UNIVERSITY OF CALIFORNIA,
IRVINE

A Novel Projection-Wise Quantization Method and A Custom Accelerator Design for Efficient Sub-4-Bit Large Language Model Inference

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Zhenyu Tang

Thesis Committee:
Professor Sitao Huang, Chair
Professor Salma Elmalaki
Professor Nader Bagherzadeh

2026

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Professor Sitao Huang, Yifan Zhang, Ye Qiao, Faraz Tahmasebi, and other fellow researchers in the UCI CORSA Lab for their professional help throughout the development of my Master's Thesis. I could not have chosen a relevant and feasible research topic, obtained the results, and completed this Master's Thesis without them. I would also like to thank Professor Nader Bagherzadeh and Professor Salma Elmalaki for taking their effort to serve on my Thesis Committee and review my work.

I would like to thank my partner, Jing, for her support and understanding along the way.

I would like to thank the authors of TeLLMe v2 and QuaRot for allowing me to use their works as the basis for my design. My achievements stand on the shoulders of their great works.

ABSTRACT OF THE THESIS

A Novel Projection-Wise Quantization Method and A Custom Accelerator Design for Efficient Sub-4-Bit Large Language Model Inference

By

Zhenyu Tang

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2026

Professor Sitao Huang, Chair

Modern Large Language Models (LLMs) offer exceptional inference accuracy but remain severely restricted by high resource and power requirements. Post-Training Quantization (PTQ) mitigates these memory bottlenecks by compressing weights to sub-4-bit regimes; however, existing Hadamard rotation-based methods that enable nearly lossless 4-bit weight and activation quantization ($W4A4$) does not push weight quantization below INT4, thereby limiting the memory size reduction potential of Hadamard transform. Furthermore, many GPU-based mixed-precision approaches lack native hardware support for integer to floating point multiplication and custom low-bit multiplication kernels. This requires runtime upcasting of integer weights and fails to leverage the resource saving potentials of low-bit representations during computation, leaving custom FPGA accelerators as the most efficient platform for low-bit mixed precision execution. To address these problems, this paper presents a projection-wise quantization method and a custom FPGA accelerator design that supports this quantization method. The proposed quantization scheme for Llama 2-7B compresses weights to 3.56 bits and achieves 10.87% reduction in weight storage relative to W4A4 QuaRot with a minimal perplexity increase of 2% and obtains a better Compression-to-Degradation Ratio than the current state-of-the-art SliM-LLM. In addition, the mixed precision accelerator design with custom Table Lookup-based matrix multiplication ker-

nels for 3-bit weight operations achieves nearly 50% reduction in LUT usage compared to standard Multiply-and-Accumulate (MAC) units. Finally, to mitigate the additional latency overhead of online Hadamard operation, the proposed accelerator design incorporates a 2-stage pipelined online Hadamard transform unit that reduces the latency of attention projection Hadamard transformations by 13.7%.

Chapter 1

Introduction and Background

1.1 Introduction

Modern Large Language Models (LLMs) have achieved remarkable capabilities across diverse artificial intelligence tasks, yet their vast parameter counts demand immense computational resources and memory bandwidth, imposing severe energy and hardware burdens during deployment. To address these overheads, post-training quantization methods have been proposed, compressing model weights and activations into low-bit representations rather than relying on standard full-precision (FP16) formats.

To mitigate activation outliers, Hadamard rotation-based methods such as QuaRot [3] and SpinQuant [13] achieve effective 4-bit weight and activation compression with minimal perplexity degradation compared to FP16, though they remain limited to 4-bit representations. To push compression further, mixed precision strategies like Slim-LLM and ResQ selectively retain critical parameters at higher bit-widths to reach sub-4-bit average precision with impressive accuracy. However, modern GPUs lack native hardware kernel support for non-standard low-bit formats (e.g. INT3 or INT2), therefore unable to convert memory savings

into actual compute speedups. To overcome both the 4-bit compression limit of existing Hadamard rotation frameworks and the compute upcasting bottlenecks of GPU-targeted mixed-precision methods, this article proposes a custom hardware accelerator pipeline on FPGA that uses a novel 3.56-bit mix precision quantization method. The main contributions of this work are summarized as follows:

- **Projection-Wise Mixed-Precision Quantization:** This paper proposes a novel projection-wise mixed-precision quantization scheme that identifies quantization-resistant projections to apply aggressive 3-bit weight quantization safely. When evaluated on selected layers of Hadamard-transformed models, the proposed method method achieves a 12% overall model compression with only a 2.1% perplexity degradation.

- **Hardware-Efficient FPGA Implementation:** The proposed quantized mixed-precision LLaMA architecture is deployed on an AMD KV260 FPGA board. To maximize hardware efficiency, custom Table-Lookup matrix multiplication kernels are implemented for 3-bit weight operations, achieving a 50% reduction in LUT utilization compared to standard Multiply-Accumulate (MAC) units. Furthermore, a 2-stage online Hadamard transform is integrated into a pipelined dataflow architecture, reducing the online Hadamard transformation latency in attention projections by 13.7%.

1.2 Background

1.2.1 Low-bit quantization for LLM

To mitigate the substantial memory and computational overhead of Large Language Models (LLMs), recent works have focused on aggressive low-bit quantization. QuaRot [3] introduces a hardware-efficient framework that applies randomized Hadamard rotations to both weight and activation matrices. By spreading concentrated outlier values uniformly across all dimensions, the Hadamard transform neutralizes feature skewing without altering the

result of matrix multiplication, enabling W4A4 (4-bit weight, 4-bit activation) quantization on Llama-2 models with minimal accuracy degradation compared to the full-precision FP16 baseline.

SpinQuant introduces learnable orthogonal rotation matrices to capture finer data distributions on top of Hadamard transform. Instead of relying solely on static, random transformations, SpinQuant utilizes a calibration dataset to optimize the rotation matrices to directly minimize quantization error [13]. This data-driven refinement allows SpinQuant to narrow the gap to full-precision performance, achieving up to a 19.1 point perplexity improvement over standard LLM-QAT methods and recovering up to 98.7 percent of the FP16 accuracy on zero-shot downstream tasks. Notably, QuaRot does not push quantization bitwidth below standard INT4 [3], while SpinQuant with INT3 weights require pairing with 8-bit activations [13] to achieve acceptable model perplexity, effectively doubling the size of KV cache and matrix multiplication latency during compute intensive prefill stage. However, the foundational capabilities of the Hadamard transform suggest further bitwidth reductions beyond INT4 (e.g. INT3 + INT4 mixed precision) remain viable and present a promising path for future exploration.

1.2.2 Mixed Precision Quantization for LLMs

Another recent direction in the exploration of low-bit quantization is to allocate different bitwidths for different weight groups based on their architectural importance. The SliM-LLM [11] framework assigns bitwidths to weight groups based on a specialized *Salience Score*, a metric that computes the Hessian sensitivity and weight magnitude relative to the loss curvature. This targeted bitwidth allocation scheme successfully suppresses quantization noise where it is most destructive, yielding a 48% reduction in perplexity degradation compared to state-of-the-art gradient-free Post-Training Quantization (PTQ) methods. Furthermore, its gradient-optimized variant (SliM-LLM+) compresses the perplexity penalty by an additional

35.1%, delivering a near-lossless performance profile relative to uniform 4-bit benchmarks.

In contrast, ResQ [18] utilizes Principal Component Analysis (PCA) to identify a low-rank, high-variance subspace within the activations. It protects the corresponding critical weights by assigning them a higher precision of 8-bit, while applying invariant Hadamard matrix rotations within the remaining low-variance subspaces to aggressively suppress outliers in the 4-bit regions. By combining PCA-driven channel selection with rotation-based mitigation, ResQ delivers up to 33% lower Wikitext2 perplexity and up to a 5.4% improvement in zero-shot accuracy over SpinQuant, while achieving up to a 3× speedup over the full-precision 16-bit baseline. Despite these performance advantages, ResQ faces notable hardware and design limitations. A significant source of online overhead arises from the explicit projection applied to the query (Q) and key (K) tensors after the Rotary Position Embedding (RoPE) step: both Q and K must be multiplied online by a per-layer matrix composed of the PCA eigenvector basis and a learned block-diagonal rotation. This operation incurs an $\mathcal{O}(\text{seq\_len} \cdot d_{\text{head}}^2)$ per-layer computational cost that cannot be pre-fused into preceding weights due to the non-commutative nature of RoPE.

