Three Birds, One Stone: Fast, Accurate-aware and Cost-Efficient Accelerator for Ternary LLM

Wonseok Jung
Korea University
Seoul, Republic of Korea
bgiant6097@gmail.com

Junseok Kang
Korea University
Seoul, Republic of Korea
jun311k@korea.ac.kr

Sangwon Shin
Korea University
Seoul, Republic of Korea
husask11@korea.ac.kr

Hongjun Um
Hanyang University
Seoul, Republic of Korea
hongjunum@hanyang.ac.kr

Jangho Lim
Korea University
Seoul, Republic of Korea
imjh98@korea.ac.kr

Yongjun Park
Yonsei University
Seoul, Republic of Korea
yongjunpark@yonsei.ac.kr

Gunjae Koo
Korea University
Seoul, Republic of Korea
gunjaekoo@korea.ac.kr

Sangwoo Park
Seoul Women's University
Seoul, Republic of Korea
psw0113@swu.ac.kr

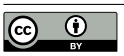
Taeweon Suh
Korea University
Seoul, Republic of Korea
suhtw@korea.ac.kr

Abstract

On-device LLM inference is increasingly important for latency- and privacy-sensitive applications, yet it remains challenging due to the high compute and storage demands. Ternary-weight LLMs are a promising direction because they dramatically reduce model size and simplify arithmetic. In practice, deploying pretrained models on edge devices typically relies on post-training quantization (PTQ), but ternary PTQ often needs fine-grained scaling to preserve accuracy, which amplifies scale-metadata traffic and sub-byte decoding overhead that fits poorly with conventional NPU datapaths. This paper presents *T-ACE*, a Ternary Accuracy-aware Compute Engine that enables efficient ternary LLM inference under PTQ by jointly designing the data representation and execution pipeline. T-ACE co-packs 64 ternary weights and power-of-two scale metadata into a naturally aligned 16-byte block, eliminating separate scale fetches and preserving aligned memory access. To decode compact ternary packing efficiently, T-ACE proposes a compact two-stage 5-trit unpacker and integrates on-the-fly decoding and scaling directly into the ternary GEMM pipeline. The evaluation on an FPGA prototype shows that decoding and scaling are fully overlapped with GEMM execution, incurring no additional cycles over baseline. Moreover, the comparison against A100/H100 baselines in a normalized setting shows that T-ACE improves accuracy-adjusted compute density (ACD) by 66.8% and accuracy-adjusted energy efficiency (AEE) by 17.6% over the best GPU baseline.

CCS Concepts

• **Hardware → Application specific processors**.



ICS '26, Belfast, United Kingdom

ACM ISBN 979-8-4007-2522-7/26/07
https://doi.org/10.1145/3797905.3807877

Keywords

Ternary LLM, Post-training quantization, Hardware accelerator, GEMM, RISC-V, Edge AI

ACM Reference Format:
Wonseok Jung, Junseok Kang, Sangwon Shin, Hongjun Um, Jangho Lim, Yongjun Park, Gunjae Koo, Sangwoo Park, and Taeweon Suh. 2026. Three Birds, One Stone: Fast, Accurate-aware and Cost-Efficient Accelerator for Ternary LLM. In *2026 International Conference on Supercomputing (ICS '26), July 06–09, 2026, Belfast, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3797905.3807877

1 Introduction

Ternary-weight neural networks have recently attracted growing attention as a promising direction for efficient large language model (LLM) inference. Unlike higher-bit quantization schemes, ternary weights enable multiplication-free computation, where expensive multipliers are replaced by simple sign selection and accumulation. Despite their extreme compression ratio, ternary models often retain higher accuracy than other ultra-low-bit formats at the same bit budget[21, 23]. These properties make ternary LLMs particularly attractive for energy- and area-constrained inference accelerators.

Post-training quantization (PTQ) enables low-cost deployment of large language models by directly quantizing pretrained weights, avoiding expensive retraining or large-scale fine-tuning. While quantization-aware training can further improve accuracy, its cost and limited applicability motivate PTQ-centric designs in practice. Recent PTQ-based designs, including PT-BitNet [15], and PTQTP [43], demonstrate that carefully designed scaling and calibration schemes are essential for preserving accuracy under aggressive low-bit quantization. In this PTQ setting, the scaling scheme becomes a central factor in balancing accuracy, storage overhead, and inference efficiency, especially for ternary LLMs. Block Data Representations (BDR) with Microexponent (MX) [8] shows that two-level scaling can substantially improve accuracy with narrow weight bits.

Nevertheless, the scaling scheme requires a large number of bits for scales, thereby amortizing the benefit of using narrower weight

Table 1: Representative MX data formats [8] and scale overhead (block size k_1=16, sub-block size k_2=2).

	MX9	MX6	MX4
Data bits per element	8-bit	5-bit	3-bit
Mantissa	7	4	2
Sign	1	1	1
Scale bits per block	16-bit	16-bit	16-bit
First-level scale	8	8	8
Second-level scale	1-bit x 8	1-bit x 8	1-bit x 8
Scale overhead	12.5%	20.0%	33.3%

bit-widths to reduce memory pressure. Table 1 shows representative MX data formats and scale overhead. The MX format organizes weights into blocks of size k_1. Each block shares the first level scale. A block is further partitioned into sub-blocks of size k_2. Each sub-block shares the second level scale. The scale overhead is defined as the ratio of scale bits to weight bits per block, as shown in Eq. (1).

$$scale\ overhead = \frac{\#bits\ for\ scales}{\#bits\ for\ weights} = \frac{s_1 + \left(\frac{k_1}{k_2}\right) s_2}{k_1 w} \quad (1)$$

where s_1 denotes the number of first level scale bits, s_2 is the number of second level scale bits per sub-block, and w is the number of bits per weight. Each block in the MX format contains one s_1 and k_1/k_2 s_2. For $k_1 = 16$, $k_2 = 2$, s_1=8-bit, and s_2=1-bit in Table 1, MX9 demands 16-bit (= 8-bit + 8 x 1-bit) for scales for each block, while the number of bits for weights is 128-bit (= 16 x 8-bit). Thus, the scale overhead is 12.5% for the MX9 format.

The scale overhead grows quickly as data precision for weight decreases from MX9 to MX4 as shown in Table 1. It is because the amount of required bits for scales remains constant per block, while the amount of bits for weights decreases with lower precision. In addition to the storage overhead relative to the packed low-bit weights, separately stored scale metadata can also incur runtime overhead since the scale metadata should be loaded from memory, consuming memory bandwidth. Moreover, the corresponding scaling operation must be performed during inference.

This paper proposes a hardware-friendly ternary format that co-compresses weights and scales into a unified representation. By compressing ternary weight values and allocating the remaining bit space to scale information, scale values are colocated with weights. This design removes the need for explicit scale loading and multiplication, while preserving sufficient expressive power for accurate inference. Building on this representation, we design a *ternary accuracy-aware compute engine (T-ACE)* that efficiently decodes and applies the embedded scales with minimal hardware overhead. As a result, the proposed approach enables accurate, fast, and cost-efficient ternary LLM inference under a PTQ setting.

Our contributions are as follows:

- We propose a **16-byte packing format** that co-packs 64 ternary weights with two-level, power-of-two scale metadata. It dramatically reduces memory traffic for loading weights and scale metadata (Fig. 7).
- We propose and design a compact and novel decoder architecture: **two-stage 5-trit unpacker**. It takes negligible hardware area (0.0076 mm^2 and 2.8 mW at 28 nm in Table 7), incurring no additional GEMM execution cycles due to full overlap with the compute pipeline.
- We design T-ACE, a **ternary-GEMM engine** specialized for ternary LLM inference with **fine-grained scaling** support. T-ACE achieves **66.8%** higher efficiency per area and **17.6%** higher efficiency per watt than the GPU baseline under the normalized setting (Table 4).

