PENSA: A Spatial Accelerator for BitNet b1.58 Inference on an HBM-equipped AMD Alveo FPGA

Anonymous Author(s)
Affiliation
Address
email

Abstract

Ternary large language models (tLLMs) replace general multiplication in dominant linear layers with sign-controlled accumulation, shifting hardware bottlenecks toward packed-weight delivery, reduction, and operator occupancy. PENSA is an end-to-end FPGA accelerator for BitNet b1.58 2B4T model on an AMD Alveo U50™. Two row-partitioned ternary matrix engines consume packed TQ1_0 weights from eight independent high-bandwidth memory (HBM) pseudo-channels, each processing 1,280 ternary weights per cycle through sign selection and integer reduction. The sequencer, attention, and vocabulary-projection units remain resident on the FPGA, while two-stream interleave scheduling overlaps sequences across the persistent pipeline to form the Penta-Unit. At 104.1 MHz, PENSA achieves 58.09 tok/s at context 128 and 24.60 tok/s at context 2048, corresponding to 2.14–3.14× the throughput of a same-model Xeon E-2356G baseline. The resulting system demonstrates end-to-end FPGA execution of 2B-scale BitNet topology without modifying its model structure.

1 Introduction

Ternary large language models (tLLMs) constrain linear-layer weights to $w \in \{-1, 0, +1\}$, replacing general matrix multiplication with conditional addition/subtraction while reducing weight-storage and memory-bandwidth requirements [1, 2]. Software implementations such as Bitnet.cpp specialize ternary inference through lookup-table (LUT) and Multiply/Add-based CPU kernels [3]. However, removing general multiplication does not make end-to-end inference uniformly compute-light: the limiting work shifts toward packed-weight delivery, reduction, operator occupancy, attention, and vocabulary projection.

FPGA accelerators explore complementary forms of ternary specialization. TeLLMe and TeLLMe v2 employ lookup-based ternary matrix multiplication [4, 5]. TENET combines sparse LUT computation with dynamic activation N:M sparsity, weight decompression, and sparse-attention dataflow [6], while TerEffic explores ternary compute–memory co–design [7]. Other designs employ sparse or LUT-oriented ternarydatapaths [8, 9, 10], while FPGA LLM accelerators more broadly exploit spatial pipelines and specialized memory organizations [11, 12, 13, 14]. However, end-to-end FPGA inference of BitNet b1.58 2B4T with direct packed-weight execution, high-bandwidth memory (HBM)-fed ternary engines, persistent heterogeneous operators, and inter-sequence overlap remains comparatively unexplored.

PENSA targets this integration point as an end-to-end platform for characterizing the consequences of native ternary execution for BitNet b1.58 2B4T. It consumes packed ternary weights from HBM, partitions linear projections across two persistent TGEMMs, and keeps autoregressive execution resident on the FPGA. The resulting measurements expose how bottlenecks shift from multiplication

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

toward data delivery and reduction at the ternary engines, persistent-unit occupancy at short contexts, and sequence-dependent attention as context length increases.

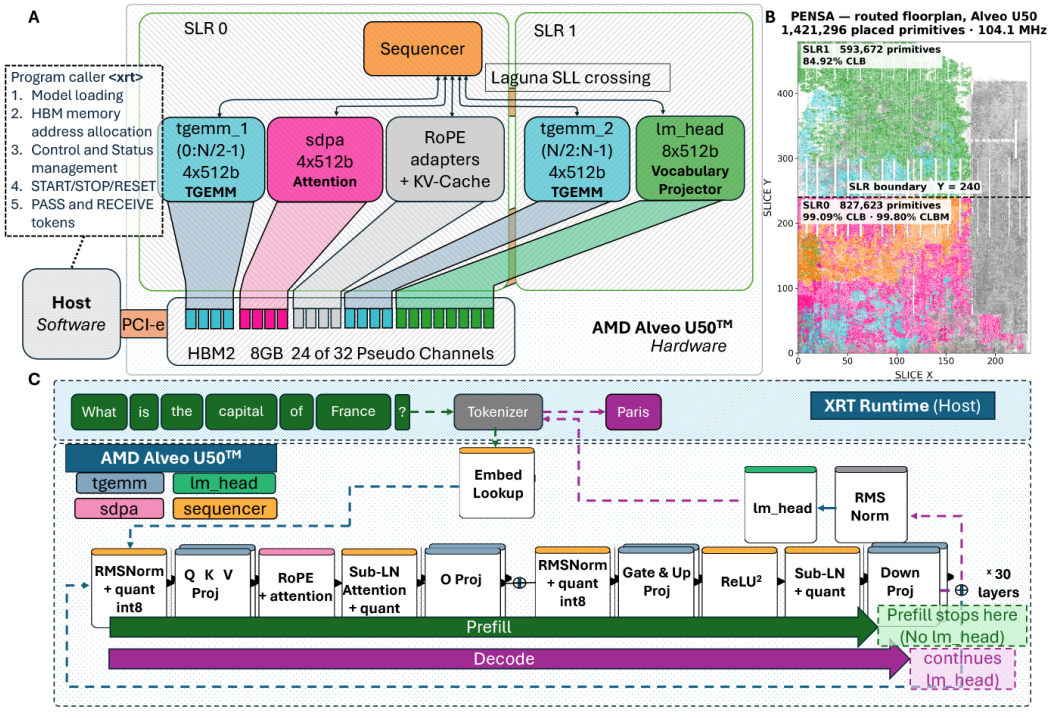

Figure 1: PENSA on the AMD Alveo U50™: (A) Penta-Unit with assigned HBM interfaces; (B) routed floorplan; and (C) resident BitNet b1.58 2B4T token-execution dataflow.