Layerwise Quantization

Layerwise Ultra-Low Bit Quantization (LUQ) [4] addresses non-uniform layer sensitivity in Multimodal Large Language Models (MLLMs) by wrapping the standard GPTQ base solver within an entropy-driven inter-layer selection framework. Intra-layer weight rounding is executed using second-order GPTQ error minimization, but the calibration set $\mathcal{D}_{\text{cal}}$ is expanded to include a mixture of visual and textual tokens to capture the higher statistical variance of multimodal activation distributions. To allocate layer bit-widths, LUQ discretizes each layer's output activations via K-means clustering and calculates their Shannon entropy $H(Y_l)$ as a proxy for functional complexity. Layers exhibiting low activation entropy are assigned to ultra-low precision (1-bit or 2-bit GPTQ), whereas high-entropy layers are preserved at 4-bit GPTQ precision. By tailoring bit allocation to functional complexity rather than en-

forcing uniform quantization, LUQ achieves a 40% and 31% reduction in memory footprint on LLaVA-1.5 and Qwen-2.5-VL, respectively, relative to uniform 4-bit baselines. Across nine Visual Question Answering (VQA) benchmarks, LUQ preserves multimodal reasoning capabilities, restricting performance degradation to less than 10% on the MME benchmark in the sub-4-bit regime.

LUQ should be considered an orthogonal strategy rather than a direct competitor to this work. While LUQ focuses on inter-layer bit allocation to match the accuracy of standard 4-bit AWQ and GPTQ baselines, this paper targets intra-layer outlier suppression via invariant Hadamard rotations to match the higher accuracy ceiling of QuaRot. Because LUQ's inter-layer entropy profiling functions independently of intra-layer rotation schemes, both techniques address distinct levels of the quantization design space and can be combined.

Mixed precision quantization methods designed for GPUs that use different bitwidths for weights and activations, such as SliM-LLM [11], face an important limitation. These methods benefit from the higher memory transfer rate of lower bitwidths, but they cannot perform low-bit matrix multiplications in the original precision. In other words, the benefits of low-bit data exist in memory transfer and storage, but not in compute. This limitation can be attributed to the fact that modern GPU architectures lack native hardware support for mixed precision multiplication. During computation, tensor cores inevitably dequantize the integer weights to match the bit-width of the activations [16]. Another limitation faced by all GPU-based quantization approaches is that, unlike FPGAs that can use custom matrix operation kernels (e.g. Table Lookup-based kernels) for low-bit weights to improve resource and latency efficiency, GPUs are restricted to fixed, MAC-based kernels [5, 17]. As a result, mixed precision quantization methods on GPU fail to leverage the low-bit representations for compute-side speedup.

Chapter 2

Proposed Mix-Precision Quantization Method

2.1 Overview

The quantization strategy can be summarized as follows: first, it identifies the weight columns associated with important activations, i.e. activations that are more sensitive to quantization loss. It then quantizes the important activations in selected projections (q, k, Gate_{up}, $\text{Gate}_{\text{proj}}$) of selected layers into INT4 using the GPTQ method, while all other weights are quantized into INT3. The weight and activation matrices undergo random matrix Hadamard transform proposed by QuaRot to suppress activation outliers[3]. The implementation of this quantization method is explained in greater detail below.

2.1.1 Identifying Important Weights for Higher Precision

To identify the weight groups associated with activation channels that are more sensitive to quantization loss, the proposed approach uses the Salience Score metric from Slim-LLM [11] on a sample of size 128 from the WikiText2 dataset [14] with a token length of 2048. The

salience score is computed via the equation below

$$S_{i,j} = \frac{w_{i,j}^2}{([H^{-1}]_{j,j})^2}, \tag{2.1}$$

The w_{i}, j in the equation measures the raw weight magnitude, and the inverse hessian $H^{(}-1)$ is the diagonal elements of the inverse Hessian matrix which approximates the sensitivity of the model loss to quantization-induced perturbations of each weight. The weight column groups with the top-12.5% salience score are quantized to a higher precision (INT4) during GPTQ, while the rest are quantized to INT3. The group size for salience score ranking is set at 128, matching the group size of GPTQ quantization.

2.1.2 Quantization Scale Calculation for INT3 and INT4 weights

During quantization, the quantization scales for each 3-bit and 4-bit regions are computed independently. The quantization equation is given as follows:

$$\text{Scale} = \frac{\max(W)}{2^{b-1} - 1} \tag{2.2}$$

Where $\max(W)$ denotes the largest value in the weight column group, and b denotes the bitwidth of the region. The weights are then quantized using GPTQ.

2.1.3 GPTQ Quantization

GPTQ [8] is a post-training quantization method that minimizes quantization error through Hessian-guided optimization. Specifically, it seeks a quantized weight matrix $\hat{W}$ that mini-

mizes the second-order approximation of the loss:

$$\Delta L \approx \frac{1}{2}(W - \hat{W})^T H (W - \hat{W}) \tag{2.3}$$

This method quantizes one column of a weight matrix at a time and uses the inverse Hessian H^{-1} to redistribute the quantization error to the remaining unquantized columns. After quantizing a column, the induced error is compensated according to:

$$w_F \leftarrow w_F - \frac{w_q - \hat{w}_q}{[H^{-1}]_{qq}} (H^{-1})_{F,q} \tag{2.4}$$

where q represents the index of the column currently being quantized, w_q is the original weight vector of that column, $\hat{w}_q = \text{quant}(w_q)$ is its quantized counterpart, F represents the set of remaining unquantized columns, and H^{-1} is the inverse Hessian matrix. By iteratively quantizing columns and propagating the resulting error to less sensitive weight directions, GPTQ substantially reduces reconstruction error compared to naive rounding-based quantization. QuaRot reports that applying GPTQ to Hadamard-rotated weights reduces model perplexity by up to 27% under INT4 weight quantization [3]. Therefore, the proposed approach employs the same GPTQ quantization method as QuaRot with quantization group size = 128 for both weight and activation and uses the resulting GPTQ-quantized QuaRot model as the baseline for perplexity comparison.

Matching the quantization group size with the group size of INT3 and INT4 weight columns allows GPTQ to compute independent quantization scales for each precision region. In contrast, an unstructured mixed-precision layout where GPTQ quantization groups are contiguous while INT3 and INT4 columns are interweaved would force columns of different bitwidths to share a common scale, causing the larger dynamic range of INT4 weights to dominate scale selection. As a result, INT3 weights would suffer from reduced quantization resolution and increased quantization error. For this reason, the proposed approach matches

the INT3 and INT4 column group sizes to the GPTQ group size, enabling each precision region to be quantized with its optimal scale.

2.1.4 Reducing Activation Outliers via Hadamard Transform

The raw activation matrices of transformer models contain large numbers of outliers that aggregate in certain input features. When the activations are quantized from FP16 to lower precision(e.g. INT4) using per-output feature quantization, the activation outliers tend to skew the quantization scale and introduce large quantization loss. To address this, Hadamard rotations transform the activation space by mathematically rotating the activation space, which dilutes the energy of isolated peaks across all features. This smooths out magnitude differences across the entire matrix, effectively eliminating outliers and shrinking the overall dynamic range. Following the QuaRot framework, the orthogonal matrix can be pre-multiplied directly into the static weight matrices offline. This eliminates online computation for most layers, requiring runtime Hadamard operations at only two points: $\text{Attn}_{\text{proj}}$ and $\text{Gate}_{\text{down}}$ [3]. The result is a uniformly distributed, low-magnitude activation space that can withstand low bit quantization with minimal accuracy loss.

The figures below show the activation matrices after Hadamard transform.

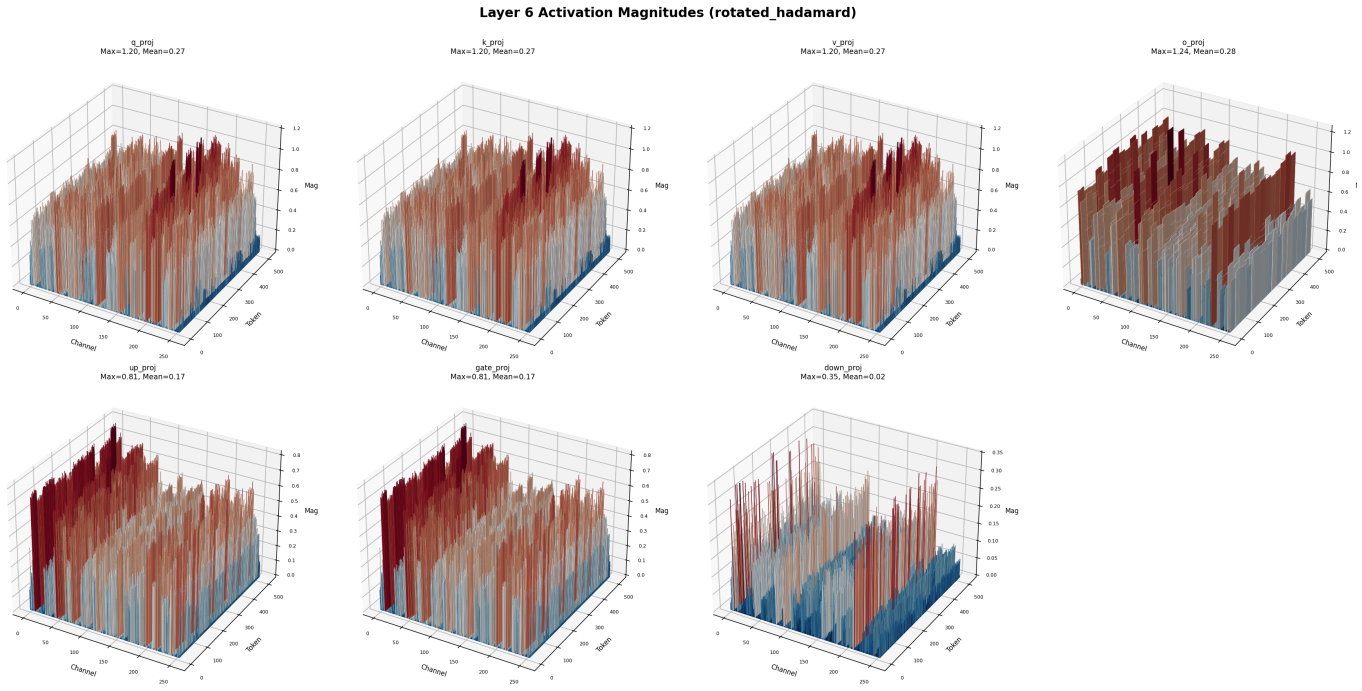

Figure 2.1: Activations Across All Projections in Layer 6 of LlaMA2-7B model After Hadamard Rotation

