	1.80e-3	6.25e-4
GQA-LUT [22]	8 Entry	2.24e-3	8.05e-1	4.24e-2	5.02e-2	4.41e-2	1.89e-1
	16 Entry	2.11e-3	8.05e-1	4.25e-2	4.07e-2	4.47e-2	1.87e-1
<b>Ours</b>	8 Entry	1.35e-5	7.94e-5	9.94e-7	2.90e-4	1.62e-4	<b>1.09e-4</b>
	16 Entry	3.55e-6	7.11e-5	9.42e-7	2.80e-4	9.12e-5	<b>9.09e-5</b>

Fig. 5. Architecture of *LUT-APP* module. The logic for extra transformations required by some non-linear functions is not included.

where an 8-bit multiplier computes the product of  $V_q$  and  $V_k$ , while a 3-bit adder sums  $S_q$  and  $S_k$ . The DFF multiplication yields a 16-bit value,  $V_m$ , and a 4-bit scale,  $S_m$ . Subsequently, the DFF adder then adds the input,  $V_m$ , to the PWL intercept,  $b$ , using 16-bit adder as follows:

$$X_{Add} = V_m + (V_b \ll ((7 + S_b) - S_m)), \quad (8)$$

where  $V_b$  is shifted based on the difference between  $S_m$  and  $S_b$  to align for consistent results. The DFF addition produces a 17-bit value,  $V_a$ , and a 4-bit scale,  $S_a (= S_{mult})$ . The DFF adder's outputs,  $V_a$  and  $S_a$ , are fed into the FP converter and transformed into FP.

## V. EXPERIMENTAL RESULTS

### A. Experimental Setup

**Software Setup** We evaluated three representative LLMs—LLaMA2-7B [33], Falcon-7B [36], and Mistral-7B [37]—on perplexity (PPL) using wikitext2 (WIKI) datasets and average accuracy using five common-sense reasoning datasets: Winograde (WG), PIQA (PQ), Hellaswag (HS), Arc-easy (A-e), and Arc-challenge (A-c). For ViT comparison, we used DeiT-Small/Base [38], ViT-Large [39], and Swin-Base [40], assessing top-1 accuracy on ImageNet-1k dataset. All experiments were conducted in PyTorch, replacing non-linear functions with non-linear function approximations without fine-tuning to isolate their impact. Since RI-LUT [21] lacks SiLU support, we adapt its GeLU implementation by removing the sigmoid scaling constant. For RECI and RSQRT functions with wide input ranges, we apply the input scaling technique from RI-LUT [21].

**Hardware Setup** We implemented *LUT-APP* module in Verilog, synthesizing all modules with Synopsys Design Compiler (2023.12) targeting a Samsung 28nm technology (500MHz, 0.9V). For fair comparison, all approximation modules from prior works were re-implemented and synthesized under identical conditions. Hardware efficiency was assessed using three metrics: area ( $\mu\text{m}^2$ ), power (mW), and LUT size (Kb).

### B. Non-linear Function Approximation Accuracy

Table II compares approximation errors across five non-linear functions common in transformer-based models: EXP, RECI, RSQRT, GeLU, and SiLU. MSEs are computed by sampling inputs at  $2^{-10}$  intervals across specified function ranges. PEANO-ViT [20] was excluded due to its limited support for non-linear functions. Existing methods exhibit distinct performance characteristics and limitations. RI-LUT [21] handles wide input ranges for GeLU and SiLU but overlooks function-specific properties, resulting in high MSE despite increased LUT entries. QUNF [23] uses segments corresponding to the input range multiplied by  $2^d$ , and as  $n$  increases, the number of partial sums grows, enabling more precise approximations. Through this, it achieves low MSE for most functions; however, due to the removal of some parameters necessary for computation to achieve a low parameter size, it exhibits low accuracy for high-curvature functions such as GeLU and SiLU. GQA-LUT [22] suffers from high MSE due to both approximation and PoT quantization errors. The precision limit of INT8 MADD datapath hinders performance, with little error reduction despite more LUT entries. In contrast, *LUT-APP* attains consistently lower MSE across all functions, including GeLU and SiLU, owing to DFF's high precision and GADE-optimized non-uniform PWL parameters. Parameter scaling together with DFF enables *LUT-APP* to markedly reduce approximation error with more entries. To the best of our knowledge, *LUT-APP* achieves up to  $6.87\times$  lower average MSE than QUNF [23], the current SOTA.

