Exploration of Unary Arithmetic-Based Matrix Multiply Units for Low Precision DL Accelerators

Prabhu Vellaisamy*, Harideep Nair*, Di Wu†, Shawn Blanton*, and John Paul Shen*
*Electrical and Computer Engineering Department, Carnegie Mellon University
{pvellais, hpnair, rblanton, jpshen}@andrew.cmu.edu
†Department of Electrical and Computer Engineering, University of Central Florida
di.wu@ucf.edu

2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 979-8-3503-5411-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISVLSI61997.2024.00126

***Abstract*—General matrix multiplication (GEMM) is a fundamental operation in deep learning (DL). With DL moving increasingly toward low precision, recent works have proposed novel unary GEMM designs as an alternative to conventional binary GEMM hardware. A rigorous evaluation of recent unary and binary GEMM designs is needed to assess the potential of unary hardware for future DL compute. This paper focuses on unary GEMM designs for integer-based DL inference and performs a detailed evaluation of three latest unary design proposals, namely, uGEMM, *tu*GEMM and *tub*GEMM, by comparing them to a conventional binary GEMM. Rigorous post-synthesis evaluations beyond prior works are performed across varying bit-widths and matrix sizes to assess the designs' tradeoffs and determine optimal sweetspots. Further, we perform weight sparsity analysis across eight pretrained convolutional neural networks (CNNs) and the LLaMA2 large language model (LLM). In this work we demonstrate how unary GEMM can be effectively used for energy-efficient compute in future edge AI accelerators.**

***Index Terms*—unary computing, matrix multiplication, deep learning accelerators, low-precision deep learning inference**

I. Introduction

Recent advancements in artificial intelligence (AI), particularly deep learning (DL), have led to notable achievements, surpassing human performance in tasks like image and speech recognition. Despite these successes, computation cost for running DL models is ever-increasing at an exponential rate [1]. This has led to the development of deep learning accelerators (DLAs) with dedicated hardware to optimize general matrix multiplication (GEMM), the fundamental operation in DL. Prominent examples include tensor cores in modern GPUs and matrix multiplication units (MXUs) in Google's Tensor Processing Units (TPUs). Additionally, devices such as edge TPU and NVIDIA's Jetson exemplify the trend toward edge computing. These DLAs leverage binary arithmetic optimizations and are characterized by their use of low-precision arithmetic and optimized dataflow to achieve hardware efficiency.

Unary computing, touted as a promising alternative to relatively complex binary arithmetic, offers unique advantages in computational efficiency, especially for low-precision tasks prevalent in AI/DL. It manifests in two forms: (i) *rate-coding* and (ii) *temporal-coding*. In rate-unary computing, data is encoded based on the frequency of 1s, directly proportional to the represented value. Temporal-unary encoding represents data as a consecutive sequence of 1s followed by 0s, with the number of 1s indicating the data value. Traditionally, rate-unary encoding allows for low-complexity arithmetic hardware suitable for stochastic computing. It significantly simplifies circuits, such as using single AND gates as multipliers and multiplexers as adders [2], providing a reasonable approximation of the ideal result with an inherent accuracy compromise. In contrast, temporal-unary encoding enables exact deterministic compute.

Among the latest innovations in this field are three unary GEMM architectures: (i) uGEMM, a unified rate-and-temporal-encoded design executing stochastic GEMM operations [3], (ii) *tu*GEMM, the first fully-temporal GEMM design ensuring deterministic compute with full accuracy [4], and (iii) *tub*GEMM, a novel temporal-binary hybrid successor to *tu*GEMM that significantly reduces latency with relatively modest hardware overhead, improving overall energy efficiency [5]. Despite comparisons against previous unary GEMM approaches, a holistic evaluation of the unary GEMM designs against traditional binary GEMM designs in today's DLAs has not been explored in literature.

Unary computing gains prominence in DL inference as the industry gravitates toward lower precision computing, propelled by advancements in quantization techniques. Although precision scaling poses challenges during training, strides have been made in retaining accuracy from FP32 (32-bit floating point) to as low as FP8 (8-bit floating point) for training and as low as INT4 and INT2 (4-bit and 2-bit integer formats, respectively) for inference [6, 7]. In the domain of large language models (LLMs), NVIDIA's Grace Hopper, with its FP8 Transformer Engine, demonstrates significant speedups for transformer-based models. Latest works like BitNet b1.58 [8], utilizing ternary weights, pave the way for 1-bit LLMs.