2 PENSA Architecture

2.1 Penta-Unit Spatial Accelerator Dataflow

Figure 1A shows the Penta-Unit, PENSA's five persistent compute units: the `Sequencer`, two ternary matrix engines (`tgemm_1` in `SLR 0` and `tgemm_2` in `SLR 1`), the attention engine (`sdpa`), and the vocabulary projector (`lm_head`). After loading the bitstream and model into HBM and starting the units, the `Sequencer` controls prefill and decode autonomously. Independently assigned HBM pseudo-channels supply the dominant data streams. The routed AMD Alveo U50™ floorplan is shown in Figure 1B.

Figure 1C shows the resident execution path. The `Sequencer` performs `Embed Lookup`, normalization, and activation quantization across all 30 transformer layers. Each layer's dual TGEMMs execute the seven ternary projections (Q, K, V, O, Gate, Up, and Down). `sdpa` applies RoPE, updates the HBM-resident K/V cache, and evaluates grouped-query attention using streaming softmax, without materializing the $T \times T$ score matrix. After the O and Gate/Up/Down paths, ReLU^2, and residual updates, final normalization feeds the BF16 `lm_head` with fused on-chip argmax. Only the selected token ID returns to the `Sequencer`, keeping execution and autoregressive feedback on the FPGA.

2.2 Dual Ternary Matrix Engines

Figure 2 shows the dual-TGEMM organization. The `Sequencer` quantizes each normalized activation to INT8 once and broadcasts the resulting vector $\mathbf{x}_q$, together with its scale, to both persistent TGEMMs. For $\mathbf{W} \in \{-1, 0, +1\}^{N \times K}$, ternary multiplication reduces to sign-controlled selection:

$$y_i = \sum_{j:W_{ij}=+1} x_{q,j} - \sum_{j:W_{ij}=-1} x_{q,j}. \tag{1}$$

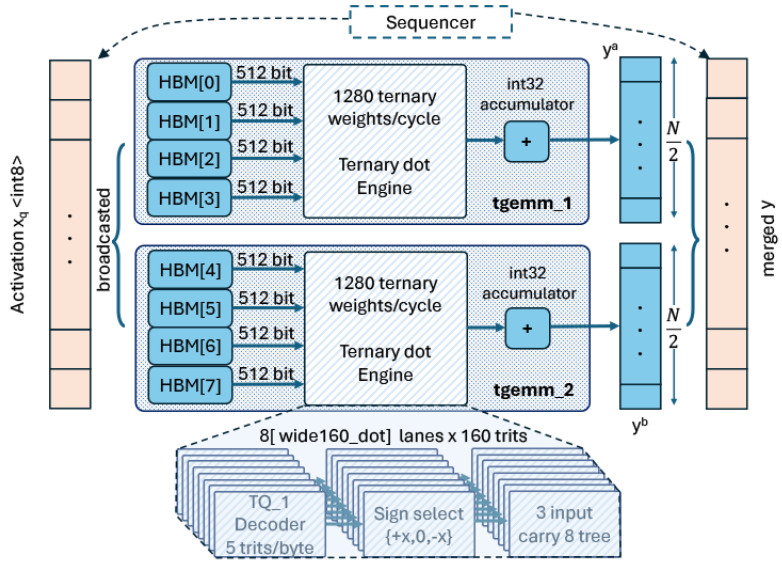

Figure 2: Dual-TGEMM datapath. A shared INT8 activation is broadcast to two row-partitioned engines, each supplied by four HBM pseudo-channels. Eight `wide160_dot` lanes per TGEMM process 1,280 packed ternary weights per cycle.

Thus, the integer dot-product datapath requires no general multiplier or DSP-based MAC. Output rows are divided evenly between the engines,

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}^{(a)} \\ \mathbf{W}^{(b)} \end{bmatrix}, \quad \mathbf{y}^{(a)} = \mathbf{W}^{(a)}\mathbf{x}_q, \quad \mathbf{y}^{(b)} = \mathbf{W}^{(b)}\mathbf{x}_q, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}^{(a)} \\ \mathbf{y}^{(b)} \end{bmatrix}, \quad \mathbf{W}^{(a,b)} \in \{-1, 0, +1\}^{(N/2)\times K}. \tag{2}$$

`tgemm_1` consumes weight stripes from HBM pseudo-channels 0–3 and `tgemm_2` from 4–7; the `Sequencer` merges their complementary output rows. Both engines therefore cooperate on the same projection rather than processing independent sequences.