### C. Accuracy Evaluation on Transformer-based Models

1) *Large Language Model Evaluation:* Table III evaluates the impact of the approximation methods on inference performance of LLMs, measured by PPL and average accuracy. RI-LUT [21] significantly degrades performance across all models. Even in its best-case scenario, it causes an average accuracy drop of 3.67% and a PPL increase of 0.830. This degradation is due to high approximation errors for GeLU and SiLU, as shown in Table II. Conversely, QUNF [23] maintains a performance close to the baseline due to low approximation error. In its best cases, it perfectly matches the baseline PPL and even improves the average accuracy by 0.15%. However, a minimal performance drop is still observed in other scenarios, with the accuracy decreasing by as little as 0.08%. This slight instability is attributed to its limited input range for the EXP. *LUT-APP* achieves the best performance, outperforming prior methods while demonstrating consistent performance preservation across

TABLE III  
INFERENCE PERFORMANCE OF LARGE LANGUAGE MODELS

Model	Method	Setting	WIKI PPL ↓	Avg. Acc. ↑ (%)
LLaMA2-7B	Baseline	-	5.477	69.00
	RI-LUT [21]	1/16 Entry	6.628/6.307	64.49/65.32
	QUNF [23]	d=2, n=4/6	5.500/5.490	68.91/68.83
	<b>Ours</b>	8/16 Entry	<b>5.481/5.477</b>	<b>68.98/68.99</b>
Falcon-7B	Baseline	-	6.631	67.76
	RI-LUT [21]	1/16 Entry	9.042/8.331	63.09/64.09
	QUNF [23]	d=2, n=4/6	6.639/6.631	67.67/67.68
	<b>Ours</b>	8/16 Entry	<b>6.695/6.630</b>	<b>67.97/67.74</b>
Mistral-7B	Baseline	-	5.154	74.08
	RI-LUT [21]	1/16 Entry	6.391/6.062	68.95/70.04
	QUNF [23]	d=2, n=4/6	5.177/5.170	73.93/74.23
	<b>Ours</b>	8/16 Entry	<b>5.162/5.160</b>	<b>74.19/74.30</b>

all models, with its worst-case accuracy drop being a mere 0.01%. The combination of DFF adaptive precision allocation, GADE-optimized PWL segments, and parameter scaling for extended dynamic range effectively addresses the limitations observed in prior methods. Notably, *LUT-APP* occasionally exceeds baseline performance across both metrics, reducing the PPL by up to 0.001 and increasing the average accuracy by at least 0.11%. We attribute these gains to the regularization effect inherent in the approximation.

2) *Vision Transformer Evaluation*: Table IV shows the top-1 accuracy of ViTs after applying each non-linear function approximation method. PEANO-ViT [20] shows performance degradation due to its hardware-friendly coarse approximations, exhibiting an accuracy drop of at least 0.04% even in its best-case scenario. GQA-LUT [22] suffers a catastrophic performance loss across all models, with accuracy collapsing to a near-zero 0.11% in the worst case. Even in its best-case scenario, the accuracy still plummeted by over 24%. This severe degradation is primarily due to the limited input range support for RECI and RSQRT, as well as PoT quantization errors. In contrast, *LUT-APP* achieves the best performance, decisively outperforming prior methods. Its superiority is evident as it robustly preserves baseline accuracy, with a worst-case drop of a mere 0.01%, and even improves upon it by at least 0.02%. This is driven by the synergy of proposed methods: DFF adaptive precision allocation, GADE-optimized PWL segments, and parameter scaling for extended dynamic range.

#### D. Hardware Implementation Results

Table V compares synthesized hardware costs for unified PWL approximation modules under identical design conditions. PEANO-ViT [20] is excluded from comparison as it employs function-specific datapaths. The hardware efficiency analysis reveals distinct trade-offs across existing approaches. RI-LUT [21] achieves minimal LUT utilization (0.096-2.256 Kb) through input range scaling, but suffers from significant area overhead (1,360-2,337  $\mu\text{m}^2$ ) due to FP arithmetic units. QUNF [23] eliminates multiplication hardware through quadratic approximation yet incurs the highest LUT size (5.802 Kb) for storing quadratic coefficients, limiting its scalability for resource-constrained accelerators. GQA-LUT [22] reduces computational complexity through the INT8 MADD datapath but still requires significant memory capacity for breakpoint