Table 2: Performance comparison according to ternary quantization formats in llama.cpp (Llama-3.2 1B) [24]

Quantization	Model Size	bpw (bits/weight)	pp512* (tokens/sec)	tg128* (tokens/sec)
TQ1_0	401.5 MiB	1.69	31.36	20.08
TQ2_0	445.0 MiB	2.06	60.01	26.85
I2_S	1.21 GiB	2.00	58.30	20.90

*pp512: prefill throughput with a 512-token prompt.
*tg128: decode throughput for generating 128 tokens.

2 Background and Motivation

2.1 Ternary Weight Packing and Decode Overheads

A ternary weight is represented with an element in the set $\{-1, 0, +1\}$ and thus takes $\log_2 3 \approx 1.58$ bits per weight. When represented in a conventional binary format, however, they are typically stored using at least 2 bits per weight. This gap between the information-theoretic minimum and the practical storage format has motivated a line of work on ternary-specific packing schemes. In these schemes, multiple trits are compacted into a single byte or word to reduce both model size and weight-access bandwidth.

A common strategy is to exploit the fact that $3^k < 2^n$ for small k, allowing k trits to be encoded into an n-bit container with some unused code space. For instance, since $3^5 (= 243) < 2^8$, five ternary weights can be stored in a single byte, achieving sub-2-bit storage efficiency [17, 45]. While these encodings are attractive from a compression standpoint, they often misalign with the underlying memory system. A packed representation may not match the fetch granularity (byte, word, or block), requiring additional bit manipulation during decoding. This effect is reflected in end-to-end performance across packing formats.

Table 2 summarizes performance according to representative ternary quantization formats used in practice: `TQ1_0` and `TQ2_0` from `llama.cpp` [13], and `I2_S` from `bitnet.cpp` [26]. All measurement data in Table 2 were obtained on the WSL environment with a 16-core based Intel® Core i9-10900X (3.70 GHz). `TQ1_0` and `TQ2_0` in Table 2 both use block-wise scaling, sharing one FP16 scale per 256 weights, which slightly increases the effective bits per weight (bpw). `TQ1_0` reduces storage using a hybrid packing scheme: a small subset of weights is stored explicitly in 2-bit form, while the majority is compressed using a 5-trit-per-byte packing method (i.e., five ternary values packed into one byte), achieving 1.69 bpw. `TQ2_0` follows an aligned 2-bit (INT2) layout per weight with per-block

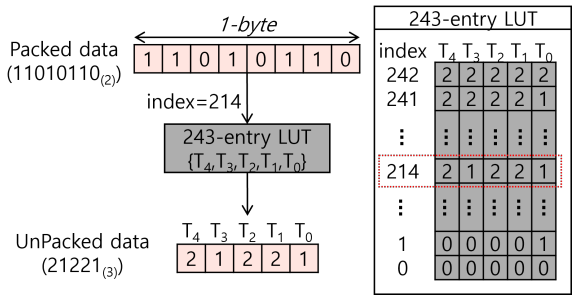


Figure 1: Naive LUT-based 5-Trit Unpacker

scale, leading to 2.06 bpw. The experiment reveals a trade-off between weight compression format and execution efficiency: `TQ1_0` achieves the smallest footprint (1.69 bpw) but suffers lower prefill and decode throughput. `TQ2_0` and `I2_S` adopting aligned 2-bit layouts increase model size, yet deliver higher throughput. Table 2 hints that an effective ternary packing scheme — rather than focusing only on compression efficiency — is imperative for balancing memory access and decoding overhead [39].

Prior research on ternary accelerators such as TENET [17] and Tereffic [45] typically adopts a lookup-table (LUT)-based unpacker to map each packed code back to several ternary values at once. As illustrated in Fig. 1, a naïve 5-trit unpacker may enumerate all 243 ($= 3^5$) valid ternary patterns and store the corresponding decoded symbols in a table. Assuming a 2-bit internal representation for each ternary value, a single such LUT requires $5 \times 243 \times 2$ bits of storage. To match the throughput of the GEMM operation, packed ternary weights must be unpacked at the same rate as they are consumed. This requires multiple unpack operations to proceed in parallel, increasing the hardware cost of the unpacker. As a result, the naïve LUT-based unpacking introduces non-trivial area and energy overhead that scales with the width of the GEMM datapath. To address this issue, we propose a cost-efficient two-stage unpacking structure where the 5-trit decoder is composed of a small comparator-based front end and a small residual lookup. This factorized design provides 5-trits/cycle decode throughput with significant area reduction, continuously feeding the GEMM pipeline.

2.2 Inflexible Architectures for PTQ Scales

Recent ternary-oriented accelerators such as TeLLMe[35], TENET[17], and TerEffic[45] demonstrate that ternary arithmetic can be efficiently exploited at the microarchitectural level; MAC operations were performed with ternary-specific primitives, such as sign selection, accumulation, or small lookup tables, thereby reducing compute complexity and energy cost [17, 35, 45]. A common characteristic of these accelerators is that they target models trained from scratch with ternary weights, where the quantization scheme and scaling strategy are fixed during training. Accordingly, scaling factors are typically applied at a predefined and coarse granularity—most commonly per tensor or per channel—and are tightly integrated into the execution flow of the accelerator. This tight coupling between model format and hardware datapath enables efficient ternary computation, but it limits applicability for finer-grained or adaptive scaling schemes required by PTQ or fine-tuned ternary models.

Applying PTQ scales in prior accelerators naturally requires partitioning execution into smaller blocks, each of which uses a single scale factor. This partitioning leads to repeated loading of scale values and weight blocks, and the application of scale to the corresponding block. The resulting overhead manifests as redundant scale operations in hardware and degraded utilization of the compute pipeline. Although prior architectures effectively leverage the simplicity of ternary arithmetic for high throughput, they are fundamentally limited in scenarios that demand PTQ scales.

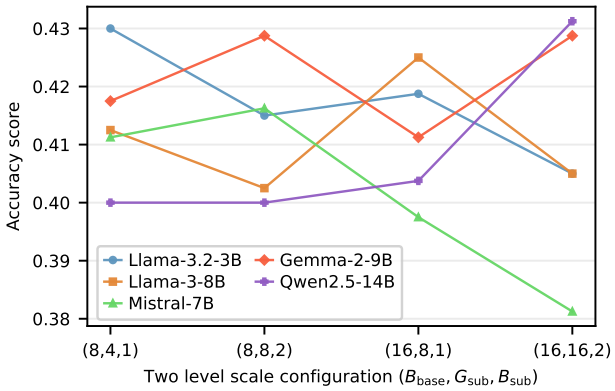
Table 3: Average accuracy scores according to scaling group sizes under ternary quantization. (Scores are averaged over HellaSwag [46], WinoGrande [36], WiC [34] and ANLI-R2 [30])

Model	Per16	Per32	Per64	Per128
Llama-3.2-3B [25]	0.405	0.4163	0.4088	0.395
Llama-3-8B-Instruct [14]	0.4175	0.4000	0.4263	0.4088
Mistral-7B-Instruct-v0.3 [18]	0.3963	0.4063	0.4463	0.430
gemma-2-9b-it [12]	0.4388	0.4313	0.4313	0.4375
Qwen2.5-14B-Instruct [44]	0.4288	0.4075	0.4100	0.390
Average	0.4173	0.4123	**0.4245**	0.4123

2.3 Scale Granularity in Ternary models

Ternary quantization enables extreme reduction in model size, but maintaining accuracy remains a challenge. Unlike higher-bit formats, ternary weights provide very limited representational capacity, making model performance highly reliant on scaling. A single shared scale per tensor, where one scale is used for the entire weight matrix of a layer, is often insufficient for large language models. The weight distributions vary significantly across layers, channels, and local regions. In particular, a small number of extreme values, referred to as outliers, can dominate the dynamic range of an entire tensor. When a coarse-grained scale is determined by such outliers, most weights are mapped to a narrow range around zero under ternary quantization. As a result, the effective representational capacity of ternary levels is severely reduced. Similar behavior has also been observed in ternary LLMs obtained via quantization-aware training of pretrained models, where per-tensor scaling can underperform finer-grained schemes such as per-channel or per-group scaling [16].