Direct packed-weight datapath. Weights remain in the model's TQ1_0 base-3 representation, which packs five ternary values per byte, corresponding to a 1.6-bit/weight packed payload [3]. Each 256-bit HBM word contains 32 bytes, or 160 ternary weights. A compile-time 256-entry decode ROM maps each byte to five ternary controls, which directly select 0, $+x_{q,j}$, or $-x_{q,j}$; no activation-dependent lookup table or multiplier is required.

Each `wide160_dot` pads the 160 selected INT8 terms to 162 and reduces them via the width-aware three-input carry tree $162_{8b} \to 54_{11b} \to 18_{13b} \to 6_{15b} \to 1_{17b}$, followed by INT32 accumulation. The pipeline runs II=1 with five logical reduction levels, versus eight for binary reduction. Eight `wide160_dot` lanes per TGEMM consume 1,280 ternary weights per cycle. Against linear reduction, the carry-oriented tree is bit-exact while reducing LUT and FF use by 12% and 76%, respectively.

Two-stream interleaving. The TGEMMs are stateless projection servers: both always cooperate on one slot and projection, while the `Sequencer` temporally interleaves two independent sequence slots across the persistent pipeline. For example, while `lm_head` processes one slot, the other can advance through transformer layers using both TGEMMs. Each slot retains independent token and K/V-cache state, improving cross-unit occupancy without changing the model computation.

3 Evaluation

Figure 3 compares decode throughput across context lengths. At the achieved post-route frequency of 104.1 MHz, limited by the platform HBM/SLR read-response path rather than a PENSA compute unit (Appendix A), two-stream interleaving achieves 58.09 tok/s at context 128 and 24.60 tok/s at 2048, yielding 2.14–3.14× the throughput of the same-model Xeon E-2356G baseline. Interleaving improves aggregate throughput by 1.37–1.71× over a single-stream execution through persistent unit

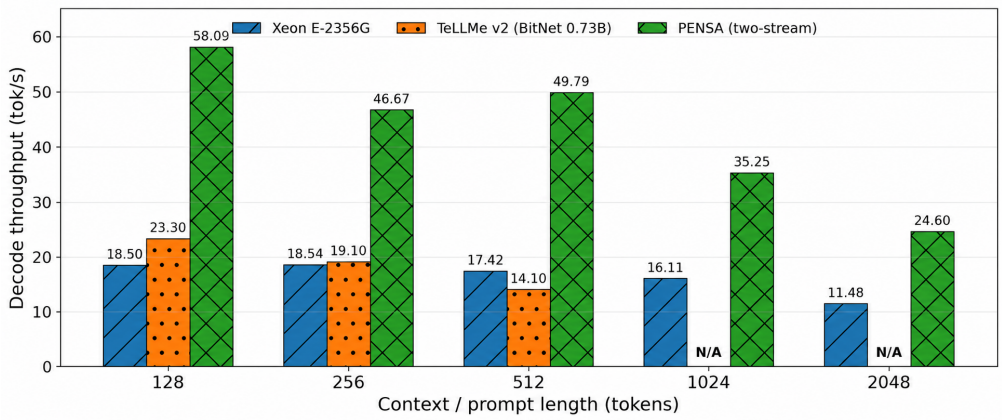

Figure 3: Decode throughput of PENSA, the same-model Xeon E-2356G baseline, and TeLLMe v2 [5]. PENSA and Xeon evaluate BitNet b1.58 2B4T at decode contexts 128–2048. TeLLMe v2 evaluates on BitNet 0.73B at [prompt,generation] configurations [128,128], [256,128], and [512,128], with longer lengths unreported.

overlap. However, it does not amortize the `lm_head` weight stream across sequences. Table 1 provides cross-model architectural positioning. While TerEffic[5] has a higher reported throughput, it reflects both cross-platform/model differences and compute–memory co-design that reduces weight traffic. PENSA instead targets the unmodified BitNet b1.58 2B4T topology, including its 328.3M-parameter BF16 vocabulary projection that moves approximately 656.6 MB per decoded token versus 416.8 MB of packed ternary weights. This projection brings the traffic floor of 12.3 and 7.8 ms, respectively (Appendix A). Prefill increases from 6.55 s at 128 tokens to 109.18 s at 2048 tokens (see Appendix D).

At context 2048, the context-dependent term contributes approximately 60% of modeled decode latency, with attention traversal accounting for 97.3% of its fitted per-key component (Appendix D). A 2,400-token two-stream run at 46.18 tok/s draws 33.46 ± 0.24 W corresponding to 0.725 J/token, or 0.405 J/token above the 14.75 W idle baseline (Appendix E). Table 2 reports FPGA model-fidelity results, which differ from the published task accuracies by 0.75–1.87 percentage points and PPL by only 0.07. Evaluation settings, uncertainty definitions, and the teacher-forced scoring path are given in Appendix F and G.

Table 1: Architectural positioning of ternary-LLM FPGA accelerators; cross-model throughput is not directly comparable.

