POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Post-Training Quantization of a Transformer-based Autonomous Driving Neural Network

Advisors
Prof. Mario Roberto Casu
Dr. Edward Manca

Candidate
Giovanni Gaddi

Academic Year 2024-2025

Summary

The ongoing development of Autonomous Driving (AD) systems has resulted in an increased demand for perception models that combine high accuracy with computational and energy efficiency. Recent advancements include BEVFusion, a state-of-the-art multi-sensor fusion Neural Network (NN) framework that combines camera and LiDAR into a unified representation called Bird's-Eye View (BEV). This approach enables robust spatial reasoning and 3D object detection. In this scenario, BEVFusion reaches competitive performance on large-scale benchmarks such as NuScenes, a dataset for multi-modal AD NNs, providing training data from camera and LiDAR sensors with 3D object annotations. However, BEVFusion's computational and memory requirements make real-time deployment on embedded or resource-constrained devices extremely difficult despite its high accuracy.

An important optimization in NN deployment is quantization. The objective of this technique is to substitute floating-point with integer arithmetic, minimizing the accuracy loss of the resulting NN. Therefore, the use of integer operations results in Quantized NN (QNN) that are lighter and easier to deploy in resource constraint scenarios.

This thesis investigates the application of Post-Training Quantization (PTQ) in BEVFusion to lower its deployment requirements, without changing the model structure nor retraining it. To this end, I developed a comprehensive framework to apply PTQ, incorporating a per-module calibration strategy that allows for quantization of NN weights and activations of the most computational-intensive layers. Moreover, with the objective of minimizing the common instabilities of PTQ, I linearly vary the scale parameters to find the best of each quantized layer of this NN.

Furthermore, I propose a Mixed-Precision Quantization (MPQ) exploration engine. MPQ is an established technique that tries to identify tradeoffs between the number of bits and overall accuracy for each layer of the NN independently. To explore this design-space, I propose a Genetic Algorithm (GA). GAs are a class of optimization engines well known for their scalability and complexity. The GA-based MPQ design space exploration varies the number of bits of weights and activations of each quantized layer of the NN, trying to keep the cosine similarity between the original and the quantized NN activations of each sub-module as close as possible.

Experimental evaluations conducted on NuScenes metrics such as mean Average Precision (mAP) and NuScenes Detection Score (NDS) show that the suggested quantization methods preserve the fundamental perception capabilities of the original design while achieving a significant decrease in model size and computational requirements. Additionally, the study validates the benefits of quantization on state-of-the-art accademic and industry AD NNs, establishing a path to deploy these big and complex NNs into an AD edge-context.

Ringraziamenti

Giunto al termine di questo percorso di studi, che culmina con la presentazione della mia tesi, riconosco non solo di aver strutturato e consolidato le mie competenze nell'Ingegneria Informatica come professionista, ma soprattutto di essere maturato come persona. Le conoscenze acquisite, le amicizie coltivate e i legami stretti in questi anni rappresentano il vero valore di questa esperienza. Desidero dedicare questo spazio finale, con profonda gratitudine, a chi ha creduto in me, offrendomi supporto e sopportazione.

La mia riconoscenza più profonda va ai miei genitori: a mio padre Andrea, per la sua costante capacità di incoraggiarmi e infondermi la giusta motivazione nei momenti di maggiore fatica, e a mia madre, per essere stata un sostegno morale prezioso, sempre presente con la sua vicinanza e sensibilità. Questa gratitudine si estende a mia sorella Anna, punto di riferimento, la cui presenza silenziosa è sempre stata una fonte di forza e grande sostenitrice. Questa conquista è in gran parte anche vostra; senza il vostro appoggio incondizionato, non sarei mai arrivato fin qui.

Un ringraziamento sincero va a tutti gli amici che hanno reso unico questo percorso. A Torino, nel primissimo anno di università, ho conosciuto Mirko e Giorgio, che con serietà e leggerezza, sono stati i depositari di grasse risate e hanno facilitato l'interazione della mia personalità con il mondo del Politecnico, rivelandosi compagni di studio, di svago e di sostegno essenziali, insieme a Pietro e Simone, con i quali negli ultimi anni ho condiviso "anche troppo": le mura, i pasti e gli inevitabili stress pre-esame.

Un pensiero speciale va a Francesco, il mio amico di più lunga data e un riferimento costante. La nostra amicizia, nata e cresciuta nel tempo, si è rivelata una presenza rassicurante e insostituibile durante i momenti più impegnativi, aiutandomi a ritrovare l'equilibrio quando ne avevo più bisogno.

Infine, ringrazio Nicoletta, un'amica più recente ma insostituibile: negli ultimi mesi è stata a me vicina, offrendo un supporto emotivo costante. La ringrazio per le discussioni stimolanti, le camminate rigeneranti, i preziosi consigli e le nostre classiche chiacchierate ragionevoli.