Finer-grained scaling (e.g., per-channel or per-group) mitigates this issue by applying a scale to small local weights, improving ternary accuracy. However, this benefit is not free: additional scales increase metadata storage and memory traffic, and require extra arithmetic operations during inference. Thus, it is imperative to find a scaling granularity that satisfies accuracy with tolerable overhead. To quantify this trade-off, we have performed evaluation over scaling group sizes using the MX quantization emulator [27], which applies ternary quantization and measures LLM performance by executing benchmarks. Table 3 shows the average scores of LLM models, where scores are averaged over HellaSwag, WinoGrande,



B_{base}: base exponent bits
G_{sub}: subgroup size
B_{sub}: subgroup offset bits

Figure 2: Accuracy scores according to two-level scalings

WiC, and ANLI-R2. The accuracy score denotes exact-match accuracy, measured as the percentage of correctly answered samples over the full evaluation set. Per16 in Table 3, for example, means that a scale is applied to a group of 16 weights. Although individual models exhibit small variations in score, Per64 shows the highest score, meaning that applying a scale to a group of 64 weights provides the best LLM performance. Thus, a group size of 64 is used as the outer granularity for subsequent analysis.

Based on MX-style two-level scaling [8], we adopt two-level scaling to cope with the accuracy loss of coarse group scaling in ternary PTQ. To determine the detailed configuration, we have examined two-level scaling configurations. In Fig. 2, $(B_{\text{base}}, G_{\text{sub}}, B_{\text{sub}})$ denotes the bit-width of the base scale exponent, the subgroup size, and the bit-width of the subgroup exponent offset, respectively. The group size is fixed to 64. For Llama-3.2-3B, the best performance was achieved with the $(8, 4, 1)$ configuration. The configuration that achieves the best score differs depending on LLM models, meaning that a configuration that is optimal for one model is not necessarily optimal for other models. Therefore, the hardware support for the flexible scaling configuration is needed to adapt variations.

3 T-ACE Design

Ternary Accuracy-aware Compute Engine (*T-ACE*) is a cost-effective ternary LLM accelerator with memory-access-friendly packed weights and scales. Fig. 4 shows the T-ACE microarchitecture and its main operations. On-chip scratchpad memory is organized as four banks. It stores INT8-based activations and packed 16-byte ternary weight tiles via Direct Memory Access (DMA). During execution, a single activation row is loaded into an activation buffer and broadcast to all compute units in parallel. Each compute unit incorporates a 256×64 PE array.

3.1 Co-Packed Weight–Scale Representation

Transporting weights and scale metadata separately incurs significant overhead due to frequent scale loads, complicating address generation. T-ACE eliminates these costs by co-packing ternary weights and their scale metadata into a single, naturally aligned 16-byte representation. As shown in Fig. 3, each 16-byte block encodes 64 ternary weights and their group-level scaling information.

Each ternary weight has three possible values. Five ternary weights can be encoded in one byte (B) with Eq. (2) where $d_t \in \{0, 1, 2\}$ ($t = 0$ for the least significant trit):

$$B = \sum_{t=0}^{4} d_t \, 3^t, \qquad B \in \{0, \ldots, 242\} \subset [0, 255]. \tag{2}$$

Values $243 \sim 255$ in B are unused. Then, 64 ternary weights can be stored in 13-byte; 60 ternary weights are stored in 12-byte (= 5-trit/byte x 12), and the remaining 4 ternary weights are stored without encoding in the last 1-byte (= 4-trit x 2-bit). The PE does not necessitate a separate decoding logic for the last 4-trits, reducing the hardware cost.

In a 16-byte block, 3 B is used to store scale metadata. Depending on the selected scale mode, these bits are partitioned into a base scale and a set of subgroup scales. To keep inference lightweight, T-ACE represents scaling factors in a power-of-two form [8, 40]: the metadata stores a base exponent S and small subgroup exponent offsets s, and the corresponding scale factors are reconstructed as 2^S and 2^{-s}, respectively.

Let $t_k \in \{-1,\ 0,\ 1\}$ denote the decoded ternary value for weight element k. If element k belongs to subgroup G_s, its effective scale is

$$\alpha_s = 2^{S-s}. \tag{3}$$

Accordingly, the reconstructed weight is given by

$$w_k \approx \alpha_s \cdot t_k, \ k \in G_s. \tag{4}$$

The number of subgroups and the bit-width of each scale are configurable at runtime, allowing different scale granularities to be supported under the same block format. We support four representative scale configurations, parameterized by the bit-width of the base scale exponent (B_{base}), the subgroup size (G_{sub}), and the bit-width of each subgroup exponent offset (B_{sub}), as summarized in Eq. (5).

$$(B_{\text{base}},\ G_{\text{sub}},\ B_{\text{sub}}) \in \left\{ \begin{matrix} (16, 16, 2),\ (16, 8, 1), \\ (8, 4, 1),\ (8, 8, 2) \end{matrix} \right\}. \tag{5}$$

This compact 16-byte format is fully decoded by the Decoder described in Section 3.2, and the resulting weights and scales are consumed by the execution fabric in Section 3.3 .

3.2 Cost-Efficient 5-Trit Unpacking

A naïve and simple 5-trit unpacker would require 243 (=3^5)-entry Lookup Table (LUT). As shown in Fig. 1, requiring numerous tables replicated across lanes. T-ACE employs a two-stage unpacker that recovers five ternary weights from one packed byte with low area and fixed latency. The first stage extracts two high-order trits, and the second stage decodes the remaining three trits.

Let P be the packed byte. T-ACE compares P against eight thresholds $\{\theta_0, \theta_1, \ldots, \theta_7\} = \{27, 54, 81, 108, 135, 162, 189, 216\}$ to generate a bitvector $b_j = \mathbf{1}$ if $P \geq \theta_j$ for $j = 0, \ldots, 7$. The elements in the bitvector are added together to compute $q = \text{popcount}(b_0, \ldots, b_7) \in$

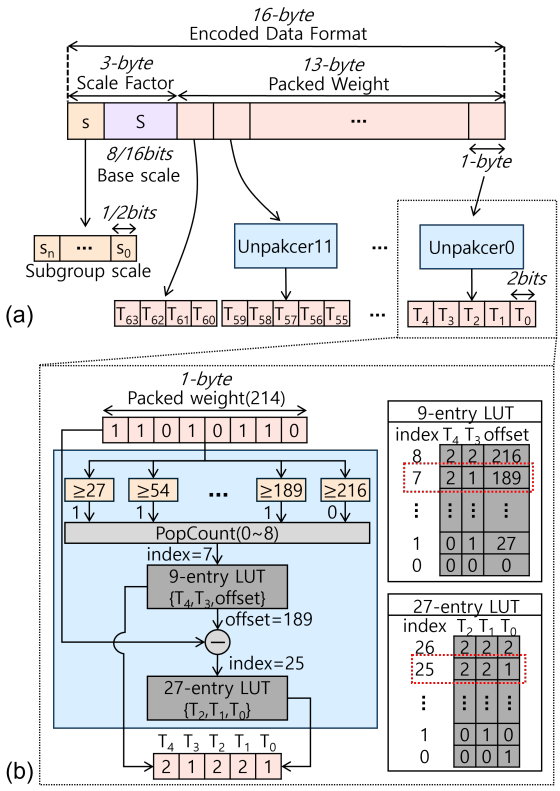


Figure 3: (a) Encoded 16-byte data format, and (b) unpacking process illustrated with a packed value of 214.