2.1.5 Projection-wise Mix Precision

On a Llama2 model that underwent QuaRot-style Hadamard transform, it was observed that different projections exhibit different sensitivity to 3-bit weight quantization. In the attention layer, further quantizing Q_{proj} and K_{proj} leads to smaller perplexity increase than quantizing V_{proj} and O_{proj}. In the MLP layer, applying 3-bit weight quantization onto Up_{proj} and $\text{Gate}_{\text{proj}}$ leads to smaller perplexity increase than Out_{proj}. The effect of different quantization projection choice on model perplexity is shown in Appendix A.1.

Attention Layer Projection Quantization Sensitivity

A likely reason for V_{proj} and O_{proj}'s heightened sensitivity to aggressive quantization is that, unlike Q_{proj} and K_{proj}, these two projections are computed much closer to the residual stream. The quantization errors of V_{proj} and O_{proj} do not undergo the operations that Q_{proj} and K_{proj} go through, such as QK^T and Softmax. The Softmax operation, in particular, reduces the

quantization error by mapping the attention logits QK^T to normalized probability scores using the formula below:

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Using the set $QK^T = [3, 2, 1]$ as an example, the quantized product $(QK^T)' = [3.1, 1.9, 1.0]$ has a quantization error of $[0.1, 0.1, 0]$. The quantization error of $\text{Softmax}(QK^T)$, on the other hand, is considerably smaller at $[0.01, 0.04, 0.01]$, given that $\text{Softmax}(QK^T) = [0.665, 0.245, 0.090]$ and $\text{Softmax}(QK^T)' = [0.685, 0.206, 0.109]$.

While the quantization errors in Q_{proj} and K_{proj} are smoothed by these operations, the error in V_{proj} and O_{proj} directly affects the residual stream. This effect is especially pronounced in O_{proj}, whose output is not multiplied with any other matrices at all before entering the residual stream, resulting in the greatest per-projection perplexity increase out of all attention layer projections.

FFN Layer Projection Quantization Sensitivity

Inside the FFN layer, it was observed that quantizing $\text{Down}_{\text{proj}}$ has a greater effect on model perplexity than quantizing Up_{proj} or $\text{Gate}_{\text{proj}}$. A plausible reason is that the FFN layer of Llama-style transformers perform element-wise operations between SwiGLU output of $\text{Gate}_{\text{proj}}$ and the output of Up_{proj}. The output of this element-wise operation is used as the input of the subsequent $\text{Down}_{\text{proj}}$ as shown in the equation below.

$$\text{FFN}(\mathbf{X}) = (\text{SiLU}(\mathbf{X}\mathbf{W}_{\text{gate}}) \odot (\mathbf{X}\mathbf{W}_{\text{up}}))\, \mathbf{W}_{\text{down}} \tag{2.5}$$

The elementwise operation can create large values that, when multiplied with $\mathbf{W}_{\text{down}}$, magnifies the perturbations on the latter. Specifically, because $\mathbf{h}_{\text{act}} = \text{SiLU}(\mathbf{X}\mathbf{W}_{\text{gate}}) \odot \mathbf{X}\mathbf{W}_{\text{up}}$ multiplies two intermediate activation vectors together, correlated features interact quadrat-

ically, producing significantly larger vector magnitudes and emergent outlier spikes in $\mathbf{h}_{\text{act}}$. Since the error introduced by weight quantization scales directly with the magnitude of incoming activations ($\Delta\mathbf{Y} \approx \mathbf{h}_{\text{act}}\Delta\mathbf{W}_{\text{down}}$), these inflated activation values act as a heavy error multiplier on $\mathbf{W}_{\text{down}}$. In contrast, $\mathbf{W}_{\text{gate}}$ and $\mathbf{W}_{\text{up}}$ operate directly on normalized inputs $\mathbf{X}$ from RMSNorm, where activations remain tightly bounded. Consequently, weight quantization noise in $\mathbf{W}_{\text{gate}}$ or $\mathbf{W}_{\text{up}}$ is scaled by much smaller input vectors, causing far less perturbation to the final output than quantizing $\mathbf{W}_{\text{down}}$. Based on these observations, the proposed approach applies the aforementioned column-wise mixed precision quantization method on Q_{proj} and K_{proj} of the attention layer, as well as Up_{proj} and $\text{Gate}_{\text{proj}}$ of the FFN layer, while keeping the weights of all other projections at INT4. This configuration achieves a favorable trade-off between model size reduction and perplexity degradation.

Study on the Effect of Projection Choice on Model Perplexity

This thesis introduces a new metric, named *Compression-to-Degradation Ratio* (CDR), to measure how much perplexity increase is required achieve the current model size reduction. A higher CDR is more ideal because it means that the given quantization method can reduce the model size with lower impact to the model's inference capability. The CDR for a given approach X compared to a State-of-the-Art baseline is calculated via the formula below:

$$\frac{(\text{Bits}_{\text{X}} \div \text{Bits}_{\text{baseline}})}{(\text{PPL}_{\text{X}} - \text{PPL}_{\text{baseline}}) \div \text{PPL}_{\text{baseline}}} \tag{2.6}$$

Table 4.3 shows that applying 12.5% INT4 + 87.5% INT3 mixed precision to V_{proj}, O_{proj}, $\text{Gate}_{\text{down}}$, and $\text{Gate}_{\text{proj}}$ has higher impact on model perplexity than the mixed precision projection choice proposed in this paper. Both configurations use mixed precision weights for layer 6-31 and GPTQ with quantization group size set to 128 and achieve the same amount of model bitwidth reduction. The perplexity of the W4A4 QuaRot baseline used for comparison in this table is 6.089.

Table 2.1: Perplexity Comparison of Mixed Precision Projection Choices

Method	**WikiText-2 PPL**	**% PPL Increase**	**CDR**
Quantizing Q_{proj}, K_{proj}, Gate_{up}, $\text{Gate}_{\text{proj}}$	6.214	2.05	5.3
Quantizing V_{proj}, O_{proj}, $\text{Gate}_{\text{proj}}$, $\text{Gate}_{\text{down}}$	6.319	3.78	2.88

Table 2.2 shows the perplexity increase and CDR of three quantization methods without using groupwise GPTQ for weight and activation quantization. GPTQ instead calculates a single quantization scale for each weight channel and each activation vector. The 12.5% INT4 + 87.5% INT3 quantization method is applied to layer 10-31. The perplexity of the W4A4 QuaRot baseline used for comparison in this table is 6.335.

Table 2.2: Perplexity Comparison of Mixed Precision Projection Choices (Without Groupwise Quantization)

Method	**WikiText-2 PPL**	**% PPL Increase**	**CDR**
Quantizing Q_{proj}, K_{proj}, Gate_{up}, $\text{Gate}_{\text{proj}}$	6.466	2.07	4.44
Quantizing V_{proj}, O_{proj}, $\text{Gate}_{\text{proj}}$, $\text{Gate}_{\text{down}}$	6.536	3.17	2.90
Quantizing V_{proj}, O_{proj}, Up_{proj}, $\text{Gate}_{\text{down}}$	6.555	3.47	2.65

The results show that quantizing Q_{proj}, K_{proj}, Gate_{up}, $\text{Gate}_{\text{proj}}$ yields better *Compression-to-Degradation Ratio* than the other two quantization choices. This choice is particularly optimal when 128-sized groupwise GPTQ is applied, as shown in the 1.84× higher CDR than alternative configuration.