Design	Model	FPGA	Reported tok/s	Distinguishing feature
TeLLMe [4]	BitNet 0.7B	KV260	9.51	Lookup-based TMM
TeLLMe v2 [5]	BitNet 0.73B	KV260	14.1–23.3	TLMM, on-chip weight buffering
TerEffic [7]	Ternary 2.7B	Alveo U280	521	HBM-assisted compute–memory co-design
TENET [6]	Ternary LLM	Stratix 10 MX	–	N:M sparsity, weight decompression
PENSA	BitNet 2B4T	Alveo U50	24.60–58.09	Direct TQ1_0, resident execution

Table 2: Model fidelity vs. published BitNet b1.58 2B [2]; accuracy differences are pp, PPL absolute.

	ARC-E	ARC-C	MMLU	GSM8K	PPL (WikiText-2)
Software Reference	74.79	49.91	53.17	58.38	14.59
PENSA	73.57±.90	48.04±1.46	52.39±.69	57.63±0.82	14.52
Δ	-1.22	-1.87	-0.78	-0.75	-0.07

4 Conclusion

PENSA demonstrates end-to-end BitNet b1.58 2B4T inference on an AMD Alveo U50™ without modifying the model topology. At 104.1 MHz, two-stream interleaving reaches 58.09 tok/s and exceeds the same-model Xeon baseline across all evaluated contexts, while preserving the model fidelity. Native ternary arithmetic shifts the limiting factors from multiplication toward packed-weight supply, reduction, persistent-unit occupancy, attention, and vocabulary projection.

References

[1] Shuming Ma et al. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

[2] Shuming Ma et al. Bitnet b1.58 2b4t technical report. *arXiv preprint arXiv:2504.12285*, 2025.

[3] Jinheng Wang, Hansong Zhou, Ting Song, Shijie Cao, Yan Xia, Ting Cao, Jianyu Wei, Shuming Ma, Hongyu Wang, and Furu Wei. Bitnet.cpp: Efficient edge inference for ternary llms. *arXiv preprint arXiv:2502.11880*, 2025.

[4] Y. Qiao, Z. Chen, Y. Zhang, Y. Wang, and S. Huang. TeLLMe: An energy-efficient ternary LLM accelerator for prefilling and decoding on edge FPGAs. *arXiv preprint arXiv:2504.16266*, 2025.

[5] Y. Qiao et al. TeLLMe v2: An efficient end-to-end ternary LLM prefill and decode accelerator with table-lookup matmul on edge FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 247–257, 2026.

[6] Z. Huang et al. TENET: An efficient sparsity-aware LUT-centric architecture for ternary LLM inference on edge. *arXiv preprint arXiv:2509.13765*, 2025.

[7] C. Yin et al. TerEffic: Highly efficient ternary LLM inference on FPGA. *arXiv preprint arXiv:2502.16473*, 2025.

[8] S. Zhu, G. Fu, M. Kjoseva, and G. Alonso. Efficient addition-based sparse GEMM for fast ternary large language model inference on edge devices. *ACM Transactions on Embedded Computing Systems*, 2026.

[9] R. Geens, J. Heldens, J. Dumoulin, and M. Verhelst. Hardware generation and exploration of lookup table-based accelerators for 1.58-bit LLM inference. In *Proceedings of ISPASS*, 2026. arXiv:2604.25183.

[10] A. Pittet, S. Zhu, V. Verdan, and G. Alonso. SSR: Sparse segment reduction for ternary GEMM acceleration. In *Proceedings of DATE*, 2026.

[11] H. Chen et al. Understanding the potential of FPGA-based spatial acceleration for large language model inference. *ACM Transactions on Reconfigurable Technology and Systems*, 18(1):1–29, 2025.

[12] J. Zheng and G. Chen. LoopLynx: A scalable dataflow architecture for efficient LLM inference. *arXiv preprint arXiv:2504.09561*, 2025.

[13] J. Zhuang, Z. Yang, S. Ji, H. Huang, A. K. Jones, J. Hu, Y. Shi, and P. Zhou. SSR: Spatial sequential hybrid architecture for latency throughput tradeoff in transformer acceleration. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 55–66, 2024.

[14] S. Zeng et al. FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 223–234, 2024. arXiv:2401.03868.

A Model and HBM Mapping

BitNet b1.58 2B4T configuration used by PENSA contains 30 transformer layers with hidden dimension $C = 2560$, feed-forward dimension $C_{\text{ff}} = 6912$, grouped-query K/V dimension $C_{\text{kv}} = 640$, and vocabulary size $V = 128256$. Each layer contains seven ternary projections: Q, K, V, O, Gate, Up, and Down. Across the transformer stack, the TGEMMs process 2.084B ternary linear weights, of which 1.593B (76.4%) belong to the feed-forward projections and 0.492B (23.6%) to the attention projections. The separate BF16 vocabulary projection contains 328.3M parameters.

Table 3 summarizes the principal HBM ownership. Dominant weight streams use independent pseudo-channels to avoid arbitration between concurrent consumers.

Table 3: Principal HBM ownership in PENSA.