$\{0, \ldots, 8\}$. q is used to index a 9-entry LUT, which contains the two high-order trits (t_4, t_3) and their base contribution

$$L[q] = 81\,t_4 + 27\,t_3. \tag{6}$$

Stage 2 uses the remainder from Stage 1.

$$R = P - L[q], \qquad R \in [0, 26]. \tag{7}$$

The remainder R from Eq. (7) is used to index a small $27(3^3)$-entry LUT, which contains the three low-order trits (t_2, t_1, t_0). This two-stage structure replaces the large 243-entry table with a comparator–popcount front end and a small LUT, yielding a short combinational path.

Fig. 3(b) illustrates the detailed two-stage behavior with a packed byte example (214). T-ACE replicates twelve 5-trit unpackers; one for each packed byte inside the 16-byte packed block. These units collectively reconstruct 60 ternary weights. The remaining four weights, and the scale metadata are read directly from the same 16-byte block.

A naïve 5-trit table stores $3^5 = 243$ entries, each returning five trits at 2 bits per trit, for $\approx$ 2,430 bits per instance. In contrast, the factored unpacker requires the following small tables:

(i) Low-order trits LUT (t_2, t_1, t_0): 27 entries × 3 trits × 2 bits = 162 bits.
(ii) High-order trits LUT (t_4, t_3): 9 entries × 2 trits × 2 bits = 36 bits.
(iii) Weighted-base LUT $\big(L[q] = 81t_4 + 27t_3\big)$: 9 entries × 8 bits = 72 bits.

The total lookup storage is 162+36+72 = 270 bits. It is approximately 9× reduction per instance. Stage 1 comprises nine parallel 8-bit comparators and a popcount, forming a shallow combinational path. Combined with the aligned 16-byte co-packed block, the decoder maintains constant per-block throughput, decoding 64 ternary weights and four subgroup scales per 16-byte into the execution units.

3.3 Execution Unit Microarchitecture

3.3.1 Pipeline Imbalance in Ternary GEMM. Fig. 5 illustrates the fundamental challenge of executing ternary matrix multiplication under asymmetric operand bit-widths. In Fig. 5(a), where both weights and activations use INT8, weight load, activation load, and computation proceed at comparable rates, resulting in a well-balanced pipeline with high utilization. In contrast, Fig. 5(b) shows the case where weights are stored in INT2 while activations remain INT8. For the same memory interface width, the INT2 weight stream can be delivered up to 4× faster than the INT8 activation stream. As a result, the weight-load stage completes early and stalls while waiting for activation data. Because computation cannot proceed without activations, processing elements remain idle during these gaps, and the overall pipeline becomes activation-bound. This imbalance implies that the higher density of ternary weights does not automatically translate into proportional throughput gains unless the dataflow and on-chip buffering are designed to match the activation bandwidth.

3.3.2 T-ACE Flow. As illustrated in Fig. 4, (1) during the decode and preload stage, a single 64-byte weight line (4×16-byte) is expanded into 256 ternary *weights*, four base scales, and a set of subgroup scales per fetch. The 256 decoded weights are written directly into four consecutive PE arrays (64 PEs per row), maintaining row alignment across the array.

In Fig. 4(2) the broadcast stage, activations enter through an *Activation Buffer*, where each activation row is broadcast to every row in the PE array. Then, in (3) the compute and scale stage, the preloaded ternary weights and their subgroup scales interact with the incoming activation via sign-select and shift operations, while the base scale bypasses the inner loop and is applied once after adder-tree reduction.

For each data transfer from memory, the weight stream (2-bit based) carries 4× more elements than the activation stream (8-bit based). To balance the memory access with computations, T-ACE chose to use the weight-stationary dataflow and a 256×64 PE array, where weights are preloaded in the PE array. T-ACE uses a simple *weight-rearrangement* scheme to effectively transfer data from memory to scratchpad: As shown in Fig. 6, vertically adjacent four weight rows (that is, 4 rows x 16-byte; 64 elements in 16-byte) are stored in consecutive locations in memory to form a 64-byte memory block. Given one DMA transfer accesses a 64-byte memory block, it allows fetching the required amount of data for activations and weights.

For execution, one activation row (64-byte) in activation buffer is broadcast to all PE rows. Then the dot product of 64 pairs occurs in each row of the PE array. During the broadcasting and computation

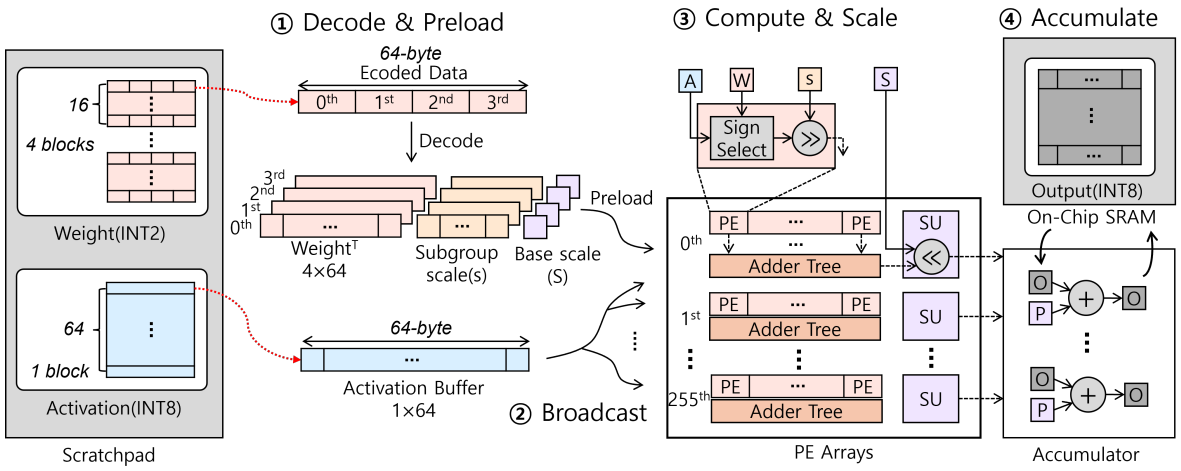


Figure 4: T-ACE microarchitecture and its main operations. (1): Decoders parse 64-byte data to preload weights and scales into PEs over 64 times. (2): Activations broadcast to all PE arrays. (3): PEs compute ternary operations with shift sub-scale correction. Results are summed by adder trees, multiplied by base scales.

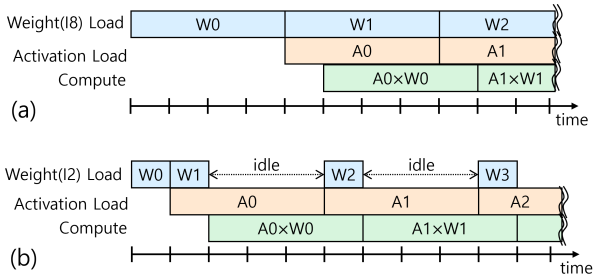


Figure 5: Pipeline comparison between (a) INT8×INT8 and (b) INT2×INT8.