Esprimo immensa gratitudine al docente Mario Roberto Casu, che mi ha affiancato e indicato la prospettiva di questa tesi. Sebbene il percorso non abbia sempre

condotto agli esiti inizialmente sperati, mi ha guidato ad approfondire inaspettatamente la mia conoscenza nel mondo dell'informatica e della sua applicazione nel mondo. Lo ringrazio sinceramente per il tempo dedicatomi, a dispetto dei suoi innumerevoli impegni professionali.

Un ringraziamento speciale a Edward: correlatore e amico. È stato il promotore entusiasta di questo progetto e senza la sua visione e il suo supporto costante questo lavoro non avrebbe avuto questa forma e questo finale. La sua disponibilità è stata fondamentale, fornendo idee cruciali, riscontri tempestivi e soluzioni pratiche per ogni problema.

Grazie

Contents

Chapter 1

Introduction

1.1 Context and Background

The foundational Transformer architecture, introduced in the seminal 2017 paper *Attention Is All You Need* [1], initiated a paradigm shift in sequence modeling. Originally developed for Natural Language Processing (NLP), it swiftly displaced Recurrent Neural Networks (RNNs) by demonstrating a superior ability to capture long-range dependencies in text, all while being significantly more parallelizable. Its core mechanism, Self-Attention, allows the model to compute a representation for each element in a sequence by dynamically weighing its relationship with all other elements. This ability to assess context globally, rather than through the constrained local-receptive fields of architectures like CNNs, was the key to its success.This revolution soon propagated to computer vision. The key breakthrough came with models like the Vision Transformer (ViT)[2], which demonstrated that by partitioning images into a sequence of *patches* (treated as "tokens"), the same Transformer architecture could achieve state-of-the-art results in image classification, object detection, and segmentation.

1.1.1 The Rise of Transformer Models in 3D Perception and Autonomous Driving

Transformer-based architectures have demonstrated strong performance and are increasingly prevalent in 3D object detection leaderboards.

Advantages and Architectural Development

Transformers, founded on the attention mechanism, have shown compelling capabilities in computer vision tasks, particularly in 3D object detection where they offer distinct advantages over conventional convolutional architectures:

1. *Flexible Interactions and Receptive Fields*: The query–key–value design in Transformers allows for more flexible interactions between different data representations. Furthermore, the self-attention mechanism provides a larger effective receptive field compared to standard convolutions[3].

2. *Long-Range Dependencies*: Transformers effectively capture long-range interactions for faraway objects and learn spatial context-aware dependencies, helping reduce false negatives and improve performance[4].

The development of Transformer architectures specifically tailored for 3D object detection has followed three main stages:

1. *Novel Transformer Modules*: This stage incorporated new Transformer modules with specialized attention mechanisms into conventional 3D detection pipelines to extract more powerful features. For instance, Pointformer[5] introduced Transformer modules into point backbones, using point features and coordinates as queries and applying self-attention to corresponding features.

2. *DETR-like Architectures*: Inspired by DETR[6], this stage introduced query-based Transformer encoder–decoder designs. These methods utilize a set of object queries to interact with features and predict 3D bounding boxes. DETR3D[7], for example, generates 3D reference points for each object query, projecting these 3D locations onto multi-view image planes to aggregate features as keys and values, and applying cross-attention to decode the 3D box.

3. *Vision Transformer (ViT)-like Architectures*: Drawing inspiration from ViT[2], this stage involved splitting inputs such as point clouds or voxels into patches and applying self-attention locally and across patches[4].

A critical challenge faced by Transformer-based models is the quadratic time and space complexity inherent to fully connected self-attention, which requires defining proper query–key–value triplets and specialized attention mechanisms.

Applications Across Different 3D Detection Modalities

Transformers have been broadly adapted across various types of 3D object detectors, showcasing their versatility across different sensor modalities and data representations.

In LiDAR-based 3D object detection, Transformers have been successfully incorporated into grid-based (voxel/pillar) and point-based approaches:

- *Voxel and Grid-Based Methods*: Grid-based detection backbones increasingly adopt Transformer-based architectures. Voxel Transformer (VoTr)[8] was proposed to exploit long-range contextual dependencies among voxels. SECOND[9]-like pipelines using sparse convolutions have inspired successors integrating

Transformer models such as CT3D[10] and VoTr[8], achieving notable performance gains. CT3D[10] utilizes a channel-wise Transformer for better proposal refinement, while SWFormer[11] replaces conventional convolutional backbones with novel voxel-based Transformer designs.

- *Point-Based Methods*: Pointformer[5] introduced a point-based Transformer designed to replace the PointNet[12] backbone for point cloud understanding.

In camera-based detection, Transformers play a crucial role in multi-view aggregation and feature fusion, particularly in addressing depth ambiguities:

- *Multi-View Detection*: Multi-view 3D object detection has advanced rapidly with BEV perception and Transformer-based fusion. Query-based multi-view detection models generate BEV object queries that interact with camera view features through cross-view attention. DETR3D[7] and BEVFormer[13] leverage Transformers to fuse multi-view image features. BEVFormer[13] introduces dense grid-based BEV queries, applies spatial cross-attention to aggregate image features, and uses temporal self-attention to fuse information from past frames.
- *Monocular Detection*: Transformers can also fuse image and depth features in monocular 3D object detection, such as in MonoDTR[14].