2.1.6 Layer-wise Mix Precision Quantization Design

The projection-wise mix precision scheme is applied to a selected range of layers due to the difference in robustness to quantization among different layers. LUQ reports that the layers towards the end have lower sensitivity to quantization loss than the middle and earlier layers [4]. The later layers' (layer 26-31) higher robustness to quantization is also observed in the experiments conducts by this approach. To further exploit the bitwidth saving benefits of the aforementioned salience determined projection-wise quantization, the proposed approach differs from LUQ in that it expands this quantization strategy to the middle layers (layer

6-26) as well, while LUQ assigns higher bitwidths to middle layers. The experimental results also show that quantizing earlier layers (layer 0-5) leads to excessive perplexity increase compared to baseline QuaRot, making the perplexity loss to model compression tradeoff less than optimal. Therefore, the weights in those layers are quantized to INT4.

Chapter 3

Hardware Accelerator Design

3.1 Overview

The proposed hardware accelerator uses TeLLMe V2's BitNet accelerator as the design basis [17] due to its highly similar overall architecture to the Llama2-7b model [20] used in the algorithmic experiments. Both models employ the same Transformer-based architecture, and their implementations of key components, including multi-head attention, positional encoding, and element-wise operations, are nearly identical. The primary difference between Llama and BitNet lies in the matrix multiplication operations used in the projection layers. While Llama-style accelerators support a wide range of data precisions, from FP16 to INT4, for projection weights, BitNet adopts ternary weights (-1,0,+1) to reduce memory footprint and computational complexity while maintaining model quality. Beyond weight precision, another reason TeLLMe V2 is selected as the design basis is that its attention unit design and fursed elementwise operations represent the state of the art (SOTA) in FPGA-based LLaMA implementation. By employing separate Reversed Prefill Attention (RPA) and Decode Attention (DA) units, the architecture eliminates causal mask computation overheads and mitigates memory bandwidth bottlenecks across both prefill and generation phases. Fur-

thermore, the inline fusion of element-wise operations—such as Rotary Position Embeddings (RoPE), SwiGLU activation functions, and dynamic precision conversions—enables continuous data streaming between processing blocks without redundant round-trips to off-chip memory. Together, these attention mechanisms and hardware-fused operations substantially reduce end-to-end execution latency, providing an optimal base architecture for edge FPGA deployment.

Building upon TeLLMe V2's framework, adapting the architecture into a flexible mixed-precision accelerator primarily requires three major hardware design tasks. First, custom weight-loading interfaces and unpacking logic are needed to stream packed INT3 and INT4 weight matrices from off-chip memory. Second, the projections with mixed precision weights require special linear engines. Third, an online Walsh-Hadamard Transform (WHT) module needs to be added and hardware-optimized for FPGA logic to effectively suppress activation outliers prior to activation quantization. By focusing modifications on these arithmetic and data-ingestion modules, the rest of the processing pipeline—including the attention engines, RMS-MAX normalization, and fused element-wise units—directly inherits TeLLMe V2's architecture with minimal modifications. The figure below shows the overall architecture of the proposed approach. The main contribution of this thesis is highlighted in purple inside the "LINEAR FUSE ELEM" block.

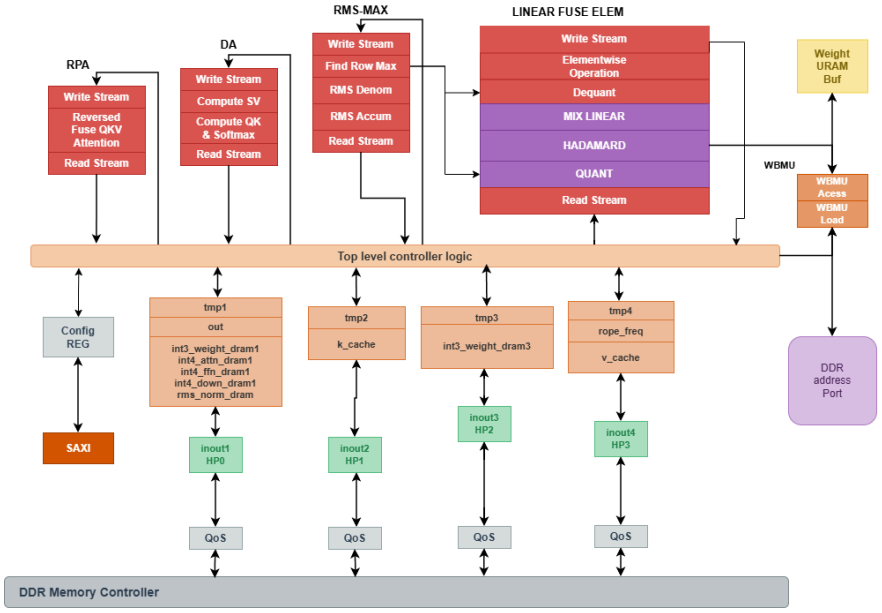

Figure 3.1: System Diagram of the Proposed Accelerator Design

3.1.1 Fused Element-Wise Operations

Derived directly from the TeLLMe v2 architecture [17], this unit integrates precision conversions and non-linear tensor operations into the streaming dataflow surrounding the core Table-Lookup Matrix Multiplication (TLMM) engine. By fusing these operations with linear computations, the architecture overlaps element-wise latencies with matrix execution:

- **Quantization & Dequantization:** Converts activations between FP16 and fixed-point precision—adapted in this work to support (INT4) activations instead of TeLLMe v2's (INT8) activations—alongside INT32-to-FP16 dequantization.

- **Rotary Position Embeddings (RoPE):** Applies offline index transformations to weight matrices to convert rotation indexing into consecutive vector memory accesses, avoiding pipeline stalls.

- **Activation & Residual Logic:** Streamlines element-wise additions for residual connections and SwiGLU multiplications directly within the streaming datapath.

3.1.2 RMS-MAX Unit

This module is also derived directly from TeLLMe v2 [17] to combine Root Mean Square Normalization (RMSNorm) and dynamic scale-factor generation into a single-pass streaming architecture. As feature vectors stream through, it computes the sum of squares in FP32 precision while concurrently tracking the channel-wise maximum absolute value (ABSMAX). Fusing normalization with max-finding delivers normalized and scaled vectors directly to downstream quantization without extra off-chip memory read passes.

3.1.3 Prefill and Decode Attention Units

Derived from TeLLMe v2 [17], the attention execution pipeline is split into specialized units optimized for the prompt prefilling ($\mathcal{O}(N^2)$) and autoregressive decoding ($\mathcal{O}(N)$) phases:

- **Reversed Prefill Attention (RPA):** Accelerates prompt prefilling under tight FPGA constraints by processing sequence blocks in a reversed and reordered schedule, eliminating causal mask computation. It streams Q, K, and V blocks through a fully fused pipeline (QK^T, online Softmax, and score-value aggregation) on-the-fly to minimize off-chip memory traffic.

- **Decode Attention (DA):** Optimizes memory-bound token-by-token generation by streaming single-token Q projections against cached K/V vectors, retaining intermediate Softmax score reductions in on-chip buffers to eliminate redundant off-chip memory round-trips.

3.1.4 Mixed Precision Linear Layer Implementation on FPGA

Implementing low-bit mix precision transformer model on FPGA requires considerable effort due to the lack of existing approaches. Recent implementations of Llama style LLMs, such as HLS Transform and TeLLMe v2 use uniform bitwidth for weights within each layer [17]

[10]. Furthermore, a mixed precision linear layer requires extra multiplexer-based selection logic to apply the designated matrix multiplication kernels for the given weight groups. This creates additional hardware overhead by consuming additional LUT and FFs and adds extra inference latency. The latency overhead can worsen if the weight columns of different precision are interweaving or non-contiguous (e.g. column 1-3, 12-18 uses INT4 weights, column 4-11 use INT3 weights).

The proposed approach addresses the above problems by processing the W3A4 and W4A4 multiplications as static, contiguous regions. This design is enabled by a key observation that weight columns with the top 12.5% salience score are the last 12.5% columns, as shown in figure 3.2. Although this phenomenon only holds true for layer 2-31, it nonetheless allows all mixed precision layers (layer 6-31) to use static matrix multiplication handling for INT3 and INT4 regions.