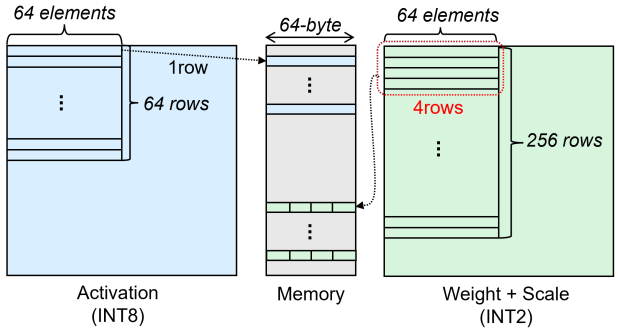


Figure 6: Weight rearrangement for efficient data access via DMA

for 64 activation rows (that is, 4096-byte = 64 rows x 64-byte/row), the next 256×64 weights (that is, 4096-byte = 256×16-byte) from scratchpad are fetched, decoded and stored in double buffers inside the PE array. This coarse-grain pipelining keeps the broadcast, dot product, and scaling phases fully occupied. In other words, the data access is perfectly balanced with computations. Each subgroup is associated with a *subgroup scale* s, encoded with 1–2 bits, which represents a small exponent offset value relative to the base scale. This subgroup scale is applied as a left shift of the partial products. All subgroups share a common base scale S, represented as an exponent with 8–16 bits. Thus, the scaled dot product can be written as:

$$\sum_{s} \sum_{k \in G_s} x_k \, t_k \, 2^{-s} \, 2^{S} \;=\; 2^{S} \cdot \sum_{s} \sum_{k \in G_s} x_k \, t_k \, 2^{-s} \qquad (8)$$

Each row in the PE array performs a 64-element dot product between an INT8 activation vector x and a ternary weight vector t. The 64 weights are divided into multiple subgroups G_s, where the number of subgroups ranges from 4 to 16 depending on the selected scale mode.

As shown in Eq. (8), the subgroup scale s is applied to each partial product, and the multiplication is implemented in hardware simply with shifters. The base scale S is applied only once after the accumulation of the 64 partial sums. This separation between the small per-PE sub-scale shifters and the shared wide base-scale shifters keeps the inner datapath small. Applying the full shift in each PE would require 64×256 16-bit shifters, whereas T-ACE uses 64×256 2-bit shifters and only 256 16-bit shifters, as shown in Fig. 4 (3). Furthermore, wide shifts inside the PE would enlarge the partial sums, increasing the cost of the adder tree and interconnect. By moving the wide scaling outside the inner GEMM datapath, T-ACE reduces area and power while preserving correct power-of-two scaling.

4 T-ACE Evaluation

This section evaluates T-ACE in terms of accuracy, computational efficiency, and hardware cost.

Table 4: Comparison of T-ACE and GPUs in terms of compute density, energy efficiency, and accuracy-adjusted metrics according to weight quantization schemes

HW	Weight Format	Scale Granularity	Compute Density [28] (TOPS/mm^2)	Energy Efficiency [28] (TOPS/W)	A_{rel} (vs. FP16)	ACD (A_{rel}TOPS/mm^2)	AEE (A_{rel}TOPS/W)
A100 Tensor Core	FP16	–	8.87	9.97	1.00	8.87	9.97
A100 Tensor Core	INT8	per-tensor	17.73	19.94	0.8294	14.71	16.54
A100 Tensor Core	INT8	group (64)	17.73	19.94	0.9964	17.67	19.87
H100 Tensor Core	FP8 (E4M3)	per-tensor	12.59	12.24	0.8516	10.72	10.42
H100 Tensor Core	FP8 (E4M3)	group (64)	12.59	12.24	0.9968	12.55	12.20
T-ACE (This work)	**Ternary**	**group (64)**	**43.14**	**34.20**	**0.6830**	**29.47**	**23.36**

· Due to the lack of publicly available area and power information for A100 [32] and H100 [33] Tensor Cores manufactured in 7 nm and 4 nm processes, all GPU results are normalized to a 28 nm process and a 1.41 GHz operating frequency for fair comparison. [28]
· T-ACE was evaluated under the same normalization assumptions (i.e., 28 nm process and 1.41 GHz operating frequency)

Table 5: Comparison of T-ACE and TENET in terms of supported quantization, compute density, and energy efficiency

HW	Weight Format	Quant. Support	Compute Density (TOPS/mm^2)	Energy Efficiency (TOPS/W)
TENET STL Core [17]	Ternary	QAT	12.80	195.05
T-ACE (This work)	**Ternary**	**PTQ**	**30.60**	**34.09**

4.1 T-ACE Efficiency

To evaluate T-ACE in terms of accuracy, compute density and energy efficiency, we have executed four benchmarks (HellaSwag, WinoGrande, WiC, and ANLI-R2) using MX emulator [27] with five LLMs: meta-llama/Llama-3.2-3B [25], meta-llama/Meta-Llama-3-8B-Instruct [14], mistralai/Mistral-7B-Instruct-v0.3 [18], google/gemma-2-9b-it [12], and Qwen/Qwen2.5-14B-Instruct [44].

The LLM accuracy is typically degraded with lower-bit quantization due to increased quantization error. To report how much the LLM quality is preserved with quantization, we introduce a metric referred to as *Relative Accuracy* (A_{rel}), as defined in Eq. (9). It is the ratio of the accuracy score of a quantized model to that of the FP16-based model under the same evaluation condition.

$$A_{\text{rel}} = \frac{\text{Accuracy score(quant)}}{\text{Accuracy score(FP16)}} \tag{9}$$

Extreme low-bit quantization schemes would deliver high efficiency in terms of TOPS/mm^2 and/or TOPS/W. However, those metrics could mislead the evaluation AI accelerators' efficiency, especially in ternary accelerators due to the accuracy drop. To incorporate accuracy into the efficiency evaluation, two new metrics are defined as shown in Eq. (10): *Accuracy-adjusted Compute Density (ACD)* and *Accuracy-adjusted Energy Efficiency (AEE)*. These metrics take accuracy into account in TOPS/mm^2 and TOPS/W.

$$\text{ACD} = A_{\text{rel}} \cdot \frac{\text{TOPS}}{\text{mm}^2}, \qquad \text{AEE} = A_{\text{rel}} \cdot \frac{\text{TOPS}}{\text{W}} \tag{10}$$

Table 4 shows the evaluation outcomes, comparing T-ACE against the GPU baselines in terms of accuracy-adjusted metrics. The comparison was performed under a unified normalized condition as described in the note below Table 4. A_{rel} is obtained by averaging the normalized scores over all model–dataset pairs, which are 20 results from five models evaluated with four benchmarks. Among the GPUs, A100 (INT8) boasts the highest ACD (17.67) and AEE (19.87) with group-64 scaling. T-ACE achieves superior performance compared to the GPUs in ACD (29.47) and AEE (23.36), which correspond to a 66.8% improvement in ACD and a 17.6% improvement in AEE over the best GPU baseline. In terms of raw hardware capability, T-ACE also provides substantially higher compute density and energy efficiency, delivering 43.14 TOPS/mm^2 and 34.20 TOPS/W.

We further compare T-ACE with TENET [17], a representative ternary accelerator for QAT-based models, as shown in Table 5. TENET focuses on QAT-based ternary execution, whereas T-ACE targets PTQ-based ternary LLMs with fine-grained group-wise scaling. Within these different design points, T-ACE achieves higher compute density (30.60 vs. 12.80 TOPS/mm^2), while TENET reports higher energy efficiency.