Transformers enable semantic alignment and feature aggregation across heterogeneous sensor modalities, such as camera images and LiDAR point clouds[3]:

- *Deep and Intermediate Fusion*: Many recent multi-modal methods employ Transformer architectures with specialized cross-attention mechanisms for feature fusion. TransFusion[15] produces object queries from initial detections and applies cross-attention to LiDAR and image features. BEVFusion[16] provides a unified and efficient modality fusion framework by projecting multi-modal features into a shared BEV space and applying Transformer-based fusion. FUTR3D[17] proposes a unified fusion framework integrating object queries with multi-sensor features, while UVTR[18] fuses object queries with image and LiDAR voxels in a Transformer decoder.
- *Fusion Strategies*: Transformer-based fusion yields strong performance on various benchmarks. RoI heads using Transformer decoders enable effective multi-modal feature fusion during RoI refinement.

The implementation of Transformer models marks a pivotal shift in architecture design for 3D object detection. By enabling sophisticated spatial, temporal, and cross-modal feature interactions, Transformers have driven significant performance gains across LiDAR-based, camera-based, and multi-modal perception systems.

1.1.2 The Paradigm of Edge Computing and the Constraints of Embedded Systems

The Paradigm of Edge Computing and the Necessity of Model Quantization

The paradigm of edge computing is fundamentally driven by the need to relocate computation from remote cloud servers to the proximity of data sources, namely sensors, cameras, IoT terminals, and embedded systems positioned at the network perimeter. This architectural shift has become indispensable in the context of pervasive IoT deployments and the advent of advanced communication technologies such as 5G[19], which collectively generate massive volumes of latency-sensitive data. Traditional cloud-centric architectures are insufficient for meeting the stringent requirements of modern intelligent applications, thereby motivating a decentralized computing model that enables real-time processing near the data origin.

The rapid migration toward edge computing is catalyzed by several key advantages[19]:

- Low Latency: Proximity between edge devices and data sources reduces end-to-end delays from potentially hundreds of milliseconds in cloud scenarios to only a few milliseconds or microseconds, which is critical for latency-sensitive applications such as autonomous driving and interactive VR/AR systems.
- Enhanced Privacy and Security: Processing data locally mitigates privacy risks associated with transmitting sensitive information (e.g., face images, speech, or private documents) to remote cloud servers .
- Network Efficiency and Scalability: By performing computation at the edge, backbone network load is significantly reduced, alleviating congestion and improving scalability as billions of connected devices generate continuous data streams.
- Energy Savings: Local offloading reduces the communication overhead for battery-powered devices, improving overall energy efficiency.

These benefits collectively underscore the necessity of shifting intelligence from centralized cloud infrastructures to distributed edge environments.

Constraints of Embedded Systems on Deep Learning Models

Although edge computing provides the ideal proximity for intelligent processing, the deployment of deep learning models on heterogeneous embedded systems introduces profound challenges. Edge devices vary widely in capability, ranging from GPU-equipped edge servers to highly constrained microcontrollers and compact single-board computers with memory capacities measured in kilobytes to a few megabytes.

The constraints of embedded systems impose fundamental obstacles to executing large-scale deep learning models [19]:

- Limited Computational Resources: Embedded processors lack the high-throughput parallelism required for large matrix operations central to deep neural networks.
- Restricted Memory and Storage: State-of-the-art models such as Transformers may contain millions or even billions of parameters, requiring hundreds of megabytes of storage—far exceeding the capabilities of typical edge devices.
- Energy and Thermal Constraints: Battery-powered and thermally limited devices cannot sustain prolonged high-intensity computation required for deep model inference[20, 19].

In addition to hardware constraints, several operational challenges arise:

- Impracticality of Local Training: Deep model training requires GPU acceleration and substantial memory bandwidth, rendering full local training impossible for most edge devices.
- Communication Overhead in Distributed Settings: When data is distributed across multiple edge nodes, synchronization of parameters or gradients introduces heavy communication costs, especially over limited wireless links[20].
- Inference Latency: Even optimized mobile processors may achieve only 1–2 frames per second on moderately sized models, far below the ultra-low latency requirements of real-time applications such as autonomous navigation [20].

These limitations illustrate a fundamental mismatch between the design assumptions of modern deep neural networks and the constrained environment of embedded edge devices.

The Necessity of Model Quantization for Edge AI

To bridge the gap between the computational demands of deep learning and the limitations of embedded systems, model optimization techniques become indispensable. Among these, **quantization** stands out as a particularly effective strategy.

Quantization reduces the numerical precision of model parameters and activations—for example, from 32-bit floating-point to 8-bit integers—thereby substantially reducing memory footprint and computational complexity. This provides several key benefits:

- Reduced Model Size: An 8-bit quantized model occupies only a fraction of the storage required by its full-precision counterpart.

- Increased Computational Throughput: Lower-precision arithmetic enables faster execution on edge-optimized accelerators.
- Lower Energy Consumption: Reduced computational effort directly translates into power savings, essential for battery-powered devices.