Unit	HBM PCs	Contents
TGEMM A	0–3	packed TQ1_0 weights
TGEMM B	4–7	packed TQ1_0 weights
`lm_head`	8–15	BF16 vocabulary weights
Sequencer	19–24	model data, layout, token state
`sdpa`	26–29	K/V cache and RoPE tables

Twenty-six of the 32 pseudo-channels carry data in the primary mapping. The remaining channels are unused or correspond to NULL-bound interfaces sharing existing bundles. The 2.084B ternary weights correspond to approximately 416.8 MB of packed payload per transformer pass, whereas the 328.3M-parameter BF16 vocabulary projection requires approximately 656.6 MB per decoded token. At 104.1 MHz, eight 512-bit interfaces provide a nominal aggregate issue rate of 53.3 GB/s, giving payload-only traffic floors of approximately 7.8 ms for the ternary weights and 12.3 ms for the vocabulary projection. Thus, despite its smaller parameter count, the unquantized vocabulary projection has the larger per-token weight stream and constitutes an important optimization target. The measured pseudo-channel ceiling is 13.1 GB/s. The 104.1 MHz operating point is the achieved post-route frequency. The critical path in the platform HBM infrastructure (`hmss_0`) is the AXI read-response path crossing an SLR boundary through Laguna registers, rather than in a PENSA compute unit. Consequently, weight traffic, request rate, recurrence, consumer occupancy, and implementation timing can limit execution before the physical HBM-channel ceiling is reached.

B Host Execution and Reproducibility

PENSA uses a resident hardware–software execution model, following the broader FPGA–host co-design principle of separating software orchestration from hardware-resident dataflow. The host loads the bitstream and model, allocates HBM buffers, configures the five persistent compute units, and starts each unit once. Prefill and autoregressive decode then execute on device; the host may poll a progress counter for reporting but performs no per-token computation or kernel launch. Generated token IDs are transferred to the host only after execution, and no AXI4-Stream traffic leaves the device during generation. Tokenization and final detokenization remain CPU-side operations.

The implementation uses AMD Vitis 2025.2 and the corresponding Xilinx Runtime (XRT). XRT performs device configuration, HBM buffer allocation and binding, kernel argument setup, persistent kernel launch, status polling, and final token ID retrieval. Model and auxiliary buffers are allocated before kernel start; the final normalization scale is aligned to the 32-element boundary required by the kernel's wide memory accesses.

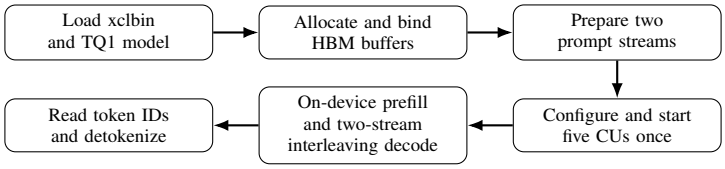

Figure 4: Host execution flow. Kernel launch occurs once; prefill, decode, and autoregressive token feedback remain resident on the FPGA.

For deterministic reference comparisons, greedy decoding is used, and the squared-ReLU feed-forward activation is enabled. Two-stream interleaving uses two equal-length prompts because both sequence slots advance under a shared token-loop position while retaining independent token and K/V state.

Generations are checked for prefix consistency across increasing generation lengths, and two-stream interleaving is exercised with different prompts to verify independent sequence state. The host executable and xclbin are treated as a matched pair because the kernel argument layout is fixed by the compiled design.

C TGEMM Microarchitecture

TQ1_0 stores five ternary weights per byte, yielding a 1.6-bit/weight packed payload. A 256-bit memory word therefore contains 32 encoded bytes representing 160 ternary weights. Each byte addresses a compile-time 256-entry decode ROM that produces five ternary controls; the controls select 0, $+x_{q,j}$, or $-x_{q,j}$ directly from the INT8 activation vector. The accompanying FP32 activation scale is applied outside the integer sign-selection datapath.

Each `wide160_dot` pads its 160 selected INT8 terms to 162 via the width-aware three-input reduction

$$162_{8b} \rightarrow 54_{11b} \rightarrow 18_{13b} \rightarrow 6_{15b} \rightarrow 1_{17b},$$

followed by INT32 accumulation. This organization has five logical reduction levels, compared with eight levels for binary reduction, and sustains II=1. Eight `wide160_dot` lanes operate in each TGEMM, corresponding to 1,280 ternary weights per cycle per engine. TGEMM A and TGEMM B use four independent HBM pseudo-channels each and process complementary output-row partitions of the same projection. Post-route implementation uses 509,234 LUTs (58.5%), 706,614 FFs (40.6%), 931.5 BRAMs (69.3%), 274 URAMs (42.8%), and 2,714 DSPs (45.6%) of the U50. The DSP total is whole-design utilization and includes non-TGEMM operators; the ternary dot datapaths themselves use no DSP-based multiplication. Relative to the preceding linear reduction, the carry-oriented tree reduces LUT and FF use by 12% and 76%, respectively.

Table 4: Post-route resource utilization of the persistent kernels.

Kernel	LUT	FF	BRAM	URAM	DSP
`Sequencer`	68,592	79,221	111	28	216
`tgemm_1`	29,028	26,069	180	82	27
`tgemm_2`	28,983	26,070	180	82	27
`sdpa`	104,023	145,193	122	60	1,111
`lm_head`	120,791	176,039	132	8	1,329
Total (kernels)	351,417	452,592	725	260	2,710

The two TGEMMs have nearly identical utilization, consistent with their symmetric row partitioning. Together they account for only 54 of the 2,710 kernel-level DSPs; the ternary dot datapaths themselves use no DSP-based multiplication, while the reported TGEMM DSPs arise from surrounding non-ternary arithmetic. In contrast, `sdpa` and `lm_head` account for 2,440 DSPs (90.0% of the kernel total), illustrating that native ternary execution shifts arithmetic resource pressure toward the remaining mixed-precision operators. Kernel totals exclude platform, HBM, and interconnect infrastructure and therefore do not equal the whole-device utilization.