This paper aims to explore the potential of unary GEMM in enhancing the computational efficiency of edge DLAs for low-precision AI inference. By comparing three latest unary GEMM designs that outperform prior unary works with traditional binary GEMM, we seek to understand their relative strengths and limitations. This comparative analysis, complemented by sparsity profiling of DL models, aims to provide insights and recommendations for future DLAs to facilitate more sustainable edge AI deployment. Key contributions are:

- Current landscape for unary GEMM compute is largely unevaluated, with unary-based designs developed in an *ad-hoc* manner. Our work is a first attempt at contextual-

DOI 10.1109/ISVLSI61997.2024.00126

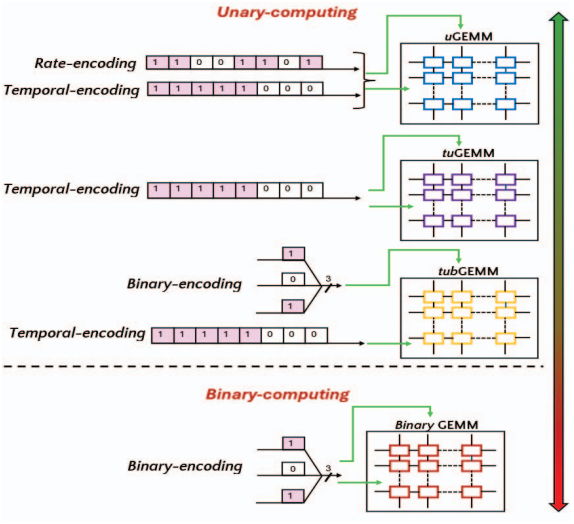

Fig. 1: Four GEMM designs: uGEMM encodes both inputs as unary (rate/temporal), *tu*GEMM encodes both inputs as temporal-unary, *tub*GEMM encodes one input as temporal-unary and other binary, and the conventional binary GEMM. The vertical ordering of the four designs reflects a spectrum from *fully-unary* design (green) to a *fully-binary* design (red).

izing the latest unary research for INT-based AI inference by juxtaposing and evaluating three recent unary GEMM designs against a traditional binary GEMM. Prior works [3][4][5] only compare against other unary GEMM.

- Previous works [3][4][5] utilize different process technologies (TSMC45 vs. Nangate45) and compare designs using only one configuration (8-bit, 16x16 GEMM). This paper extends beyond those works by comparing across varying bit-widths and matrix sizes using a single technology (Nangate45). This enables fair comparison across all architectures, providing a more holistic evaluation.
- Weight sparsities of eight CNNs and LLaMA2 LLM are profiled to assess unary designs' ability to inherently exploit sparsity to improve latency and energy efficiency.
- Design tradeoffs are assessed amongst the unary and binary designs, and the optimal design sweetspot for hardware efficiency is determined. Further, potential future directions for edge DLAs are discussed, anticipating a new wave of unary arithmetic research for AI compute.

II. Background on Unary GEMM Architectures

We first review the three unary GEMM designs (Figure 1).

A. uGEMM

uGEMM [3] adopts a *unified*-unary GEMM architecture, supporting both rate-coding and temporal-coding. It introduces *unified*-unary computing units for multiplication (uMULT), scaled addition (uSADD), and non-scaled addition (uNSADD). To mitigate long latency and high energy consumption, it incorporates *early termination*, reporting a 98% increase in energy efficiency and a minimal loss of less than 0.5% from the ideal output. While being the first to support fully streaming execution, a non-streaming variant is evaluated for a fair comparison with other designs, using binary inputs converted to unary-coded inputs. uGEMM requires 2^w cycles for a GEMM computation, with w as bitwidth. Parallel adder trees are used to accumulate all partial sum bitstreams.