The central challenge of a quantization-focused research effort is to adapt complex, high-fidelity models—particularly large Transformer architectures—to the strict resource budgets of embedded systems while maintaining acceptable accuracy. Successfully achieving this objective will enable real-time, privacy-preserving, and energy-efficient deep learning on edge devices, thereby fulfilling the broader vision of pervasive intelligent Edge AI.

1.2 Thesis Motivation

The rapid advancement of autonomous driving systems is fundamentally reliant on the capabilities of their underlying perception models. State-of-the-art frameworks, such as BEVFusion [16], have demonstrated high accuracy in 3D object detection by fusing data from multiple sensors like cameras and LiDAR into a comprehensive Bird's-Eye View (BEV) representation. However, this high fidelity comes at a steep price: the massive computational and memory demands of these complex neural networks create a critical bottleneck, making their real-time deployment on power and resource constrained vehicle hardware a formidable challenge. This thesis is directly motivated by this pressing conflict between algorithmic performance and practical viability. It seeks to bridge the gap by exploring robust model optimization techniques that can drastically reduce the deployment requirements of large-scale perception models without compromising their accuracy.

1.2.1 The Necessity of Compressing Complex Transformer Models

Transformer architectures have become a cornerstone of modern perception systems, particularly in 3D object detection for autonomous driving. As discussed earlier, these models excel at capturing long-range dependencies, integrating multi-view and multi-modal features, and providing robust spatial–temporal reasoning [3]. However, their expressive power comes with significant computational and memory demands, making naïve deployment on embedded automotive platforms impractical.

Autonomous driving systems require real-time perception with strict latency constraints, processing LiDAR, camera, and radar data at 10-30 Hz to ensure safe decision-making. Yet, state-of-the-art Transformer models—especially those performing bird's-eye-view fusion and multi-sensor integration, such as BEVFusion

[16]—incur high computation and memory overheads due to dense attention operations, high-dimensional embeddings, and repeated spatiotemporal feature fusion. Datasets like nuScenes [21] highlight the complexity of real-world traffic scenes, emphasizing the need for perception models to maintain both accuracy and efficiency under challenging conditions.

Embedded AD platforms, constrained by limited memory, processing throughput, and energy budgets, cannot sustain the full computational load of these large Transformer architectures [19, 20]. Even high-end automotive SoCs must share resources across the full perception-planning-control stack, making efficient model design critical.

Consequently, model compression is essential for deploying Transformer-based perception systems in autonomous vehicles. Quantization, in particular, reduces numerical precision of weights and activations (e.g., from 32-bit floating-point to 8-bit integers), thereby decreasing model size, increasing inference speed, and lowering energy consumption [22]. Complementary techniques, such as pruning and knowledge distillation, can further streamline computation while preserving the geometric and semantic reasoning crucial for robust detection.

Compression is not merely a hardware optimization; it is a prerequisite for real-time, reliable, and scalable autonomous driving perception. Properly compressed Transformer models can meet the stringent latency requirements of embedded AD systems, maintain high accuracy in complex urban environments captured in datasets like nuScenes [21], and support multi-modal fusion frameworks such as BEVFusion [16], ultimately enabling safer and more efficient autonomous mobility.

1.2.2 Develop and Apply a Post-Training Quantization (PTQ) Scheme

This thesis is motivated by the imperative to bridge the gap between high model accuracy and practical deployment efficiency. Specifically, it focuses on **Post-Training Quantization (PTQ)**, a technique that converts a pre-trained FP32 model into a lower-bit representation (e.g., 8-bit integers (INT8)) without the need for model retraining or access to large annotated datasets [22]. The core research objective is to develop and rigorously apply a novel PTQ scheme that minimizes the inherent performance degradation associated with reducing numerical precision. Current PTQ methods often struggle to maintain accuracy, particularly for complex architectures or those with heterogeneous layer sensitivity to quantization noise. The proposed work aims to contribute a robust PTQ methodology, potentially incorporating advanced techniques like channel-wise or layer-wise calibration, adaptive rounding strategies, or non-uniform quantization schemes.

The successful development and application of this PTQ scheme will yield compressed models, leading to several practical benefits. Firstly, the reduced precision directly translates to smaller model memory footprint and faster inference speed

due to the lower bandwidth requirements and the efficiency of integer arithmetic on modern hardware. For instance, moving from FP32 to INT8 typically results in a four-fold reduction in model size and potentially significant speedups on accelerators optimized for INT8 operations. Secondly, this work will be applied to practical, high-impact tasks (e.g., image classification or object detection) to demonstrate the real-world efficacy of the scheme. The overarching goal is to enable the deployment of sophisticated AI capabilities on edge devices.