D Additional Performance Results

Table 5 reports single-stream and two-stream interleaving decode throughput at 104.1 MHz. Two-stream interleaving raises aggregate throughput by 1.37–1.71× across the evaluated contexts by overlapping independent sequence slots within the persistent units; both TGEMMs continue to cooperate on one projection at a time.

Table 5: Single-stream and two-stream interleaving decode throughput at 104.1 MHz.

Context	Single (tok/s)	Two-stream (tok/s)	Gain
128	37.15	58.09	1.56×
256	33.49	46.67	1.39×
512	29.07	49.79	1.71×
1024	25.64	35.25	1.37×
2048	17.92	24.60	1.37×

Measured prefill times at sequence lengths 128, 256, 512, 1024, and 2048 are 6.55, 7.80, 21.30, 42.42, and 109.18 s, respectively. The measured two-stream decode latency is approximated by

$$t_{\mathrm{decode}}(T) \simeq 16.15\ \mathrm{ms} + (0.01186\ \mathrm{ms/key})T. \tag{3}$$

where T is the active context length. The fit gives $R^2 = 0.972$, RMSE 1.40 ms, and a maximum residual of 2.25 ms (10.5%) at $T = 256$. Measured throughput is locally non-monotonic between contexts 256 and 512. The present measurements do not isolate this deviation, so the linear model is used only to characterize the overall context-dependent trend. At $T = 2048$, the context-dependent component contributes approximately 60% of modeled latency. Operator-level timing attributes approximately 97.3% of the fitted per-key term to attention traversal.

E Power Measurement

Power is sampled using `xrt-smi` JSON telemetry during a 2,400-token two-stream decode run completing in 51.97 s (46.18 tok/s aggregate). Table 6 reports idle and active rail power together with dynamic power, defined as active minus idle. Energy per token is computed from the corresponding power divided by measured aggregate throughput.

Table 6: Power and energy during two-stream decode.

Rail	Idle (W)	Active (W)	Dynamic (W)	Energy/token
Board	14.75 ± 0.02	33.46 ± 0.24	18.70	0.725 active / 0.405 dynamic
VCCINT	6.37 ± 0.03	15.51 ± 0.74	9.14	0.336 active / 0.198 dynamic
VCCINT_IO	0.44 ± 0.02	3.73 ± 0.06	3.29	0.081 active / 0.071 dynamic

The VCCINT rail contributes 15.51 W active and 9.14 W above idle, corresponding to 0.336 and 0.198 J/token, respectively. These values represent FPGA-core rail consumption rather than TGEMM-only power. VCCINT also exhibits greater variation than board-level power, consistent with phase-dependent activity within the FPGA fabric.

F Evaluation Methodology

FPGA likelihood evaluation uses a teacher-forced scoring mode in which the host supplies the token sequence rather than feeding back the on-chip argmax. At a host-selected position, `lm_head` returns logits for requested token indices together with sufficient statistics for each of the eight vocabulary partitions,

$$m_p = \max_i z_{p,i}, \qquad s_p = \sum_i \exp(z_{p,i} - m_p).$$

The host reconstructs the full-vocabulary normalization as

$$M = \max_p m_p, \qquad \log Z = M + \log \sum_p s_p e^{m_p - M}, \qquad \log P(t) = z_t - \log Z.$$

Approximately 80 B are returned per scored position, avoiding transfer of the full vocabulary distribution.

A custom EleutherAI `lm-evaluation-harness` v0.4.12 adapter uses this scoring path for likelihood-based tasks and the native autoregressive path for generation. MMLU uses `cais/mmlu` version 1.0 with the test split, five-shot examples from the dev split, the deterministic `first_n` sampler, and accuracy as the metric. The same five dev examples are therefore shared by all test items within a subject and their K/V state is reused. Because the four MMLU alternatives are single-token choices, their logits are obtained from one scored position.

GSM8K uses `generate_until` with greedy decoding (`do_sample=false`, temperature 0), the benchmark stop strings, and the `flexible-extract` filter. The reported FPGA result uses four-shot prompting and achieves 0.5763 ± 0.82 exact-match accuracy. Reported $\pm$ values are harness-reported standard errors across evaluation items within a single evaluation pass, rather than variation across repeated runs. ARC-Easy and ARC-Challenge use

`lm-evaluation-harness` v0.4.12 task version 1.0 in the 0-shot setting with `multiple_choice` output. Each candidate's full answer text is scored as a continuation of "Question: {question}\n Answer:", and the highest-likelihood candidate is selected. Both `acc` and byte-length-normalized `acc_norm` are recorded; comparisons with published results use the same shot count and metric variant.