B. tuGEMM

*tu*GEMM [4] is the pioneering counter-based GEMM architecture supporting deterministic computing with fully temporal-encoding. Two design variants, *serial* and *parallel*, offer different area-latency tradeoffs. *tu*GEMM achieves significantly lower area and power consumption but at the cost of quadratically worse latency relative to uGEMM. Authors in [4] present post-synthesis results for different low bit-precisions and matrix sizes but comparison with uGEMM is only performed for 8-bit 16x16 GEMM. Latency for *tu*GEMM scales exponentially with bitwidth due to operation on nested temporal-coded bitstreams, leading to a worst-case latency of $N * (2^{w-1})^2$ cycles for a GEMM compute, with N as the common dimension of input matrices and w as the bitwidth.

C. tubGEMM

*tub*GEMM [5] utilizes hybrid temporal-unary and binary (tub) encoding, performing exact (deterministic) computation using sequential multipliers and accumulators. Compared to uGEMM, it achieves substantial reductions in area (89%), power (87%), and energy (50%) for 8-bit 16x16 GEMM in 45nm CMOS. Notably, authors in [5] also evaluate *tub*GEMM across varying bitwidths and matrix sizes on TSMC N5 process node. Relative to *tu*GEMM, *tub*GEMM reduces worst-case latency to $N * (2^{w-2})$ cycles, optimizing its temporal encoding with a novel 2-*unary* scheme that halves the latency.

III. Evaluation Methodology

This section details the evaluation methodology for post-synthesis Power-Performance-Area (PPA), ensuring fair comparison among all GEMM designs. Additionally, the setup for weight sparsity analysis profiled for DL models is described.

A. Setup for PPA Analysis

All GEMM designs are synthesized using Nangate45 open-source library with Synopsys Design Compiler, operating at a clock frequency of 400 MHz. Synthesis is performed exhaustively for 2-, 4-, and 8-bits across two matrix sizes: 16x16 and 32x32. The chosen bitwidths and matrix sizes are amenable for edge AI inference in modern mobile system-on-chips (SoCs). All GEMM designs implement non-scaled bipolar compute [3] and utilize the outer product dataflow as in [4, 5]. The binary-based GEMM ('bGEMM') serves as our baseline benchmark. bGEMM is synthesized using DesignWare multipliers and adders, and incurs N cycles for a single GEMM compute with outer product dataflow. Along with area and power results, we further derive energy and area-delay product (ADP) metrics based on the GEMM latencies. Note that ADP inherently captures and normalizes the spatio-temporal trade-offs in the unary GEMM designs, such as in

TABLE I: 45nm post-synthesis area (in μm^2) for the four GEMM designs with varying bit-widths and matrix sizes.

Configuration		uGEMM	*tu*GEMM	*tub*GEMM	bGEMM
2-bit	16x16	99,445.7	13,436.4	19,112.6	16,739.1
	32x32	791,794.4	52,272.4	76,375.5	67,201.7
4-bit	16x16	203,920.7	29,061.0	38,912.6	44,925.8
	32x32	1,799,961.0	117,261.3	151,933.6	180,458.6
8-bit	16x16	445,396.2	61,064.0	99,916.8	132,786.9
	32x32	3,689,829.0	235,470.9	338,692.7	560,778.5

TABLE II: 45nm post-synthesis power (in mW) for the four GEMM designs with varying bit-widths and matrix sizes.

Configuration		uGEMM	*tu*GEMM	*tub*GEMM	bGEMM
2-bit	16x16	42.2	4.9	5.0	7.7
	32x32	323.8	18.3	19.8	30.9
4-bit	16x16	64.1	9.2	9.9	22.4
	32x32	513.6	37.2	39.1	88.3
8-bit	16x16	100.8	19.7	26.1	72.8
	32x32	784.4	74.7	90.9	321.3

serial (considered here) vs. parallel *tu*GEMM (omitted for brevity). Worst-case (WC) latency is calculated by multiplying compute cycles (Sec. II) with the clock period (2.5 ns).