Design Considerations for PTQ in Transformer-Based 3D Detection

Applying PTQ to Transformers for 3D perception presents unique challenges:

- *Attention Sensitivity*: Self-attention operations are highly sensitive to quantization errors, as small deviations in key, query, or value tensors can propagate and degrade object localization or multi-view feature fusion.
- *Multi-Modal Fusion*: Models integrating LiDAR and camera data, such as BEVFusion [16], rely on precise feature alignment across modalities. Quantization must preserve this cross-modal consistency to avoid performance degradation.
- *Activation Range Calibration*: Dynamic ranges of intermediate activations, particularly in spatiotemporal and cross-attention layers, must be accurately estimated. Improper calibration can introduce clipping or saturation artifacts, reducing detection accuracy.

PTQ Workflow for 3D Perception Models

A typical PTQ workflow for Transformer-based 3D detection involves several key steps:

1. *Layer-Wise Weight Quantization*: Convert model weights to lower-precision integers using per-channel or per-tensor scaling factors. Per-channel scaling is often preferred for attention and linear projection layers due to higher sensitivity in some channels.
2. *Activation Calibration*: Pass a small subset of representative sensor data (e.g., LiDAR point clouds or multi-view images from nuScenes [21]) through the network to record activation statistics and determine optimal quantization ranges.
3. *Integer-Only Inference Conversion*: Replace floating-point operations with integer equivalents where supported by the target hardware, enabling efficient execution on automotive SoCs or edge accelerators.

4. *Evaluation and Fine-Tuning*: Assess the quantized model's accuracy and latency. If significant performance loss is observed, lightweight calibration techniques, such as bias correction or outlier clipping, can be applied without full retraining.

Possible Benefits and Impact on Autonomous Driving

Implementing PTQ may allow Transformer-based 3D detection models to meet real-time inference requirements on embedded AD platforms while maintaining high detection performance:

- *Reduced Memory Footprint*: Lower-precision weights and activations significantly decrease storage requirements, enabling deployment on resource-constrained vehicle SoCs.
- *Faster Inference*: Integer arithmetic operations are more efficient than floating-point, reducing latency and enabling higher frame rates critical for safety-critical perception [19].
- *Energy Efficiency*: Lower computational overhead directly translates to reduced power consumption, improving the operational efficiency of electric or battery-powered vehicles.

By designing and applying a PTQ scheme, autonomous driving systems can leverage the superior perception capabilities of Transformer models, such as BEVFusion [16], in a manner that is compatible with real-world embedded constraints. This would enable high-fidelity 3D detection, multi-modal fusion, and real-time inference on production AD platforms without compromising safety or performance.

1.3 Thesis Outline

In **Chapter 2** i present the primary object detection model studied in this work: BEVFusion, a Transformer-based architecture designed for multi-sensor 3D perception in autonomous driving. This thesis first presents the dataset and evaluation metrics used to assess performance, with particular emphasis on the nuScenes dataset, which provides a comprehensive multimodal benchmark for real-world driving scenarios. The architecture is then described in detail, highlighting its sensor fusion strategies, modular design, and the mechanisms enabling bird's-eye-view representation from LiDAR and camera inputs.

In **Chapter 3** the study then focuses on compressing this complex Transformer model to meet the constraints of embedded systems and edge computing environments. A post-training quantization (PTQ) scheme is developed to reduce model size and computational requirements while preserving detection accuracy. Uniform

quantization results are analyzed across different bit-widths to evaluate their impact on performance and efficiency.

In **Chapter 4** building upon the PTQ scheme, a mixed-precision quantization (MPQ) approach is proposed, employing a genetic algorithm to optimize layer-wise bit assignments. The resulting models are evaluated not only in terms of detection accuracy and memory footprint but also with consideration for the efficiency of the quantization process itself, which must operate under hardware constraints. The work emphasizes developing a quantization methodology that is memory-efficient during both optimization and inference, maintaining performance while remaining mindful of the practical feasibility of deployment on resource-constrained edge devices.

In **Chapter 5**, the thesis concludes with a summary of the key findings and contributions, followed by a discussion of potential directions for future work, including further optimization strategies, deployment on diverse large transformer models.

Chapter 2

The BEVFusion Model

2.1 Dataset and Evaluation Metrics

Choosing a dataset that includes a variety of scenarios, extensive sensor data, and high-quality annotations is essential for assessing autonomous vehicle perception algorithms. The nuScenes [23] dataset was selected because it offers a comprehensive multimodal sensor suite with cameras, lidar, and radar, full 360-degree coverage, diverse conditions such as rain and nighttime, and 3D annotations. Its scale, variety, and inclusion of semantic maps make it particularly well suited for training and evaluating contemporary 3D recognition and tracking algorithms. The dataset and the detection metrics utilized for assessment are described in the following sections.

2.1.1 The NuScenes Dataset for Autonomous Driving

The nuScenes dataset [23] was created to fill major gaps in earlier benchmark datasets for autonomous vehicle (AV) perception. Previous datasets often did not include full sensor suites, multimodal data, or a wide variety of driving conditions and locations. Reliable detection and tracking systems depend on machine learning models that work best when they have synchronized data from range sensors and cameras, which motivated the creation of a dataset combining multiple sensor types. An example nuScenes sample is shown in Figure 2.1.

Multimodal sensing is important because each sensor type has strengths and weaknesses.