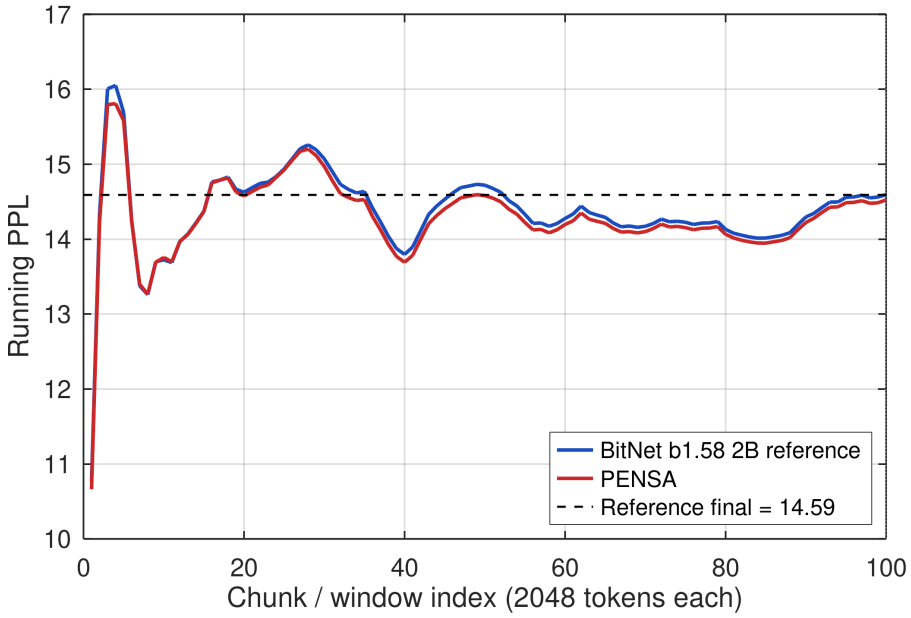

Figure 5: Running perplexity of PENSA (red) and BitNet b1.58 2B reference (blue) as a function of chunk index. The reference PPL value reported in Table 2 is indicated by the dashed line.

G Perplexity Computation

Perplexity (PPL) is a standard intrinsic measure of LLM quality that evaluates the model's ability to predict every token in a text corpus. For a sequence of N tokens, $x_1, \ldots, x_N$, an autoregressive model assigns each token the conditional probability $P(x_i \mid x_1, \ldots, x_{i-1})$. Perplexity is the exponential of the average negative log-likelihood:

$$\mathrm{PPL} = \exp\left[-\frac{1}{N-1}\sum_{i=2}^{N}\log P(x_i \mid x_1, \ldots, x_{i-1})\right]. \tag{4}$$

Lower PPL indicates more accurate prediction of the observed sequence; a perfect model would have $\mathrm{PPL} = 1$.

Each conditional $\log P(x_i \mid \cdot)$ is obtained via the same teacher-forced scoring path described in Appendix F: the host pre-fills the on-chip context with ground-truth tokens rather than feeding back the on-chip argmax, and `lm_head` returns the per-partition statistics (m_p, s_p) from which $\log P(x_t) = z_t - \log Z$ is reconstructed on the host. Scoring an entire context window in a single forward pass, rather than one pass per token, keeps this cost linear in N rather than quadratic, at the same approximately 80 B per scored position as the likelihood tasks in Appendix F.

Because on-chip state is reset at each context-window boundary, a chunked evaluation conditions each prediction on only the tokens within its own window, not the full corpus prefix. For a corpus of N tokens partitioned into $K = \lceil N/W \rceil$ windows of length W (the final window may be shorter), with window k spanning tokens $x_{b_k+1}, \ldots, x_{e_k}$ for $b_k = (k-1)W$ and $e_k = \min(kW, N)$, the first token of each window has no in-window context and is excluded from scoring, giving

$$\mathrm{PPL} = \exp\left[-\frac{1}{N-K}\sum_{k=1}^{K}\sum_{j=b_k+2}^{e_k}\log P\big(x_j \mid x_{b_k+1}, \ldots, x_{j-1}\big)\right], \tag{5}$$

which reduces to the single-sequence expression above when $K = 1$ ($W \geq N$). Equivalently, log-likelihoods and scored-token counts are accumulated across all windows before the single final exponentiation (not averaged per-window) since the latter is not equivalent for windows of unequal length and is sensitive to how loss is weighted across them.

PPL is sensitive to the choice of context window W and stride $S \leq W$: small windows inflate PPL because every token near a window's start is scored with truncated context, and this effect is indepen-

dent of any model or hardware. In the results reported here, both the reference model and PENSA were evaluated on the entire WikiText-2 test corpus with $W = S = 2048$, i.e. non-overlapping windows at the model's full context length, matching `llama.cpp`'s default (`-ppl-stride` unset) chunking behavior ($\texttt{start} = i \cdot W$, $\texttt{end} = \texttt{start} + W$, cache cleared between chunks).