B. Setup for Weight Sparsity Analysis

Two types of weight sparsities are considered: 1) Word sparsity signifies zero values (percentage of weights that have zero magnitude); 2) Bit sparsity denotes small magnitude values (percentage of '0' bits in the temporal-unary bitstream). In the extreme case when all bits are '0's, bit sparsity subsumes word sparsity. Higher bit sparsity (*b_spa*) implies a large number of '0' bits (i.e., small number of '1' bits) in the temporal-unary bitstream, leading to lower dynamic latency (and energy consumption) for *tu*GEMM and *tub*GEMM:

$$\text{Dynamic Latency} = \text{WC Latency} * (1 - b\_spa) \quad (1)$$

Eight pretrained quantized INT8 CNNs are imported from Torchvision for sparsity profiling: 1) MobileNetV2, 2) MobileNetV3, 3) InceptionV3, 4) ShuffleNetV2, 5) GoogleNet, 6) ResNet18, 7) ResNet50, and 8) ResNeXt101. These models are widely used in leading DL literature. Similar to the methodology in [5], maximum values within each feature map are tracked, and the corresponding counts are averaged across convolution and fully connected layers to derive bit sparsity.

Additionally, a pretrained quantized INT32 LLaMA2-70B model from Huggingface is profiled. Due to constraints on accessing low-precision LLaMA2 model weights, a pragmatic approach is adopted by considering the most significant bits (MSBs) for sparsity profiling, as used in prior works [7, 9] without impacting the distribution and sparsity significantly. The weights are extracted from two fully connected (FC) layers in the attention layer and a 2-layer feed-forward network (FFN) after the attention layer. We also profile the query (Q) and key (K) tokens in the attention layer, using small representative inputs. Average maximum value per 32x32 block is considered, as largest value bottlenecks GEMM compute.

IV. Experimental Results

This section presents four types of evaluation for the GEMM designs (uGEMM, *tu*GEMM, *tub*GEMM, bGEMM) across 8, 4, 2-bit precisions and 16x16, 32x32 matrix sizes: 1) 45nm CMOS post-synthesis area-power results, 2) Energy results derived from worst-case latencies, 3) Area-Delay Product (ADP) results derived from worst-case latencies, and 4) Workload-dependent sparsity and corresponding latency-energy results. Further, PPA results are also shown for larger 64x64 and 128x128 matrix sizes for 4-bit precision.

A. Area-Power Evaluation

Tables I and II illustrate the 45nm post-synthesis cell area and total power, and are pivotal for gauging trade-offs in GEMM hardware efficiency for AI inference. The lowest (best) and highest (worst) values are marked in green and red, respectively. *tu*GEMM outperforms all other designs in area-power efficiency across all configurations owing to its simplistic counter-based streamlined architecture without the need for any huge adder trees. As it relies predominantly on counters for the temporal accumulation of vector-vector products, it incurs very high latency leading to highest energy consumption among all designs (Table III as will be explained in Sec. IV-B). *tub*GEMM is the next optimal design in area-power efficiency across 4-bits and 8-bits. bGEMM outperforms *tub*GEMM in area at 2-bits but scales considerably worse with increasing bitwidth. uGEMM, while versatile in supporting both rate-coding and temporal-coding, exhibits lower area-power efficiency across all designs due to its unified unary approach. Figure 2 plots these results for 32x32 GEMM.

More specifically, uGEMM, *tub*GEMM and bGEMM are approximately 7x, 1.5x and 2x worse than *tu*GEMM on average for 16x16 GEMM area. uGEMM scales poorly with matrix size resulting in an increased gap of 15x for 32x32 GEMM, while the gap remains consistent for *tub*GEMM and bGEMM, implying *tu*GEMM, *tub*GEMM and bGEMM scale similarly with matrix sizes. In power, *tu*GEMM and *tub*GEMM are close and consistently outperform uGEMM and bGEMM, with uGEMM consuming the most power (about 10x for 8-bit 32x32 compared to *tu*GEMM). The power efficiency of *tu*GEMM and *tub*GEMM is superior mainly because temporal-unary encoding results in only two signal transitions due to consecutive ones followed by zeros, in contrast to rate-unary and binary encoding with multiple signal transitions.

In terms of bitwidth scaling, all designs scale linearly on log scale (green trendlines in Figure 2). However, the slopes are different for each of the designs (lower slope indicates better scaling). Specifically, for area, *tu*GEMM and *tub*GEMM scale best (2.12 slope), closely followed by uGEMM (2.16) and finally bGEMM (2.90). Similarly, the four designs incur slopes of 2.02, 2.15, 1.56 and 3.25 respectively for power, indicating best scaling for uGEMM. uGEMM's superior bitwidth scaling is due to its simpler stochastic single-gate multiplier. In contrast, with increasing matrix sizes, the adder trees in uGEMM become significantly denser resulting in poor scaling.