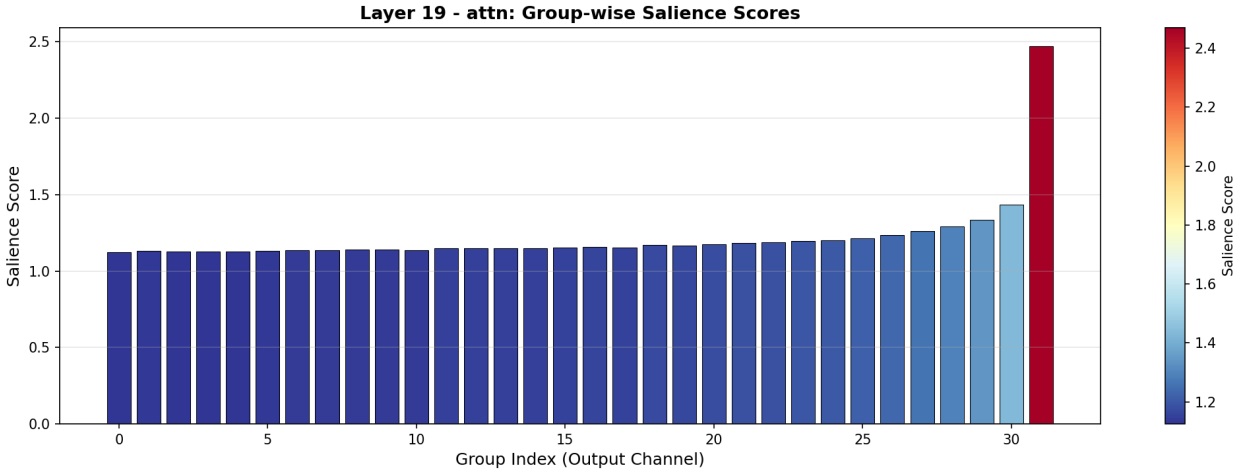

Figure 3.2: Columns with the Highest Salience Score in Layer 19 Attention Layer Projections

The pseudocode below details the overall structure of the mix precision linear layer.

```
// matrix multiplication for channels using INT3 weight
int col_i = 0;
while(col_i < int3_output_channel){
    unpack_weights_int3();
    matmul_table_lookup_int3();
    col_i += column_group;
}
// matrix multiplication for channels using INT4 weights
int col_i = int3_output_channel;
while(col_i < output_channel_count){
    unpack_weights_int4();
    matmul_int4();
    col_i += column_group;
}
```

Figure 3.3: Mixed Precision Linear Layer Structure

3.1.5 Weight Loading

Due to the existence of two different types of weights, extra DRAM ports must be added to the TeLLMe v2's transformer block design [17]. The accelerator design of the proposed approach uses total of 8 DRAM ports. Both INT3 and INT4 weights can be packed into the 256 bit AXI bus with zero bandwidth waste since they can be evenly divided by 256. INT3 weights in the mixed precision projections are assigned 3 DRAM ports, while the INT4 weights at attention layer projections, FFN layer gate and up projection, and FFN layer down projection are given 2 ports for each use case. During loading, all INT3 DRAM ports load weights in parallel, ensuring the model throughput and compressing the overall loading time by 3×. INT4 weight loading are limited to 2 DRAM ports because unpacking the INT4 weights loaded through 3 DRAM ports under the current linear layer architecture

incurs excessive resource and latency overhead relative to weight loading speedup.

3.1.6 INT4 and INT3 Mix Precision Matrix Multiplication Kernel

Tiled Matrix Multiplication

Linear operations ($\mathbf{W} \times \mathbf{A}$) for both INT4 and INT3 weights are executed in a tile-wise manner using a fixed tile size of 64×16. To maximize parallel processing density and throughput, all tile loops are fully unrolled. This execution strategy enables activation blocks to be cached and reused across consecutive weight tiles, reducing redundant memory accesses. Selecting a 64×16 tile configuration provides an optimal balance between maximizing computational parallelism during linear projection execution and constraining hardware resource consumption, mitigating the heavy logic and DSP footprint typically incurred by aggressive loop unrolling.

Lookup-table Enabled W3A4 Matrix Multiplication

For INT3 $\times$ INT4 matrix multiplication, the proposed method uses a LUT-based matrix multiplication kernel distinct from the one in TeLLMe v2 [17]. At each projection, an 2D LUT containing all 128 possible multiplication outcomes is created. A group of one activation plus one weight are used as indices to access the INT3 $\times$ INT4 result. The negative INT4 activations are mapped to positive values during matrix operation, while negative numbers in the INT3 weights are pre-processed to map to positive numbers to avoid extra instructions during inference. Inside the matrix multiplication loop, each W $\times$ A is replaced by a table lookup.

The proposed one-time generated table lookup method is compared with two other methods in terms of total latency and resource usage. The first alternative method is a group-wise table lookup matrix multiplication kernel inspired by TeLLMe v2 [17]. This method generates $T_{\text{MAC\_TABLE\_NUM}}$ parallel lookup tables for activation groups of 2. The complete dimensions

of the lookup table is given as follows:

$$Size_{\text{LUT}} = T_{\text{MAC\_TABLE\_NUM}} \times 64 \tag{3.1}$$

The first dimension of the LUT, $T_{\text{MAC\_TABLE\_NUM}}$, is set to 32 so that it can be evenly divided by the input dimensions of all projections (1536 for all attention layer projections, Gate_{up} and $\text{Gate}_{\text{proj}}$, and 4096 for FFN layer's $\text{Gate}_{\text{down}}$). The second dimension, 64, is the number of possible outcomes of the multiply-accumulate operations of a group of 2 activations. A lookup table is generated and populated with values at the start of each tile and subsequently used in the matrix multiplication loops. The second kernel design performs tile-wise matrix multiplication. It performs conventional multiply-and-accumulate without any need for lookup tables. To ensure the fairness of comparison, all three methods are tested with a processing tile size of 16×64, and all addition and multiplication operations within the matrix multiplication, lookup table generation, and table lookup loops except for the accumulate operation in the Group-wise LUT implementation are bind to Lookup-table (LUT) and FF (Flip-Flop) rather than DSPs.

Table 3.1: Latency and Resource Utilization Comparison for Different INT3 Matrix Multiplication Methods

Method	**Total W3A4 Latency (ns)**	**DSP**	**FF**	**LUT**
Group-wise LUT	29,040	64	24,033	52,308
Conventional MAC	19,320	0	2,364	36,107
One time LUT generation	18,900	0	9,086	17,945

The table above shows the total latency and resource utilization of the three methods for Attn_{q} projection. The group-wise table lookup method uses the most FFs and LUTs out of all three approaches even when it accumulate operation are assigned to DSP. While table lookup itself is much cheaper than MAC operations, the lookup table generation function requires

22,144 FF and 42,392 LUT, more than all other methods combined. A likely reason for this high resource usage is that each lookup table precomputes all 64 possible outcomes of W 3× A of 32 groups of 2 activations in parallel. Also, the group-wise lookup tables are generated for all weights inside Attn_q, regardless of their bitwidths. The lookup table generation function is therefore called 24 times and adds a non-negligible 7440 ns of overhead, making the group-wise lookup table method the least optimal option from a latency perspective. The one-time generated table lookup method uses fewer than 50% of the LUTs required by conventional MAC despite using nearly 7000 more FFs.

3.1.7 DRAM and Weight Buffer Design For Mixed Weights

The accelerator loads the entire INT3 and INT4 weight projection matrices onto the on chip memory before each linear layer projections. Activations, on the other hand, are loaded one tile at a time during linear operations. This stream loading strategy harmonizes the underlying control logic across compute and decode stage, seamlessly adapting to the distinct compute and memory characteristics of each:

- **Compute-Intensive Prefill Stage:** Because prompt processing is inherently compute-bound, this streaming scheme maximizes hardware utilization by concurrently processing large batches of tokens. By streaming activations in structured tiles of $T_{\text{act}} = 2\times T_{\text{MAC\_TABLE\_NUM}}$, the pipeline parallelizes the matrix workload across prompt tokens, executing up to $32 \times T_{\text{MAC\_TABLE\_NUM}}$ parallel multiply-accumulate (MAC) operations per cycle to fully saturate the processing array.

- **Memory-Bound Decode Stage:** During token-by-token generation, the inference bottleneck shifts from compute density to memory bandwidth. Because activations are produced incrementally, the processing elements (PEs) consume activation tiles immediately upon loading. This zero-wait ingestion prevents downstream functional units in the pipelined linear layer from stalling, ensuring maximum throughput during

memory-bound decoding.

By maintaining a static weight footprint on-chip while streaming activation tiles, the architecture ensures high throughput in prefill through batch parallel compute and maintains lower latency in decode.

W4A4 Matrix Multiplication

The proposed approach also experimented with different matrix multiplication methods for the W4A4 portion of the mix precision linear layer kernel. It was initially hypothesized that a one-time generated table lookup kernel, similar to the one adopted for W3A4 multiplication, can also reduce LUT usage when the weights become 4-bit. Two variations of table lookup kernel are implemented and compared with conventional MAC-based kernel. Variation 1 uses a unified lookup table of size 16×16 to store all possible products of both W3A4 and W4A4 multiplications. This design requires matching the negative values in the INT3 or INT4 range to the corresponding positive lookup table indices during inference. It also abandons the static INT3 and INT4 column design shown in 3.4 in favor of a single matrix multiplication function for both W3A4 and W4A4 and a selection logic that switches between INT3 and INT4 weight unpacking function based on the index range of the current column group. A pseudo-code representation of Variation 1's structure is shown below:

```
int col_i = 0;
while(col_i < output_channel_count){
    if(col_i < int3_column_dim){
        unpack_weights_int3();
    }
    else {
        unpack_weights_int4();
    }
    matmul_table_lookup_16x16();
    col_i += column_group;
}
```

Figure 3.4: The Linear Layer Structure of the Unified Table Lookup Approach

Variation 2 uses a 16×16 lookup table that for $INT4 \times INT4$ multiplication results while using a 8×16 lookup table for the W3A4 portion of the mix precision linear layer. Since variation 1 uses the same table lookup multiplication for INT3 and INT4 weights, Table 4.4 compare the total latency and resource utilization for linear operation of the entire Attn_q weight matrix.