- Cameras provide appearance information but have limited depth accuracy.
- Lidar gives accurate 3D position information but produces sparse data.
- Radar can detect objects at long range and estimate their speed, but its spatial resolution is lower.

Using all three together improves performance in difficult conditions and adds redundancy, which is important for safety [23].

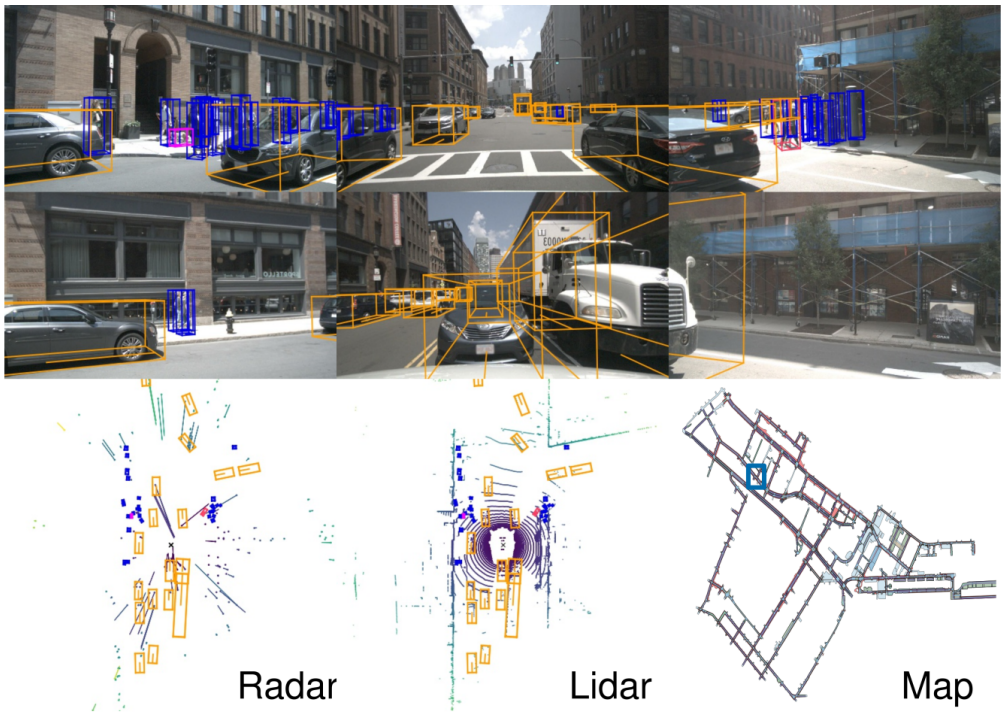


Figure 2.1: Example sample from the nuScenes dataset showing synchronized camera and lidar views with 3D annotations. Image reproduced from the nuScenes dataset [23].

nuScenes is the first AV dataset that includes a complete autonomous-vehicle sensor setup with 6 cameras, 5 radars, and 1 lidar, giving full 360-degree coverage. It also includes map information and a wide range of environments and weather. The dataset contains 1,000 fully annotated 20-second scenes recorded in Boston and Singapore, with 3D bounding boxes for 23 classes and 8 attributes, plus semantic maps. Later releases such as nuScenes-lidarseg add point-level semantic labels, increasing its usefulness for detection, tracking, and segmentation. nuScenes also introduced new 3D detection and tracking metrics and provided a devkit, evaluation code, taxonomy, annotator guidelines, and database schema [23].

Within the AV dataset landscape, nuScenes advanced the field in terms of scale, sensor types, and annotation detail. Earlier datasets such as CamVid[24], Cityscapes[25], Mapillary Vistas[26], BDD100K [27], ApolloScape[28], and D2-City[29] mainly focused on 2D labels for RGB images. Multimodal datasets have been less common because they are expensive to collect and annotate. Important predecessors include:

- KITTI[30]: Included dense lidar scans, front-facing stereo images, GPS/IMU data, and 200k 3D boxes across 22 scenes.

- H3D[31]: Contained 160 crowded scenes with 1.1M 3D boxes and full 360-degree object annotation.
- ApolloScape[28]: Provided static depth maps and 70k 3D boxes.
- KAIST[32]: Included RGB, thermal, RGB stereo, 3D lidar, and GPS/IMU data, offering nighttime coverage but limited size and mostly 2D annotations.

Feature	**nuScenes (2019)**	**KITTI[30] (2012)**	**H3D[31] (2019)**	**Waymo Open[33] (2019)**
Full Sensor Suite (6 Cams, 5 Radar, 1 Lidar)	Yes	No (Lidar/Stereo)	No (Lidar/-Cam)	No (Lidar/-Cam)
Radar Data	Yes (1.3M)	No	No	No
Number of Scenes	1k	22	160	1k
Size (hr)	5.5	1.5	0.77	5.5
3D Boxes	1.4M	200k	1.1M	12M†
Night/Rain Data	Yes/Yes	No/No	No/No	Yes/Yes
Map Layers	11	0	0	0

Later datasets, such as Waymo Open[33], increased annotation volume through higher capture frequency. By releasing real-world data with a full 360-degree multimodal sensor suite, rich annotations, and semantic maps across diverse urban settings and conditions, nuScenes[23] became an important benchmark for large-scale AV perception research.