Key Takeaway: *tu*GEMM has the best area-power efficiency, closely followed by *tub*GEMM and bGEMM. uGEMM

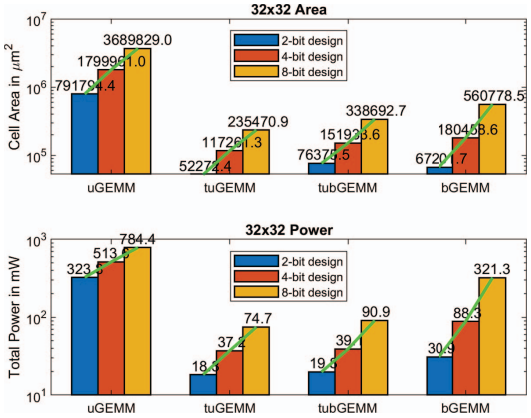

Fig. 2: 45nm post-synthesis area and power scaling across 2, 4, and 8-bits for 32x32 matrix size. Y-axis is in log scale.

is almost 10x worse. *tu*GEMM and *tub*GEMM scale well with bitwidths and matrix sizes. uGEMM scales well with bitwidths but poorly with matrix sizes, and vice versa for bGEMM.

B. Energy Evaluation with Worst-Case Latency

Table III shows energy consumption based on worst-case latency for each GEMM design. Despite its optimal area-power efficiency, *tu*GEMM incurs the highest energy due to its nested counter-based temporal accumulation, resulting in significant latency for 8-bits and above. bGEMM is the most energy-efficient for 4- and 8-bits owing to just N cycles per GEMM independent of bitwidth, whereas *tub*GEMM outperforms bGEMM for 2-bits. *tub*GEMM is reasonably close (1.8x) to bGEMM's energy consumption at 4 bits, with uGEMM trailing behind (2.9x). With these findings, 4-bit precision emerges as a focal point for detailed analysis. Further post-synthesis results for larger Google EdgeTPU (64x64) and CloudTPUv3 (128x128) matrix sizes are detailed in Table IV for 4-bits. It shows *tu*GEMM as most efficient in area and power but least in energy, as expected. Notably, *tub*GEMM consumes just 1.2x more energy than bGEMM for EdgeTPU (64x64) but outperforms bGEMM at CloudTPUv3 (128x128) array size, resulting in 12% more energy efficiency, even with worst-case latency. This suggests optimal scalability for *tub*GEMM targeting DLAs with large (beyond 64x64) 4-bit and 2-bit processing element (PE) arrays.

Key Takeaway: *tub*GEMM is most energy-efficient at 2-bits, and is comparable with bGEMM at 4-bits. For large PE arrays, *tub*GEMM emerges as a better candidate than bGEMM, owing to its high scalability of energy efficiency with matrix sizes, even with worst-case latency.

C. Area-Delay Product Evaluation with Worst-Case Latency

Table IV also shows Area-Delay Product (ADP) values for 64x64 and 128x128 matrix sizes. ADP is a useful metric to analyse the spatio-temporal trade-offs of GEMM designs. Table IV shows that bGEMM has the lowest (best) ADP due to its minimal latency, closely followed by *tub*GEMM with

TABLE III: 45nm post-synthesis energy (in nJ) for worst-case GEMM latencies with varying bit-widths and matrix sizes.

Configuration		uGEMM	*tu*GEMM	*tub*GEMM	bGEMM
2-bit	**16x16**	0.42	**0.78**	**0.20**	0.31
	32x32	3.24	**5.86**	**1.58**	2.47
4-bit	**16x16**	2.56	**23.55**	1.58	**0.90**
	32x32	20.54	**190.46**	12.51	**7.06**
8-bit	**16x16**	64.51	**12,910.59**	66.82	**2.91**
	32x32	502.02	**97,910.78**	465.41	**25.70**

TABLE IV: 45nm post-synthesis area, power, and energy (with worst-case latency) for EdgeTPU (64x64) and Cloud TPUv3 (128x128) GEMM sizes for 4-bit precision.