Table 3.2: Total Linear Layer Latency and Resource Utilization Comparison Under Different INT4 Matrix Multiplication Methods (Attn Q Projection)

Method	**Total Latency**	**FF**	**LUT**
Conventional MAC	43,320	12,762	59,085
Unified Table Lookup (Var. 1)	70,560	5,042	44,973
Separate Table Lookup (Var. 2)	43,320	20,500	93,670

Variation 1 of the table lookup based INT4 matrix multiplication uses the least amount of LUT and FFs, but also leads to excessive latency (62.9% more than the latency of conven-

tional MAC). Also, using a unified lookup table function for both INT3 and INT4 means that not only will INT3 weights need to be upcasted into INT4 for table lookup, it also creates massive memory waste since half of the entries remain unused for 87.5% of multiplications during mix precision projections. Variation 2's high LUT and FF usage is due to the need to partition the 16 ×16 lookup table for INT4 multiplication. Without partitioning the lookup table, the BRAM usage can easily exceed the total BRAM capacity of the target device.

Another reason that the unified table lookup approach is not used is that it leads to excessive BRAM utilization in gate down projection, where all weights are kept in INT4 precision. As shown in table 2.3, the BRAM usage easily exceeds the total BRAM capacity, and applying memory access optimizations such as array partitioning transforms the BRAM pressure into high LUT usage.

Table 3.3: Total Linear Layer Compute Latency and Resource Utilization Comparison Against 2 Different Unified 16x16 Lookup Table Methods ($\text{Gate}_{\text{down}}$ projection)

Method	**Total Latency (ns)**	**BRAM**	**FF**	**LUT**
Unified Lookup Table	65,280	550	1,931	35,134
Unified Lookup Table (memory optimized)	43,320	0	20,500	93,670
Conventional MAC	65,000	0	2,436	36,926

Since the table lookup methods for INT4 × INT4 matrix multiplications lead to either excessive latency or resource usage, the proposed approach uses the simpler and BRAM friendly conventional MAC for INT4 weights in mix precision and uniform precision projections.

3.1.8 2-Stage Pipelined Online Hadamard Transform Unit

To reduce outliers in activations, Hadamard matrix rotation is applied during multiple projections. While most Hadamard rotations, such as the ones in Q, K, V projections of the attention layer and $Gate_{\text{up}}$ and $Gate_{\text{proj}}$ of the FFN layer, are applied to the weight matri-

ces prior to inference to avoid extra inference time latency, the activation of $Gate_{\text{down}}$ and $Attn_{\text{proj}}$ must be rotated online via Fast Walsh Hadamard Transform (FWHT) [7] to cancel out the Hadamard matrix multiplication applied to the weight matrix:

$$\text{Attn}_{\text{out}} = (\mathbf{O} \cdot \mathbf{H}) \cdot \left(H^T W_O Q\right) \tag{3.2}$$

$$\text{MLP}_{\text{out}} = (\mathbf{Y} \cdot \mathbf{H}) \cdot \left(H^T W_{\text{down}} Q\right) \tag{3.3}$$

The H in Equations 3.2 and 3.3 refers to the online Hadamard matrices. They are cancelled out by H^T. Although QuaRot reports that adding two online Hadamard multiplications implemented as FWHT adds a modest 2% end-to-end latency increase for Llama2 models on GPU [3] [6], the extra latency for FPGA deployment becomes non-negligible, mainly due to the reduced number of computing resources on FPGA. A single O_{proj} online Hadamard transform incurs 3.88×10^8 ns of extra latency, which more than doubles the latency of O_{proj} itself. To mitigate this latency problem, the O_{proj} Hadamard transform is split into two parts and connected through dataflow. The first part performs FWHT on 12 subgroups of 128 elements each, and the second part performs a 12-point Hadamard mixing across the sub-vectors at each fixed intra-block index of the FWHT output. Fig. 3.5 shows the overall flow of the new online Hadamard rotation during O_{proj}.

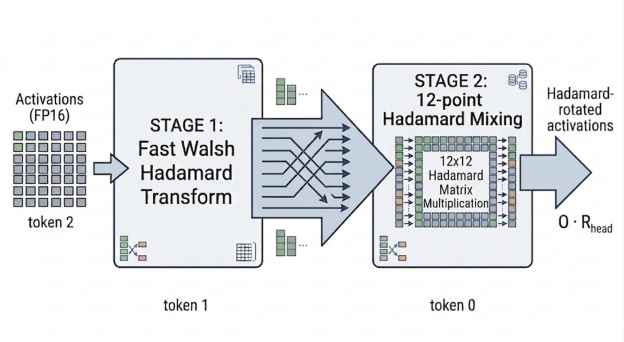

Figure 3.5: Overall Flow of Online Hadamard Rotation

The diagram above shows that the new 2-stage Hadamard unit performs 12-point Hadamard mixing on token 0 while token 1 undergoes FWHT. Functions connected via Vitis HLS dataflow execute concurrently in a pipelined fashion. Consequently, the total execution latency T_{total} across N dataflow-connected sub-functions is governed by the bottleneck stage:

$$T_{\text{total}} = \max_{i \in \{1,2,\ldots,N\}} \{T_i\} \tag{3.4}$$

where T_i represents the latency of the i-th sub-function. Because activations in O_{proj} are read, rotated, and written on a per-token basis, the total online Hadamard transformation latency for sequence length $L \geq 1$ simplifies under dataflow execution to:

$$T_{\text{total}} = \max\left(T_{\text{FWHT}}, T_{\text{load}}, T_{\text{store}}\right) \tag{3.5}$$

By overlapping function execution, the architecture avoids the sequential latency sum of unoptimized datapaths:

$$T_{\text{sequential}} = T_{\text{FWHT}} + T_{\text{mixing}} + T_{\text{load}} + T_{\text{store}} \tag{3.6}$$

In the above equations, t_{mixing} denotes the latency of the Hadamard mixing. It is estimated

that it takes 5.322×10^7 ns to complete. t_{load} and t_{store}, which denotes the latency of loading the activations from the FIFO stream and writing the post-rotation output, takes 980 ns each. Although splitting online Hadamard rotation into two stages in the dataflow chain adds an extra pair of load and unload operation, these additional operations do not contribute to the total effective latency because they can be done concurrently with FWHT when token length $L \geq 1$.

Hadamard scale merging

Hadamard rotation at both O_{proj} and $Gate_{\text{down}}$ requires one multiplication with the Hadamard scale for every activation. Since the scale does not have to be computed or loaded on chip during inference, it can be merged with the quantization scale in the activation quantization function later in the linear layer. This optimization saves 1536 multiplications during O_{proj} and 4096 multiplications during $Gate_{\text{down}}$, further reducing the overhead of online Hadamard rotation.

Chapter 4

Experiments

4.1 Experimental Setup

The proposed quantization method is applied to Llama2-7B model [15]. It applies INT4 quantization to activation and kv cache and INT3 + INT4 quantization to weights with an average bitwidth 3.56 bits. Weights and activations are symmetrically quantized with a default clipping ratio of 1.0, while the kv cache is quantized using a clipping ratio of 0.95. Model perplexity is evaluated using 128 samples from WikiText-2 [14] with a sequence length of 128. The perplexity evaluation is run on Nvidia RTX 3090. The mix precision accelerator is designed and evaluated on Vitis HLS 2023.1, and the design targets Kria KV260 Vision AI Starter Kit which features a custom Zynq UltraScale+ MPSoC that contains 117,120 LUTs, 234,240 FFs, 1,248 DSP slices, 144 blocks of BRAM, and 64 blocks of URAM [2].

4.2 Evaluations

4.2.1 Model Perplexity Comparison

The perplexity on WikiText 2 dataset of Llama2 model is compared with those of QuaRot, SpinQuant, SliM-LLM, ResQ, OmniQuant, and AWQ. OmniQuant is a post-training quantization method that optimizes learnable weight clipping thresholds and equivalent transformations to handle activation outliers, while AWQ ensures model performance under W4A16 configuration by protecting the salient weight channels based on activation magnitudes [12, 19, 3, 18, 11, 13].