4.2 Memory Traffic Efficiency of T-ACE

This section evaluates the benefit of using the T-ACE's packing structure in terms of the number of incurred memory traffic. Reducing memory traffic is critical for the inference performance. Memory traffic is quantified by counting the number of 64-B memory block requests required to stream *activations*, *weights*, and *scale metadata* for representative GEMMs. 64-B is the DMA granularity in Chipyard SoC [1]. In matrix multiplication ($O_{M\times N} = A_{M\times K} \times W_{K\times N}$), M is the number of rows in the activation matrix (which varies depending on batch or tokens processed at once), K is the reduction dimension (dot-product length), and N is the output width. We fix K=N=3,200 as in BitNet-b1.58 3B.

Activations are represented in INT8 format. Thus each row in the activation matrix takes 3,200 bytes, corresponding to 50 (= 3,200 / 64) block requests. Then loading activations requires $50 \times M$ requests. For weights, T-ACE stores 64 ternary weights and their scale metadata together in one 16-B block. Thus the full $K\times N$ weight-scale tile requires 2.56 MB (= $3{,}200 \times 3{,}200$ / $64 \times 16B$) memory. Loading weights and scales requires 2.56 MB / 64 B = 40,000 block requests, independent of M.

Fig. 7 shows the number of memory block requests of T-ACE and two other typical schemes. *Case1* and *Case2* assume that weights and scales are stored in separate arrays. In both cases, weights are represented in INT2 (2-bit) format. When it comes to scales, in Case1, 3-B scales required for 64-weight blocks are stored in consecutive locations in memory. In Case2, each 3-B scale is placed at 64-B aligned location in memory.

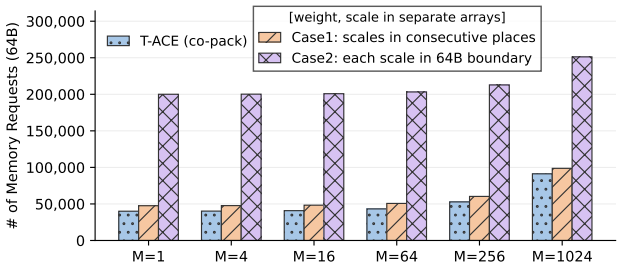


Figure 7: Number of memory block (64-B) requests for GEMM ($A_{M\times K} \times W_{K\times N}$ with K=N=3200) according to the number of rows (M) in activation matrix

In Case1, loading scales for the weight matrix requires 7,500 (= 3,200 × 3,200/64 × 3 B) / 64 B) block requests. For M=1, the total number of requests for loading activations, weights and scales is 47,550 = 50 (activations) + 40,000 (weights) + 7,500 (scales). For M=1,024, it is 98,700 = 51,200 (activations) + 40,000 (weights) + 7,500 (scales). Case2 requires a separate request for each scale. Since the tile contains 160,000 weight blocks (= 3,200 × 3,200 / 64), Case2 incurs 160,000 requests for scales. As shown in Fig. 7, typical schemes storing the scale metadata in separate array introduce a non-trivial amount of requests. Case1 adds 7,500 block requests for the weight matrix, which is an 8%–19% increase in total requests depending on M. Case2 inflates the scale traffic to as many as 160,000 additional requests, increasing the total by 175%–400%. Considering that the memory access has a huge impact on performance, the proposed packing scheme is essential for the inference efficiency.

4.3 T-ACE Design Evaluation on FPGA

This section reports the hardware cost and performance of T-ACE on FPGA. T-ACE is designed with Chisel [3], a hardware construction language. It is integrated into a Chipyard SoC [1], in place of Gemmini NPU. Chipyard SoC allows execution-level measurements under realistic system setting. It was synthesized and implemented using Vivado, targeting an UltraScale+ XCU280 FPGA on the Alveo U280 board [2]. This prototype enables cycle-accurate evaluation of dataflow, decoder behavior, and scale application.

Table 6 shows the FPGA resource utilization of T-ACE components. The Decoder and scale unit in T-ACE take only a small fraction of hardware resources: the decoder uses 9.23% of LUTs and 0.24% of FFs, compared with the compute unit, while the scale unit uses 1.05% of LUTs and 3.42% of FFs (no BRAM/DSP usage in either unit). It proves that the proposed on-the-fly decoding and scaling logic is lightweight. Fig. 8 shows the place-and-routed layout of T-ACE on the FPGA.

To examine whether the decoding and base-scale units introduce pipeline stalls, the latency of a GEMM execution is measured on the FPGA prototype. We have executed ternary GEMM workloads derived from the BitNet-b1.58 3B model, based on BS1 SEQ2048 and BS1024 SEQ1, two representative inference settings. BS1 SEQ2048 corresponds to the prefill phase with batch size = 1 and a long

Table 6: FPGA resource utilization of T-ACE blocks.

T-ACE HW Blocks	FPGA Resources			
	LUT	FF	BRAM	DSP
Decoder	35.2K (2.70%)	0.5K (0.02%)	0 (0.00%)	0 (0.00%)
Scale Unit	4.0K (0.30%)	7.0K (0.27%)	0 (0.00%)	0 (0.00%)
Compute Unit	381.3K (29.25%)	204.7K (7.85%)	0 (0.00%)	0 (0.00%)

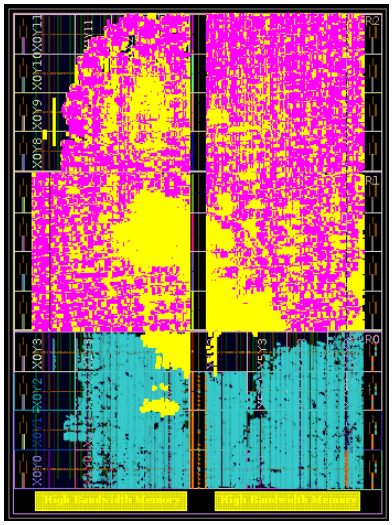


Figure 8: Place-and-routed view on UltraScale+ XCU280 FPGA. T-ACE is in pink. NPU subsystem including on-chip SRAM is in yellow)

sequence length (2048 tokens). This long sequence produces large-scale GEMMs with high arithmetic intensity. BS1024 SEQ1 represents the decode phase with 1024 parallel single-token requests, leading to much smaller, latency-sensitive GEMMs. Cycles are counted from the time an input tile is issued to the compute unit until the output tile is produced and written back to the on-chip SRAM. Off-chip memory effects are excluded. Cycle counts are collected using on-chip performance counters. Baseline is a simple ternary GEMM unit that is used as PE arrays in T-ACE.

Fig. 9 shows execution cycles. For both BS1 SEQ2048 and BS1024 SEQ1, T-ACE (with the decoder and scale unit) reports the execution cycles identical to the baseline. This is because T-ACE was designed with pipelining: Decoder processes the next block while PE array computes GEMM for the current one. The scaling is applied after GEMM in the following stage. As a result, the decoding and scale application are fully overlapped with computation, achieving the throughput the baseline provides.

4.4 T-ACE Design Evaluation with ASIC

This section reports the area and power breakdown of T-ACE based on synthesis result using an ASIC technology node. All T-ACE blocks are synthesized using Synopsys Design Compiler with the GSCL 45 nm standard-cell library [19]. To facilitate comparison across reference designs as in Table 4, the synthesis results are normalized to a 28 nm process node using the DeepScale tool [37].

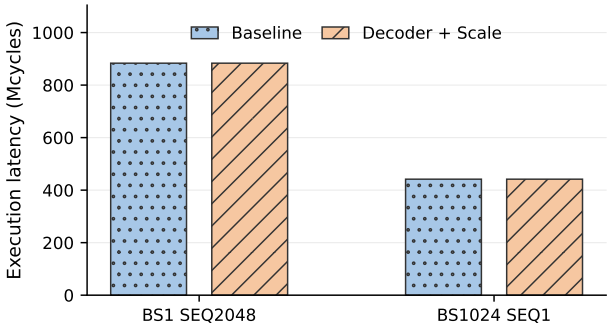


Figure 9: Execution latency (in cycles) measured on the FPGA prototype for ternary GEMM workloads from the BitNet-b1.58 3B model.