Configuration		uGEMM	*tu*GEMM	*tub*GEMM	bGEMM
4-bit Area (mm^2)	**64x64**	**15.89**	**0.46**	0.59	1.09
	128x128	**140.24**	**1.83**	2.41	6.64
4-bit Power (mW)	**64x64**	**4,115.21**	**145.52**	154.42	496.77
	128x128	**32,973.04**	**579.28**	620.92	2,794.80
4-bit Energy (nJ)	**64x64**	164.61	**1,490.12**	98.83	**79.48**
	128x128	1,318.92	**11,863.65**	**794.78**	894.34
4-bit ADP (mm^2-ns)	**64x64**	635.6	**4,710.4**	377.6	**174.4**
	128x128	5,609.6	**37,478.4**	3,084.8	**2,124.8**

2.2x and 1.5x higher ADP for 64x64 and 128x128 arrays, respectively (this gap is reduced with increasing matrix sizes). uGEMM's ADP is ~3x higher than bGEMM on average and *tu*GEMM's ADP is ~20x higher (infeasibly large).

Key Takeaway: Despite unary designs having better area than bGEMM, the substantial latency increase results in worse ADP, indicating potential room for area-latency improvement.

D. Sparsity-Driven Latency-Energy Evaluation

The weight sparsity profiling results are summarized in Table V. Across most CNN models, there is a consistent word sparsity of approximately 2%, with MobileNetV3 being the outlier with 9.5% of its weights as zeros. For the LLaMA2-70B LLM model, the 8-bit self-attention tokens exhibit comparable weight sparsity, averaging around 2.8% (equivalent to approximately 2B zero weights out of the total 70B weights). 4-bit (36%) and 2-bit (84%) word sparsities are significantly higher. In contrast, the attention FC and FFN layers involve much lower word sparsities (negligible for 8-bits).

Bit sparsity subsumes word sparsity and directly translates to latency and energy improvements for *tu*GEMM and *tub*GEMM. CNNs display significantly better bit sparsity (~43%) compared to LLM (negligible for 8-bit FC/FFN layers and ~30% for tokens). 4- and 2-bit values show varying sparsities for LLM. Plugging in the bit sparsity values from Table V (in fractional form) into Equation 1, DL workload-dependent energy values are derived for 8-, 4-, and 2-bits for 32x32 *tu*GEMM and *tub*GEMM. Note, only the two temporal-unary designs can leverage bit sparsity. These values are plotted to the right in Figure 3 with worst-case energy values to the left. Three notable improvements for *tub*GEMM upon leveraging sparsity are: 1) Enhanced 2-bit energy efficiency, further increasing the gap with bGEMM. 2) Earlier cross-over point with bGEMM, indicating *tub*GEMM can now outperform or perform on par with bGEMM for 3-bits. 3) More discernable energy gap with uGEMM at 8-bits.

TABLE V: Profiled weight sparsities of CNNs and LLM. Word sparsity denotes percentage of zero weights; bit sparsity denotes percentage of zero bits within temporal-unary weights.

CNN	Word (%) 8 bits	Bit (%) 8 bits
MobileNetV2	2.25	44.66
MobileNetV3	9.52	38.59
GoogleNet	1.91	45.91
InceptionV3	1.99	45.61
ShuffleNetV3	1.43	47.18
ResNet18	2.04	45.3
ResNet50	2.45	46.24
ResNeXt101	2.64	44.23
LLaMA2-70B Layer (LLM)	**Word (%) 2/4/8 bits**	**Bit (%) 2/4/8 bits**
Attention FC layer weight	20.7 / 2.85 / 0.0613	50.00 / 12.50 / 0.82
FFN layer weight	20.8 / 3.02 / 0.0524	50.00 / 12.5 / 0.80
Self attention Q	82.8 / 35.0 / 2.71	0.56 / 8.89 / 28.84
Self attention K	85.1 / 37.4 / 2.94	8.19 / 8.58 / 32.52