Table 4.1: Comparison of Model Quantization Approaches

Method	**Weight Bitwidth**	**Activation Bitwidth**	**Wikitext 2 PPL ↓**
Proposed Approach	**3.56**	**4**	**6.214**
QuaRot [3]	4	4	6.089
SpinQuant [13]	4	4	5.9
Slim-LLM [11]	3	16	6.24
ResQ [18]	4.5	4.5	5.80
AWQ [12]	4	16	5.60
OmniQuant [19]	3	16	6.03

As shown in Table 4.3, the reported perplexity (PPL) for QuaRot (6.089) differs from the value reported in its original paper (6.093) due to the application of a different weight and activation quantization clipping ratio. The proposed approach yields a higher perplexity than the state-of-the-art SpinQuant for two main reasons: first, while SpinQuant employs learned orthogonal matrix rotations, the proposed approach is built upon the QuaRot framework, which utilizes randomized Hadamard matrices; second, it adopts the same modified clipping ratios that leads to higher perplexity in QuaRot-style models. The proposed approach yields slightly higher perplexity than OmniQuant, SpinQuant, ResQ and AWQ. OmniQuant's lower perplexity compared to the proposed approach can be attributed to its use of a learnable quantizer and FP16 activations. AWQ achieves a better perplexity score at the cost of an additional 10% weight memory and expanding the size of the activation memory and KV

cache by 4 times. SpinQuant uses learned Hadamard rotational matrices than QuaRot which makes the perplexity score inherently better than QuaRot and its derivatives. ResQ has the second lowest perplexity and maintains low bitwidths (4.5-bit) for both weight and activations. Nevertheless, the proposed approach successfully pushes the average weight bitwidth down to 3.56 bits on top of QuaRot without a substantial increase in perplexity. Notably, its perplexity is only slightly higher than the W3A16 OmniQuant and its parent model QuaRot. Furthermore, it achieves a slightly better perplexity than SliM-LLM, outperforming it even though SliM-LLM relies on full 16-bit activations while the proposed method maintains strict 4-bit activation quantization.

4.2.2 Memory Saving

Memory Saving Compared to W4A16 Quantization

While W4A16 is a popular choice for post-training quantization as demonstrated by GPTQ [8], AWQ [12], and OmniQuant [19], keeping activations in FP16 introduces key limitations: first, it requires extra latency for runtime integer to FP16 weight dequantization; second, FP16 operations incur higher arithmetic latency than integer datapaths; and third, 16-bit activation states inflate prefill memory footprint. Fig. 4.2 illustrates the activation memory savings of the proposed approach relative to a W4A16 baseline compared against SpinQuant [13], QuaRot [3], and ResQ [18]. The activation memory savings of the proposed approach is on parity with QuaRot and SpinQuant and is better than the 4.5-bit activation ResQ.

Table 4.2: Activation Bitwidth and Memory Saving Relative to $W4A16$

Approach	**Activation Bitwidth**	**Activation Memory Saving (%) ↑**
ResQ [18]	4.5-bit	71.88%
QuaRot [3]	4-bit	75.00%
SpinQuant [13]	4-bit	75.00%
Proposed	**4-bit**	**75.00%**

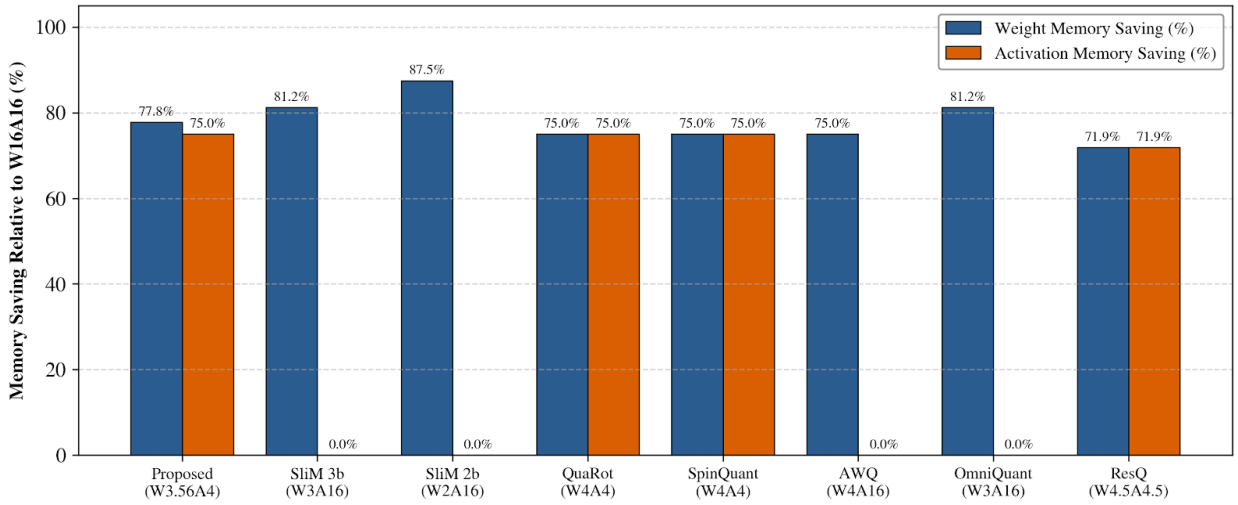

Figure 4.1: Memory Saving Percentage ↑ Relative to FP16 Baseline

Memory Saving Compared to FP16 Baseline

Fig. 4.1 shows that the proposed approach achieves more than 75% weight and activation memory size reduction compared to FP16 baseline. It save more weight memory than SpinQuant, QuaRot, and ResQ. While SliM-LLM and OmniQuant save more weight memory due to its use of 3-bit and 2-bit weights, their activation memory is 4× the size of that of the proposed approach [19, 11].

4.2.3 Compression-to-Degradation Ratio Comparison

The CDR of SliM-LLM [11] and OmniQuant [19] is obtained by comparing model perplexity against AWQ [12]. The CDR of the proposed approach is obtained by comparing its WikiText2 perplexity with that of QuaRot. The CDR of a variation of the proposed approach that applies uniform INT3 quantizations for the weights inside mixed precision projections in layers 6-31 while keeping all other weights in INT4, hereby named *Proposed Approach (3.5-bit)* due to its overall bitwidth of 3.5032 bits, is also listed in the table below for comparison.

The proposed approach that uses 1:7 ratio of INT4 to INT3 weights inside mixed precision

Table 4.3: Comparison of Model Quantization Approaches

Method	**PPL**	**Reference PPL**	**Memory Saving (%)**	**CDR**
Proposed Approach	**6.214**	**6.089**	**10.87**	**5.3**
Slim-LLM (W3A16) [11]	6.240	5.60	25	2.189
OmniQuant (W3A16) [19]	6.03	5.60	25	3.259
Proposed Approach (3.5-bit)	6.253	6.089	12.42	4.617

projections has the best CDR despite its relatively small model size reduction. Although SliM-LLM achieves 25% model compression compares to AWQ, its achievment comes at the cost of considerable perplexity degradation. The CDR of 2.189 indicates that the bitwidth saving potential of SliM-LLM is only 2.189 times the loss of model inference capability, much less than the proposed approach's 5.3. OmniQuant benefits from its learnable quantizer [19] and achieves lower perplexity and higher CDR than SliM-LLM for the same amount of memory saving, but still falls behind the proposed approach. Proposed Approach (3.5-bit) has slightly higher memory saving percentages and has the additional benefit of enabling simpler hardware implementation for mixed precision layers since it no longer requires 12.5% column groups to be kept in INT4. However, it produces less optimal CDR than the method that uses 12.5% INT4 weights and 87.5% INT3 weights for mixed precision projections.

4.2.4 Resource Saving in the Accelerator Design

Table-lookup based INT3 matrix multiplication kernel use fewer than 50% of the LUTs required by conventional MAC-based kernel despite a moderate increase in FF usage. This tradeoff is optimal because LUT is more constrained on KV260 than FFs, as shown in Section 4.1.

Table 4.4: Resource Utilization Comparison Between INT3 and INT4 Matrix Multiplication on KV260

Method	**FF**	**FF (%)**	**LUT**	**LUT (%)**
LUT-based INT3 MatMul	**9,086**	**3.88%**	**17,945**	**15.32%**
INT3 Matrix Multiplication	2,364	1.01%	36,107	30.83%

The mixed precision implementation have the same computational latency as the W4A4 implementation for all projections except $Attn_{\text{proj}}$. The $Attn_{\text{proj}}$ latency of the proposed mixed precision linear layer benefits from the 2-stage pipelined online FWHT unit that overlaps the latency of 12-point Hadamard mixing with that of group-wise Hadamard transform.