Table 7: Area and power breakdown of T-ACE at 28 nm node

T-ACE HW Blocks	#Units	Area / Unit (mm^2)	Total Area (mm^2)	Total Power (W)
Decoder	4	0.001893	0.007572	0.002803
Scale Unit	256	0.000448	0.114694	0.000356
Compute Unit	1 (256×64 *PEs*)	1.071024	1.071024	0.961094
Total	–	–	**1.193290**	**0.964253**

· All results are normalized to a 28 nm process.

Timing constraints are set so that all blocks satisfy the timing budget for 1 GHz operation in the 28 nm node.

Table 7 presents the area and power breakdown of T-ACE at 28 nm. Similar to Table 6 for FPGA, the Compute Unit integrating the PE array and adder tree takes the dominant area and power consumption in T-ACE. The decoder occupies only a small portion of the total area and power. It accounts for about 0.635% of area and 0.291% of power in T-ACE. It also shows that ternary unpacking and decoding logic incurs minimal hardware cost. The scale unit also has a small footprint because subgroup scaling is implemented with simple shifts instead of multipliers. Overall, the extra logic for fine-grained scaling and co-packed decoding remains lightweight.

5 Related Work

As LLMs continue to scale up, low-bit quantization has become a key technique for reducing model size and computational cost [9, 22, 47]. A wide range of methods explore per-tensor, per-channel, or block-wise quantization using quantization-aware training (QAT) [5, 29, 31, 38], whereas post-training quantization (PTQ) offers a light-weight alternative without costly retraining while maintaining accuracy [4, 11, 16, 20, 42]. These works show that finer-grained scaling improves accuracy by better capturing various weight distributions, but also increases the amount of scale metadata that must be stored and applied during inference, introducing additional overhead [6, 7, 10]. More recent work such as Microscaling (MX) [8] employs two-level shared scaling with power-of-two scale factors, enabling efficient shift-based implementation while preserving accuracy at low precision. Building on this insight, we adopt a MX-style two-level scaling and co-pack the scales with ternary weights in a unified format. This co-packed design reduces memory traffic by avoiding separate scale fetches and keeping weight and scale data contiguous in memory.

Recent ternary formats such as TQ1/TQ2 and I2_S pack multiple trits into a single byte to reduce model size and weight-transfer bandwidth in software inference frameworks [39]. Beyond compact storage, LUT-based compute units such as TMAC [41] and LUT Tensor Core [28] replace multipliers with table lookups to support efficient low-bit matrix multiplication and flexible (W, A) precision pairs (e.g., $W4A$FP16) without redesigning arithmetic datapaths. Custom ASIC and FPGA accelerators have explored efficient execution of ternary LLMs. TENET [17] introduces a multiplier-free ternary matrix–vector engine based on sign selection and accumulation to achieve high-throughput ternary computation in ASIC. TerEffic [45] targets FPGA deployment with structured tiling, multi-bank buffers to operate under limited on-chip resources. While these designs demonstrate the potential of ternary computation, efficient deployment remains challenging due to additional overhead from scale metadata handling, sub-byte data alignment, and memory access. These challenges motivate architectures that integrate lightweight decoding logic and scale handling into the data representation to enable efficient ternary execution.

6 Discussion and Limitations

Our accuracy results are obtained using a general-purpose PTQ with lightweight calibration, rather than a ternary-specialized quantization procedure. Recent work suggests that ternary models can benefit substantially from PTQ methods tailored to the extreme low-bit regime, such as improved outlier handling, more expressive yet hardware-friendly scaling, and targeted post-training fine-tuning (e.g., short QAT or LoRA-style adaptation). Applying such ternary-aware PTQ and fine-tuning techniques to T-ACE's representation is likely to further improve A_{rel} without changing the proposed co-packed format. Incorporating state-of-the-art ternary PTQ/QAT recipes could be a promising direction for future work.

Although T-ACE targets PTQ-oriented ternary deployment, its ternary-weight decoding path can also be used for QAT-based ternary models. Full compatibility, however, depends on the scale format. For instance, BitNet-b1.58 uses floating-point per-tensor scales, whereas T-ACE currently supports shift-based scale application. Thus, while the proposed decoding flow remains applicable, efficient end-to-end support for QAT-based models with floating-point scales would require additional scale-handling logic, which we leave as future work.

Extending the current hardware and kernel-level evaluation to the full-system using the FPGA environment (such as Alveo) is our future work. The present evaluation focuses on microarchitectural behavior and GEMM execution, and does not include integration with higher-level LLM software frameworks such as llama.cpp and bitnet.cpp. Integrating and running these LLM frameworks on T-ACE is left as future work to enable end-to-end evaluation of latency and system throughput under realistic workloads.

While T-ACE's 256×64 PE array is primarily optimized for high-throughput GEMM workloads, LLM autoregressive decoding often involves GEMV-like or highly asymmetric matrix shapes, which

can lead to reduced PE-array utilization. We plan to address this limitation in future work through flexible hardware partitioning.

7 Conclusion

This paper presents *T-ACE*, the Ternary Accuracy-aware Compute Engine, for ternary LLM inference under post-training quantization. T-ACE addresses a key practical challenge of ternary models: achieving high hardware efficiency without sacrificing model quality when fine-grained scaling is required. T-ACE enables efficient ternary GEMM execution while supporting the scaling flexibility needed for accurate PTQ ternary inference. Experimental results demonstrate that T-ACE achieves higher accuracy-aware efficiency than A100/H100 Tensor Core baselines under our normalized setting. Compared to the GPUs, T-ACE provides 66.8% improvement in ACD and 17.6% improvement in AEE. In terms of raw hardware capability, T-ACE also provides substantially higher compute density and energy efficiency, delivering 43.14 TOPS/mm^2 and 34.20 TOPS/W. The 16-B aligned packing scheme dramatically reduces the memory traffic for loading weights and scale metadata. The area and power evaluations for both FPGA and ASIC indicate that the Decoder and scaling units incur negligible overhead. Overall, T-ACE provides a practical blueprint for deploying ternary LLMs on cost- and energy-constrained platforms, bridging the gap between aggressive ternary compression and reliable real-world inference.

Acknowledgments

This work was partially supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. RS-2024-00459774, RISC-V based system software development for open ecosystem of SDR; No. RS-2025-02304483, Chiplet-Based Hub SoC Development Optimized for On-Device AI). This work was supported by a research grant from Seoul Women's University (2026-0054). This work was supported in part by the NATO Science for Peace and Security Programme under Grant G7200. Sangwoo Park and Taeweon Suh are co-corresponding authors.

References

[1] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Jonathan Bachrach, Sophia Shao, Borivoje Nikolić, and Krste Asanović. 2020. Invited: Chipyard – An Integrated SoC Research and Implementation Environment. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC '20)*. San Francisco, CA, USA, 1–6. doi:10.1109/DAC18072.2020.9218756

[2] Avnet, Inc. 2019. *Alveo U280 FPGA Accelerator Card Datasheet*. https://www.avnet.com/fsp/opasdata/d120001/medias/docus/196/XLX-A-U280-A32G-DEV-G-Datasheet.pdf Accessed: 2026.

[3] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: Constructing Hardware in a Scala Embedded Language. In *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*. San Francisco, California, 1216–1225. doi:10.1145/2228360.2228584

[4] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2023. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems* 36 (2023), 4396–4429.

[5] Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. 2025. EfficientQAT: Efficient Quantization-Aware Training for Large Language Models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 10081–10100.