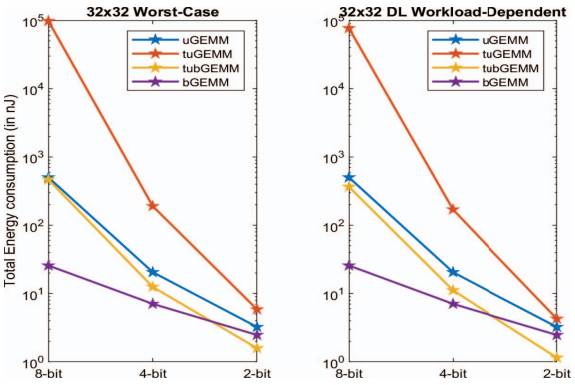

Fig. 3: Energy consumption for 32x32 GEMM across 8-, 4- and 2-bits. Y-axis is in log scale. Compared to left worst-case plot, note for *tubGEMM*, increased energy efficiency at 2 bits, earlier cross-over point with bGEMM, and larger energy gap to uGEMM at 8 bits, with sparsity (right plot).

Key Takeaway: Overall, *tub*GEMM stands out as the best design for low-precision AI inference (4 and 2 bits) due to its high area-power-energy efficiency, further enhanced through bit sparsity in DL workloads. *tu*GEMM's low hardware complexity makes it reasonable for applications (especially 2 bits) where area and power are highly constrained but high latency can be tolerated. uGEMM is suitable where the compute infrastructure demands rate-unary inputs, particularly for small matrix sizes. Finally, bGEMM is desirable for low-latency compute, especially for 8-bits and above.

V. Potential for Unary-based AI Compute

Our findings suggest the following areas for further exploring of unary based designs for energy-efficient AI inference:

1) Temporal-Unary Compute: While rate-unary methods in stochastic computing have been widely studied, temporal-unary computing remains under-explored. Temporal-unary designs like *tu*GEMM and *tub*GEMM, which use standard digital circuits, show promise for AI applications. Stochastic computing can lead to accuracy loss, as seen when a 96.08% accurate INT8 quantized MLP model drops to 94.7% with uGEMM. Temporal-unary, however, offers deterministic computation without this accuracy degradation. The *n*-unary encoding from [5] presents opportunities for further latency and energy efficiency improvements, with possible schemes to offset any trade-offs. With the *tub*GEMM design displaying PPA results close to conventional binary GEMM design, further design optimization can lead to it outperforming across all configurations, especially for weights of 2-4 bits.

2) Leveraging Sparsity: The sparsity results in Table V and analysis in Sec. IV-D highlight opportunities for bit sparsity exploitation. Given the growing importance of pruning and SparseML libraries in edge AI, naturally exploiting sparsity, as seen in *tu*GEMM and *tub*GEMM, offers substantial prospects for improving latency and energy efficiency.

3) Low-Precision Compute: Given the ongoing advancements in quantization techniques [6–9], AI inference is progressively embracing ultra-low bit-precisions. This trend affords unary designs the opportunity for exponential reductions in compute latency, rendering them a viable alternative to binary-based processing elements for 4 bits and below.

References

[1] OpenAI. (2018) Ai and compute. [Online]. Available: https://openai.com/blog/ai-and-compute/

[2] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *TCAD*, vol. 37, no. 8, pp. 1515–1531, 2017.

[3] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. San Miguel, "uGEMM: Unary Computing Architecture for GEMM Applications," in *ISCA*, 2020.

[4] H. Nair, P. Vellaisamy, A. Chen, J. Finn, A. Li, M. Trivedi, and J. P. Shen, "tugemm: Area-power-efficient temporal unary gemm architecture for low-precision edge ai," in *2023 ISCAS*. IEEE, 2023.

[5] P. Vellaisamy, H. Nair, J. Finn, M. Trivedi, A. Chen, A. Li, T.-H. Lin, P. Wang, S. Blanton, and J. P. Shen, "tubgemm: Energy-efficient and sparsity-effective temporal-unary-binary based matrix multiply unit," in *2023 ISVLSI*. IEEE, 2023, pp. 1–6.

[6] N. Wang, J. Choi, and K. Gopalakrishnan, "8-bit precision for training deep learning systems," 2018.

[7] J. Choi, S. Venkataramani, V. V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 348–359, 2019.

[8] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, "The era of 1-bit llms: All large language models are in 1.58 bits," *arXiv preprint arXiv:2402.17764*, 2024.

[9] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park, "OWQ: Outlier-Aware Weight Quantization for Efficient Fine-Tuning and Inference of LLMs," *arXiv*, 2024.