TABLE IV  
INFERENCE PERFORMANCE OF VISION TRANSFORMER MODELS

Model	Method	Setting	ImageNet-1k Top-1 Acc. ↑ (%)
DeiT-Small	Baseline	-	79.83
	PEANO-ViT [20]	-	79.13
	GQA-LUT [22]	8/16 Entry	55.75/55.41
	<b>Ours</b>	8/16 Entry	<b>79.79/79.82</b>
DeiT-Base	Baseline	-	81.80
	PEANO-ViT [20]	-	81.65
	GQA-LUT [22]	8/16 Entry	2.09/1.91
	<b>Ours</b>	8/16 Entry	<b>81.83/81.83</b>
ViT-Large	Baseline	-	85.84
	PEANO-ViT [20]	-	84.83
	GQA-LUT [22]	8/16 Entry	51.22/52.69
	<b>Ours</b>	8/16 Entry	<b>85.83/85.82</b>
Swin-Base	Baseline	-	83.60
	PEANO-ViT [20]	-	83.56
	GQA-LUT [22]	8/16 Entry	0.11/0.11
	<b>Ours</b>	8/16 Entry	<b>83.58/83.62</b>

TABLE V  
PERFORMANCE COMPARISON OF LUT-BASED PWL APPROXIMATION

Method	Setting	Area( $\mu\text{m}^2$ )	Power( $mW$ )	LUT size(Kb)
RI-LUT [21]	1 Entry	1,360	0.48	0.096
	16 Entry	2,337	0.84	2.256
QUNF [23]	d=2, n=4	2,579	0.65	5.802
	d=2, n=6	3,433	0.88	5.802
GQA-LUT [22]	8 Entry	904	0.29	1.928
	16 Entry	1,425	0.52	4.040
<b>Ours</b>	8 Entry	<b>818</b>	<b>0.27</b>	<b>1.160</b>
	16 Entry	<b>1,332</b>	<b>0.58</b>	<b>2.360</b>

storage to store all breakpoints for each scale factor. In contrast, our proposed *LUT-APP* module demonstrates superior hardware efficiency across all metrics. The proposed approach demonstrates 4.19 $\times$  area reduction and 3.26 $\times$  power savings compared to existing methods while maintaining competitive memory requirements. This efficiency stems from GADE-optimized segment that reduces LUT size and DFF processing units that enable lightweight operations. The combination allows *LUT-APP* to deliver the best overall area-power-memory balance for practical on-device deployment.

## VI. CONCLUSION

This paper presented *LUT-APP*, a unified dynamic-precision approximation framework for accelerating non-linear operations in on-device transformer inference. The proposed approach addresses the fundamental trade-off between approximation accuracy and hardware efficiency. First, DFF enables adaptive bit-width allocation based on input magnitude. Second, GADE optimizes PWL segment placement within fixed LUT constraints. Third, hardware-efficient DFF processing units provide a unified INT8 datapath for all non-linear functions. Experimental evaluation demonstrates 6.87 $\times$  lower approximation error compared to SOTA. When deployed in various ViTs and LLMs, our method results in negligible performance degradation without requiring fine-tuning. Hardware synthesis in 28nm CMOS achieves 4.19 $\times$  area reduction and 3.26 $\times$  power savings than prior works. These results establish *LUT-APP* as a practical solution for deploying transformers on resource-constrained edge devices, enabling efficient inference across mobile and embedded platforms.