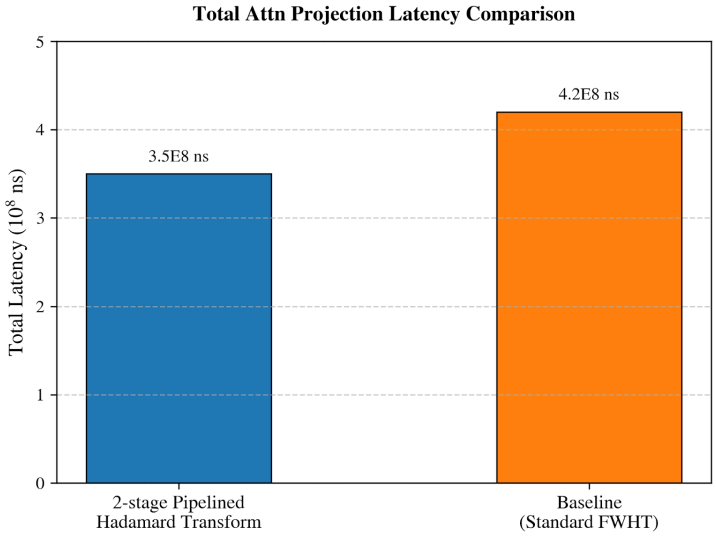

Figure 4.2: Total Attention Projection Latency Comparison ↓ (Optimized 2-stage FWHT vs Standard FWHT) [7]

Lastly, the proposed accelerator design benefits from the reduced bidwiths of 87.5% INT3 + 12.5% INT4 weights in mix precision projections. As shown in Table 4.3, projections that use the 87.5% INT3 + 12.5% INT4 configuration achieves 43% less loading latency, or 1.78× per-projection weight loading speedup. This performance gain is driven by two key factors: 3-DRAM port parallel loading for INT3 weights and the higher packing density of INT3 weights compared to INT4. Loading 12.5% weights in INT3 takes only 1.15E4 ns in attention and 3.07E4 ns in FFN. Dual-DRAM port parallel loading for INT4, on the other hand, takes 2.316E4 ns to load 12.5% of the weight matrix in attention and 6.17E4 ns in FFN. This reduction in weight transfer overhead could potential translate to extra benefits and throughput improvement during the memory transfer intensive autoregressive decoding phase.

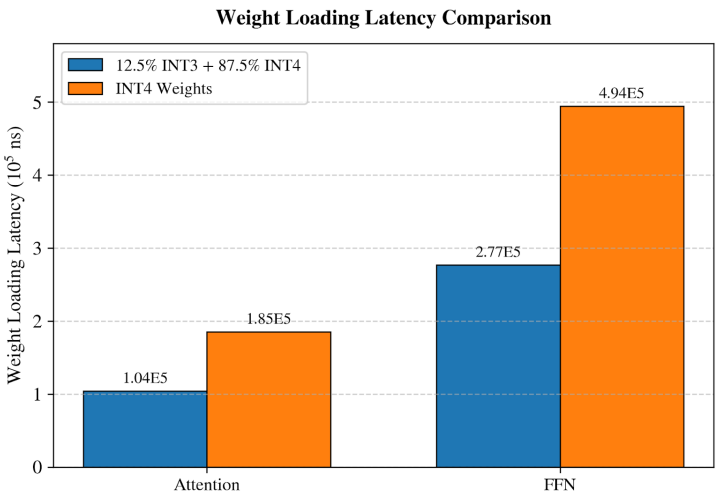

Figure 4.3: Weight Loading Latency ↓ Comparison (Mixed Precision vs INT4)

Chapter 5

Future Research

Model Scalability and Architecture Diversity

While the proposed projection-wise mixed-precision framework demonstrates clear efficacy on mid-scale models, a priority for future work is evaluating its scalability across broader model sizes and architectural families. Specifically, this strategy will be validated on larger-scale models such as Llama2-70B. Furthermore, the proposed quantization method can be applied to more compact and FPGA-friendly transformer LLMs—such as Phi-3-Mini [1], Gemma-2B [9], and Qwen-2.5 (1.5B/3B) [21]—to investigate whether rotation-based precision mapping can effectively mitigate accuracy degradation in highly parameter-constrained regimes.

Hardware-Software Co-Design and FPGA Kernel Optimization

The current mixed-precision linear layer implementation on FPGA introduces non-trivial resource overheads. Because projections with INT3 + INT4 weights currently require instantiating separate matrix multiplication kernels for each precision level, overall DSP and LUT usage exceeds that of a uniform INT4 baseline. Future hardware research will proceed along two trajectories: (i) designing area-efficient, resource-optimized INT3 processing ele-

ments to lower single-kernel footprint, and (ii) restructuring mixed-precision projections to execute over a unified INT3 weight format (e.g., via non-uniform bit-packing or low-rank scaling vectors), thereby eliminating dual-kernel instantiation and realizing true compute resource saving on FPGA.

Sub-3-Bit Precision and Extreme Model Compression

Finally, in the future the model compression can be further expanded by exploring ultra-low bit-width regimes (e.g. INT2 and ternary weight). While the current mixed-precision configuration achieves $\sim 10\%$ memory reduction over standard INT4 baselines while preserving accuracy, extending lower-precision quantization to Hadamard transformed models holds significant potential. The future research will focus on combining Hadamard transformed projection-wise mixed precision quantization with other techniques such as learned channel scales [19] to maximize compression efficiency without incurring severe perplexity penalties.

Chapter 6

Conclusion

This thesis presents an efficient low-bit mixed precision quantization method that ensures model inference quality while pushing weight quantization to 3.56-bit and an FPGA implementation of this method that overcomes the mixed precision upcasting bottlenecks inherent to GPUs. The proposed quantization method achieves 12% overall model compression and 75% reduction in activation memory over W4A16 baselines on Llama 2-7B, maintaining strong model performance with only 2.1% perplexity degradation. Deployed on a resource-constrained AMD KV260 FPGA, the proposed hardware accelerator reduces latency and improves resource efficiency while maintaining model performance through the Hadamard transform. The custom 3-bit table-lookup based matrix multiplication kernels reduce LUT usage by over 50% compared to standard MAC units, and an integrated 2-stage online Hadamard transform unit reduces attention projection latency by 13.7%. Ultimately, this work presents a hardware-algorithm co-design that achieves efficient model compression and compute-side resource savings through the deployment of a sub-4-bit weight quantized model on FPGA.

Bibliography

[1] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann, J. R. Lee, Y. T. Lee, Y. Li, W. Liu, C. C. T. Mendes, A. Nguyen, E. Price, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, X. Wang, R. Ward, Y. Wu, and D. Yu. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] AMD Xilinx. *Kria KV260 Vision AI Starter Kit Data Sheet (DS986)*. Advanced Micro Devices, Inc., 2023. Data Sheet DS986 (v1.2).

[3] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, M. Jaggi, D. Alistarh, T. Hoefler, J. Hensman, and P. Cameron. QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[4] S. Bhatnagar, A. Xu, K.-H. Tan, and N. Ahuja. LUQ: Layerwise Ultra-Low Bit Quantization for Multimodal Large Language Models. *arXiv preprint arXiv:2509.23729*, 2025.

[5] J. Cong, B. Liu, Z. Zhang, P. P. Zou, and Y. Zhao. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[6] T. Dao. fast-hadamard-transform: Fast Hadamard Transform in PyTorch. `https://github.com/Dao-AILab/fast-hadamard-transform`, 2023.

[7] S. Fouladi. fastwht: Fast Walsh-Hadamard Transform Implementation. `https://github.com/sfouladi/fastwht`, 2019.

[8] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

[9] Gemma Team. Gemma 2: Improving open language models via two-stage knowledge distillation. *arXiv preprint arXiv:2408.00118*, 2024.

[10] A. He, D. Key, M. Bulling, A. Chang, S. Shapiro, and E. Lee. HLSTransform: Energy-Efficient Llama 2 Inference on FPGAs Via High Level Synthesis. *arXiv preprint arXiv:2405.00738*, 2024.

[11] W. Huang, H. Qin, Y. Liu, Y. Li, Q. Liu, X. Liu, L. Benini, M. Magno, S. Zhang, and X. Qi. SliM-LLM: Salience-Driven Mixed-Precision Quantization for Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[12] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

[13] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort. SpinQuant: LLM Quantization with Learned Rotations. In *International Conference on Learning Representations (ICLR)*, 2025.

[14] S. Merity, C. Xiong, J. Bradbury, and R. Socher. WikiText-2 Language Modeling Dataset, 2016.

[15] Meta AI. Llama 2 7b model hub repository. `https://huggingface.co/meta-llama/Llama-2-7b-hf`, 2023. Accessed: 2026.

[16] NVIDIA Corporation. *Parallel Thread Execution ISA Version 8.5*. NVIDIA Corporation, 2024. CUDA Design Guide.

[17] Y. Qiao, Z. Chen, Y. Zhang, Y. Wang, and S. Huang. TeLLMe: An Efficient End-to-End Ternary LLM Prefill and Decode Accelerator with Table-Lookup Matmul on Edge FPGAs. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2026.

[18] U. Saxena, S. Sharify, K. Roy, and X. Wang. ResQ: Mixed-Precision Quantization of Large Language Models with Low-Rank Residuals. *arXiv preprint arXiv:2412.14363*, 2024.

[19] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2024.

[20] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, P. W. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, Y. Xu, Z. Y. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[21] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, and J. Zhou. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.