2.1.2 Evaluation Metrics: NDS and mAP

The nuScenes detection task requires detecting 10 object classes using 3D bounding boxes, as well as predicting attributes (e.g., sitting vs. standing) and velocities. The evaluation metrics are designed to measure many aspects of detection performance, going beyond traditional Intersection over Union (IoU) metrics.

1. **Mean Average Precision (mAP)** Instead of using IoU for matching predictions to ground truth, nuScenes uses a modified Average Precision (AP) metric where a match is defined based on the 2D center distance (d) on the ground plane.
    - Matching Criterion: A prediction matches a ground truth object if the 2D distance between their centers on the ground plane is below a threshold. This removes dependence on object size and orientation.
    - Purpose of Center Distance Matching: This approach helps small objects like pedestrians and bicycles, which would often get 0 IoU even with a small position error.

- Calculation: AP is computed as the normalized area under the Precision–Recall curve, considering only recall and precision values above 10
- Averaging: The final mAP is averaged over distance thresholds $D = 0.5, 1, 2, 4$ meters and over all object classes C.

2. **True Positive (TP) Metrics**
   In addition to mAP, nuScenes reports five True Positive (TP) metrics for every prediction that matches a ground truth box within a 2-meter center distance. Each metric is measured in its natural unit. If a class does not reach at least 10% recall, all TP errors for that class are set to 1. The five TP metrics are:

   (a) **Average Translation Error (ATE):** The 2D Euclidean distance between box centers (meters).

   (b) **Average Scale Error (ASE):** Defined as $1 - \text{IoU}$ after aligning orientation and translation between prediction and ground truth.

   (c) **Average Orientation Error (AOE):** The smallest yaw angle difference (radians), measured over 360° except for barriers (180°).

   (d) **Average Velocity Error (AVE):** The absolute 2D velocity difference (m/s), using the L2 norm.

   (e) **Average Attribute Error (AAE):** Defined as $1 - \text{accuracy}$.

   The Mean TP metric (mTP) is computed as the average of the cumulative mean of each TP metric at recall levels above 10%, averaged over all classes. Metrics that do not apply (e.g., velocity for stationary objects) are not reported.

3. **nuScenes Detection Score (NDS)**
   The NDS combines all detection aspects into a single score, representing both detection performance and prediction quality—something traditional IoU-based mAP cannot fully capture.

   - Formula: NDS is a weighted combination of mAP and the five Mean TP metrics:

$$\text{NDS} = \frac{1}{10}\left[5 \cdot \text{mAP} + \sum_{\text{TP} \in \text{TP metrics}} (1 - \min(1, \text{mTP}))\right]$$

   - Weighting: Half of the score comes from detection performance (mAP), while the other half depends on prediction quality (location, size, orientation, velocity, and attributes).
   - Normalization: Since some TP metrics can exceed 1 (e.g., mATE, mAOE, mAVE), they are capped between 0 and 1 in the NDS calculation.

In this thesis, the effectiveness of the quantized model is evaluated using all the metrics described above. However, special focus is placed on mAP and NDS, as these two give the clearest and most meaningful comparison of detection performance and overall model quality.

The mAP metric measures how well the model can correctly detect and classify objects. This is important for checking if quantization affects the model's main detection ability. A large drop in mAP would show that the model has trouble identifying objects after quantization, which would make it unsuitable.

The NDS, on the other hand, gives a broader view by combining mAP with the quality of the predicted bounding boxes, including their position, size, orientation, speed, and other attributes. All the other metrics described above are included in NDS, but they are also compared separately to better understand the specific effects of quantization on each aspect of the model. This is especially important in autonomous driving, where accurate location and movement predictions are needed for safe decisions. By looking at both detection accuracy and prediction quality, NDS gives a more complete evaluation of the model.

Focusing on mAP and NDS allows for a clear understanding of how quantization affects the model, not just in detection performance, but also in its ability to produce reliable and useful predictions in a complex environment like nuScenes.

2.2 BEVFusion Architecture Overview

2.2.1 Bird's Eye View (BEV) and Sensor Fusion for 3D Perception

Accurate and reliable perception in autonomous driving requires robust multi-sensor fusion, as autonomous systems rely on complementary signals from cameras, LiDARs, and radars. Cameras provide rich semantic information, LiDAR captures precise 3D geometry, and radars provide instant velocity measurements. However, these sensors operate in fundamentally different modalities—perspective view for cameras versus 3D point clouds for LiDAR—necessitating a unified representation suitable for multi-task, multi-modal feature fusion.

Traditional sensor fusion approaches generally fall into two categories, both of which introduce significant information loss:

1. LiDAR-to-Camera Projection (Geometric-Lossy): Projects LiDAR point clouds onto the camera plane to create sparse RGB-D images, which can distort spatial relationships in 3D.