[6] Kamran Chitsaz, Quentin Fournier, Gonçalo Mordido, and Sarath Chandar. 2024. *Exploring Quantization for Efficient Pre-training of Transformer Language Models*. arXiv:2407.11722

[7] Steve Dai, Rangha Venkatesan, Mark Ren, Brian Zimmer, William Dally, and Brucek Khailany. 2021. VS-Quant: Per-vector Scaled Quantization for Accurate Low-precision Neural Network Inference. *Proceedings of Machine Learning and Systems* 3 (2021), 873–884.

[8] Bita Darvish Rouhani, Ritchie Zhao, Venmugil Elango, Rasoul Shafipour, Mathew Hall, Maral Mesmakhosroshahi, Ankit More, Levi Melnick, Maximilian Golub, Girish Varatkar, et al. 2023. With shared microexponents, a little shifting goes a long way. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[9] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems* 35 (2022), 30318–30332.

[10] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*, Vol. 36. 10088–10115.

[11] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. arXiv:2210.17323

[12] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving Open Language Models at a Practical Size. arXiv:2408.00118 [cs.CL] https://arxiv.org/abs/2408.00118

[13] Georgi Gerganov et al. 2023. *llama.cpp: Efficient LLaMA inference in C/C++*. https://github.com/ggml-org/llama.cpp GitHub repository, accessed: 2025.

[14] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[15] Yufei Guo, Zecheng Hao, Jiahang Shao, Jie Zhou, Xiaode Liu, Xin Tong, Yuhan Zhang, Yuanpei Chen, Weihang Peng, and Zhe Ma. 2025. PT-BitNet: Scaling up the 1-Bit Large Language Model with Post-Training Quantization. *Neural Networks* 191 (Nov. 2025). doi:10.1016/j.neunet.2025.107855

[16] Hong Huang, Decheng Wu, Rui Cen, Guanghua Yu, Zonghang Li, Kai Liu, Jianchen Zhu, Peng Chen, Xue Liu, and Dapeng Wu. 2025. Tequila: Trapping-free ternary quantization for large language models. *arXiv preprint arXiv:2509.23809* (2025).

[17] Zhirui Huang, Rui Ma, Shijie Cao, Ran Shu, Ian Wang, Ting Cao, Chixiao Chen, and Yongqiang Xiong. 2025. *TENET: An Efficient Sparsity-Aware LUT-Centric Architecture for Ternary LLM Inference On Edge*. arXiv:2509.13765

[18] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. *Mistral 7B*. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[19] Gilles Lamant. 2002. *Generic Standard Cell Library: Functional Specifications*. Cadence Design Systems, San Jose, CA. Revision 0.8.

[20] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems* 6 (2024), 87–100.

[21] Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, Lin Xiao, Yuandong Tian, Bilge Soran, Raghuraman Krishnamoorthi, Tijmen Blankevoort, and Vikas Chandra. 2025. ParetoQ: Improving Scaling Laws in Extremely Low-bit LLM Quantization. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*. https://openreview.net/forum?id=PMSNd8xTHp NeurIPS 2025 poster.

[22] Shuming Ma, Hongyu Wang, Shaohan Huang, Xingxing Zhang, Ying Hu, Ting Song, Yan Xia, and Furu Wei. 2025. *BitNet b1.58 2B4T Technical Report*. arXiv:2504.12285

[23] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. 2024. *The Era of 1-bit LLMs: All Large Language Models Are in 1.58 Bits*. arXiv:2402.17764

[24] Meta AI. 2024. *Llama 3.2 1B*. https://huggingface.co/meta-llama/Llama-3.2-1B Hugging Face model card, accessed: 2025.

[25] Meta AI. 2024. *Llama 3.2 3B*. https://huggingface.co/meta-llama/Llama-3.2-3B Hugging Face model card, accessed: 2025.

[26] Microsoft. 2025. *microsoft/BitNet: Official inference framework for 1-bit LLMs*. https://github.com/microsoft/BitNet Accessed: 2025.

[27] Microsoft. 2026. MicroXcaling: PyTorch Emulation Library for Microscaling (MX)-Compatible Data Formats. https://github.com/microsoft/microxcaling. Accessed: 2026.

[28] Zhiwen Mo, Lei Wang, Jianyu Wei, Zhichen Zeng, Shijie Cao, Lingxiao Ma, Naifeng Jing, Ting Cao, Jilong Xue, Fan Yang, and Mao Yang. 2025. LUT Tensor Core: A Software-Hardware Co-Design for LUT-Based Low-Bit LLM Inference. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 514–528. doi:10.1145/3695053.3731057

[29] Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. 2022. Overcoming Oscillations in Quantization-Aware Training. In *International Conference on Machine Learning*. 16318–16330.

[30] Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. *Adversarial NLI: A New Benchmark for Natural Language Understanding*. arXiv:1910.14599 [cs.CL] https://arxiv.org/abs/1910.14599

[31] Jacob Nielsen, Lukas Galke, and Peter Schneider-Kamp. 2025. When Are 1.58 Bits Enough? A Bottom-up Exploration of Quantization-Aware Training with Ternary Weights. In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART)*.

[32] NVIDIA Corporation. 2020. *NVIDIA Ampere GPU Architecture*. https://www.nvidia.com/en-us/data-center/ampere-architecture/

[33] NVIDIA Corporation. 2022. *NVIDIA Hopper GPU Architecture Technical Brief*. https://www.nvidia.com/en-us/data-center/technologies/hopper-architecture/

[34] Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (Long and short papers)*. 1267–1273.

[35] Ye Qiao, Zhiheng Chen, Yifan Zhang, Yian Wang, and Sitao Huang. 2025. *TeLLMe: An Energy-Efficient Ternary LLM Accelerator for Prefilling and Decoding on Edge FPGAs*. arXiv:2504.16266

[36] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Commun. ACM* 64, 9 (Aug. 2021), 99–106. doi:10.1145/3474381

[37] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5.

[38] Shyam A. Tailor, Javier Fernandez-Marques, and Nicholas D. Lane. 2021. Degree-Quant: Quantization-Aware Training for Graph Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[39] Jinheng Wang, Hansong Zhou, Ting Song, Shijie Cao, Yan Xia, Ting Cao, Jianyu Wei, Shuming Ma, Hongyu Wang, and Furu Wei. 2025. BitNet.cpp: Efficient Edge Inference for Ternary LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 9305–9322.

[40] Xinyu Wang, Vahid Partovi Nia, Peng Lu, Jerry Huang, Xiao-Wen Chang, Boxing Chen, and Yufei Cui. 2025. PoTPTQ: A Two-step Power-of-Two Post-training for LLMs. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*. https://openreview.net/forum?id=QjRIozXQNy

[41] Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. 2025. T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge. In *Proceedings of the Twentieth European Conference on Computer Systems*. 278–292. doi:10.1145/3689031.3696099

[42] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *International Conference on Machine Learning*. 38087–38099.

[43] He Xiao, Runming Yang, Qingyao Yang, Wendong Xu, Zhen Li, Yupeng Su, Zhengwu Liu, Hongxia Yang, and Ngai Wong. 2026. *PTQTP: Post-Training Quantization to Trit-Planes for Large Language Models*. arXiv:2509.16989 [cs.LG] https://arxiv.org/abs/2509.16989

[44] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. *Qwen2 Technical Report*. arXiv:2407.10671 [cs.CL] https://arxiv.org/abs/2407.10671

[45] Chenyang Yin, Zhenyu Bai, Pranav Venkatram, Shivam Aggarwal, Zhaoying Li, and Tulika Mitra. 2025. *TerEffic: Highly Efficient Ternary LLM Inference on FPGA*. arXiv:2502.16473

[46] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence?. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

[47] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812* (2020).