## REFERENCES

- [1] S. I. Lee, K. Koo, J. H. Lee, G. Lee, S. Jeong, S. O, and H. Kim, "Vision transformer models for mobile/edge devices: a survey," *Multimedia Systems*, vol. 30, no. 2, p. 109, 2024.
- [2] B. Min, H. Ross, E. Sulem, A. P. B. Veysseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–40, 2023.
- [3] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI open*, vol. 3, pp. 111–132, 2022.
- [4] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering edge intelligence: A comprehensive survey on on-device ai models," *ACM Comput. Surv.*, vol. 57, no. 9, Apr. 2025. [Online]. Available: <https://doi.org/10.1145/3724420>
- [5] V. Shankar, "Edge ai: a comprehensive survey of technologies, applications, and challenges," in *2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET)*. IEEE, 2024, pp. 1–6.
- [6] A. Singh, S. C. Satapathy, A. Roy, and A. Gutub, "Ai-based mobile edge computing for iot: Applications, challenges, and future scope," *Arabian Journal for Science and Engineering*, vol. 47, no. 8, pp. 9801–9831, 2022.
- [7] M. Huang, A. Shen, K. Li, H. Peng, B. Li, Y. Su, and H. Yu, "Edgellm: A highly efficient cpu-fpga heterogeneous edge accelerator for large language models," *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1–14, 2025.
- [8] J. Jang, Y. Kim, J. Lee, and J.-J. Kim, "Figna: Integer unit-based accelerator design for fp-int gemm preserving numerical accuracy," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 760–773.
- [9] S. Moon, J.-H. Kim, J. Kim, S. Hong, J. Cha, M. Kim, S. Lim, G. Choi, D. Seo, J. Kim *et al.*, "Lpu: A latency-optimized and highly scalable processor for large language model inference," *IEEE Micro*, 2024.
- [10] B. J. Kang, N. Kim, and H. Kim, "Lra-qvit: Integrating low-rank approximation and quantization for robust and efficient vision transformers," in *Forty-second International Conference on Machine Learning*.
- [11] S. Jeong, S. Lee, and H. Kim, "Lowgradq: Adaptive gradient quantization for low-bit cnn training via kernel density estimation-guided thresholding and hardware-efficient stochastic rounding unit," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–2.
- [12] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 092–28 103, 2021.
- [13] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of machine learning and systems*, vol. 6, pp. 87–100, 2024.
- [14] D. Choi and H. Kim, "Gradq-vit: Robust and efficient gradient quantization for vision transformers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 15, 2025, pp. 16 019–16 027.
- [15] Z. Song, Y. Xu, Z. He, L. Jiang, N. Jing, and X. Liang, "Cp-vit: Cascade vision transformer pruning via progressive sparsity prediction," *arXiv preprint arXiv:2203.04570*, 2022.
- [16] X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," *Advances in neural information processing systems*, vol. 36, pp. 21 702–21 720, 2023.
- [17] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *International conference on machine learning*. PMLR, 2023, pp. 10 323–10 337.
- [18] N. J. Kim and H. Kim, "Trunk pruning: Highly compatible channel pruning for convolutional neural networks without fine-tuning," *IEEE Transactions on Multimedia*, vol. 26, pp. 5588–5599, 2023.
- [19] W. Wang, W. Sun, and Y. Liu, "Improving transformer inference through optimized non-linear operations with quantization-approximation-based strategy," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.
- [20] M. E. Sadeghi, A. Fayyazi, S. Azizi, and M. Pedram, "Peano-vit: Power-efficient approximations of non-linearities in vision transformers," in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, 2024, pp. 1–6.
- [21] J. Kim, J. Lee, J. Choi, J. Han, and S. Lee, "Range-invariant approximation of non-linear operations for efficient bert fine-tuning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.
- [22] P. Dong, Y. Tan, D. Zhang, T. Ni, X. Liu, Y. Liu, P. Luo, L. Liang, S.-Y. Liu, X. Huang *et al.*, "Genetic quantization-aware approximation for non-linear operations in transformers," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [23] H. Du, C. Wen, Z. Chen, L. Zhang, Q. Sun, Z. Yan, and C. Zhuo, "Algorithm-hardware co-design of a unified accelerator for non-linear functions in transformers," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
- [24] S. Tuli and N. K. Jha, "Acceltran: A sparsity-aware accelerator for dynamic inference with transformers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4038–4051, 2023.
- [25] J. Wang, L. Zhang, X. Li, H. Yang, and Y. Liu, "Ulseq-ta: Ultra-long sequence attention fusion transformer accelerator supporting grouped sparse softmax and dual-path sparse layernorm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 3, pp. 892–905, 2023.
- [26] Z. Li and Q. Gu, "I-vit: Integer-only quantization for efficient vision transformer inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 065–17 075.
- [27] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021. [Online]. Available: <https://arxiv.org/abs/2106.09685>
- [28] O. Shamir and T. Zhang, "Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes," in *International conference on machine learning*. PMLR, 2013, pp. 71–79.
- [29] J. M. Hinson and J. Staddon, "Matching, maximizing, and hill-climbing," *Journal of the experimental analysis of behavior*, vol. 40, no. 3, pp. 321–331, 1983.
- [30] S. Ólafsson, "Metaheuristics," *Handbooks in operations research and management science*, vol. 13, pp. 633–654, 2006.
- [31] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, vol. 80, no. 5, pp. 8091–8126, 2021.
- [32] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on evolutionary computation*, vol. 15, no. 1, pp. 4–31, 2010.
- [33] 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.
- [34] T. J. Choi, C. W. Ahn, and J. An, "An adaptive cauchy differential evolution algorithm for global numerical optimization," *The Scientific World Journal*, vol. 2013, no. 1, p. 969734, 2013.
- [35] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.
- [36] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, É. Goffinet, D. Hesslow, J. Launay, Q. Malartic *et al.*, "The falcon series of open language models," *arXiv preprint arXiv:2311.16867*, 2023.
- [37] A. Q. Jiang *et al.*, "Mistral 7b," 2023. [Online]. Available: <https://arxiv.org/abs/2310.06825>
- [38] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.
- [39] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
- [40] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.