The customized PPL procedure enables comparison of per-chunk PPL dynamics against the reference. Over 100 representative measured chunks (Figure 5), PENSA (red) closely tracks the published BitNet b1.58 2B reference (blue) (MAE = 0.0771, RMSE = 0.0878, Pearson $r = 0.997$, mean absolute relative difference = 0.53%), with a maximum single-chunk discrepancy of 1.51%. Running PPL oscillates sharply over the first 40 chunks before settling into a narrow band around its final value of approximately 14.59. This behavior reflects the cumulative-average nature of PPL and the heterogeneous composition of the WikiText-2 test file, which concatenates distinct Wikipedia articles rather than forming a statistically uniform text stream. As the evaluation window moves between articles, per-chunk PPL therefore rises and falls with the underlying content rather than merely reflecting noise. The near-identical agreement between PENSA's PPL curve and BitNet b1.58 2B reference, particularly through this oscillatory region, provides evidence that these variations are reproducible properties of the text and model rather than implementation artifacts.

- **Claims**

  Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

  Answer: [Yes]

  Justification: The abstract and Introduction state the scope of PENSA as an end-to-end FPGA implementation of BitNet b1.58 2B4T and distinguish same-model CPU comparisons from cross-model architectural comparisons. The reported throughput, fidelity, and bottleneck observations are supported by the Evaluation and appendices.

- **Limitations**

  Question: Does the paper discuss the limitations of the work performed by the authors?

  Answer: [Yes]

  Justification: The Evaluation and Appendix D report the strong context dependence of decode latency and the substantially increasing prefill latency. Cross-model accelerator results are explicitly presented as architectural context rather than direct performance comparisons.

- **Theory assumptions and proofs**

  Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

  Answer: [N/A]

  Justification: The paper presents an accelerator architecture and empirical evaluation and does not claim new theoretical results requiring formal proofs.

- **Experimental result reproducibility**

  Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

  Answer: [Yes]

  Justification: Sections 2–3 and Appendices A–F describe the model dimensions, ternary representation, TGEMM organization, HBM mapping, host execution, implementation resources, evaluation methodology, and performance measurements needed to reproduce or independently verify the reported results.

- **Open access to data and code**

  Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

  Answer: [No]

Justification: The paper provides implementation and reproducibility details, but the complete FPGA implementation and experiment scripts are not currently released as an anonymized open-access artifact.

- **Experimental setting/details**

  Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

  Answer: [Yes]

  Justification: This work evaluates inference rather than training. The model, hardware configuration, operating frequency, context lengths, execution modes, CPU baseline, and evaluation methodology are specified in the Evaluation and appendices.

- **Experiment statistical significance**

  Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

  Answer: [Yes]

  Justification: Table 2 reports uncertainty for the downstream evaluation tasks, with its definition and evaluation procedure given in Appendix E. Appendix D additionally reports R^2, RMSE, and the maximum residual for the decode-latency fit.

- **Claims**

  Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

  Answer: [Yes]

  Justification: The abstract and Introduction state the scope of PENSA as an end-to-end FPGA implementation of BitNet b1.58 2B4T and distinguish same-model CPU comparisons from cross-model architectural comparisons. The reported throughput, fidelity, and bottleneck observations are supported by the Evaluation and appendices.

- **Limitations**

  Question: Does the paper discuss the limitations of the work performed by the authors?

  Answer: [Yes]

  Justification: The Evaluation and Appendix D report the strong context dependence of decode latency and the substantially increasing prefill latency. Cross-model accelerator results are explicitly presented as architectural context rather than direct performance comparisons.

- **Theory assumptions and proofs**

  Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

  Answer: [N/A]

  Justification: The paper presents an accelerator architecture and empirical evaluation and does not claim new theoretical results requiring formal proofs.

- **Experimental result reproducibility**

  Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

  Answer: [Yes]

  Justification: Sections 2–3 and Appendices A–E describe the model dimensions, ternary representation, TGEMM organization, HBM mapping, host execution, implementation resources, evaluation methodology, and performance measurements needed to reproduce or independently verify the reported results.

- **Open access to data and code**

  Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The paper provides implementation and reproducibility details, but the complete FPGA implementation and experiment scripts are not currently released as an anonymized open-access artifact.

- **Experimental setting/details**

  Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

  Answer: [Yes]

  Justification: This work evaluates inference rather than training. The model, hardware configuration, operating frequency, context lengths, execution modes, CPU baseline, and evaluation methodology are specified in the Evaluation and appendices.

- **Experiment statistical significance**

  Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

  Answer: [Yes]

  Justification: Table 2 reports uncertainty for the downstream evaluation tasks, with its definition and evaluation procedure given in Appendix E. Appendix D additionally reports R^2, RMSE, and the maximum residual for the decode-latency fit.

- **New assets**

  Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

  Answer: [N/A]

  Justification: The submission does not release a new dataset, pretrained model, or public software artifact.

- **Crowdsourcing and research with human subjects**

  Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

  Answer: [N/A]

  Justification: The research does not involve crowdsourcing or human subjects.

- **Institutional review board (IRB) approvals or equivalent for research with human subjects**

  Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

  Answer: [N/A]

  Justification: The research does not involve human subjects, so IRB or equivalent ethics approval is not applicable.

- **Declaration of LLM usage**

  Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

  Answer: [N/A]

  Justification: LLMs are not used as an original or non-standard component of the research methodology. BitNet b1.58 2B4T is the inference workload evaluated by the proposed accelerator rather than a tool used to develop the research method.