2. Camera-to-LiDAR Projection (Semantic-Lossy / Point-Level Fusion): Augments LiDAR points with camera features, but only a small portion of camera information is used, limiting semantic richness.

BEVFusion addresses these limitations by performing fusion in a shared Bird's-Eye View (BEV) space, treating geometric and semantic information equally. Advantages include:

- Information Preservation: Maintains geometric structure and semantic density.
- Geometric Fidelity: LiDAR-to-BEV projection avoids distortions.
- Semantic Density: Camera features are cast into BEV using predicted depth distributions.
- Task Agnosticism: BEV representation supports diverse 3D perception tasks.

The transformation of multi-view camera features from perspective view to BEV is non-trivial due to depth ambiguity. Following prior work (LSS), BEVFusion predicts a discrete depth distribution for each pixel and uses it to project camera features into BEV space.

Once LiDAR and camera features are converted into BEV, they are fused elementwise and refined with a convolution-based BEV encoder to compensate for local misalignments.

Fused BEV features are passed to task-specific heads:

- 3D Object Detection: Uses a class-specific heatmap to predict object centers, with regression heads estimating size, rotation, and velocity.
- BEV Map Segmentation: Treated as multiple binary segmentation problems to handle overlapping map classes.

2.2.2 Key Modules: Input Encoders (Camera/LiDAR), Feature Fusion, and Detection Head

BEVFusion uses a unified BEV representation to integrate multi-modal features for multi-task 3D perception. Its core modules include:

- Input Encoders:
  - Camera Encoder: Extracts semantic features from multi-view RGB images using a backbone with FPN.
  - LiDAR Encoder: Extracts geometric features from voxelized point clouds and projects them into BEV by flattening along the z-axis.
- Camera-to-BEV Transformation:
  - Predicts discrete depth distributions for each pixel.

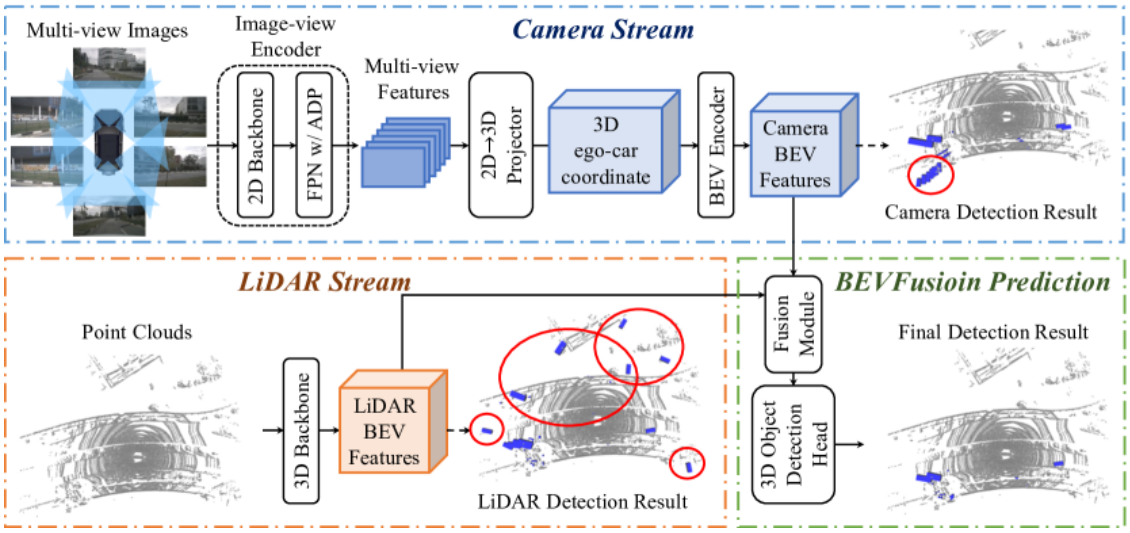


Figure 2.2: General architecture of BEVFusion, illustrating the camera encoder, LiDAR encoder, view transformer, BEV fusion module, and task-specific heads. Adapted from BEVFusion [16].

  - Projects features into 3D space according to depth.
  - Aggregates into BEV grids using BEV pooling.

- Feature Fusion (BEV Encoder): Fuses LiDAR and camera BEV features elementwise. The BEV encoder refines fused features and corrects for misalignments.

- Task-Specific Heads:

  - 3D Object Detection: Center-based formulation with regression of object attributes.
  - BEV Map Segmentation: Independent binary segmentation for each map class.

This modular design allows BEVFusion to efficiently support multiple 3D perception tasks using a shared BEV representationa as shown in Figure 2.2.

2.2.3 Detailed Structural Breakdown, Referencing the `mmdetection3d` Implementation

The BEVFusion [16] architecture in this configuration is a multimodal 3D detection model that processes both camera and LiDAR data, fusing them into a unified Bird's-Eye-View (BEV) representation. Its design integrates a Swin Transformer

[34] for image feature extraction and SECOND [9] components for LiDAR feature processing, following implementations in `mmdetection3d` [35].

Input Encoders The input features from Camera (Image) and LiDAR (Point Cloud) are processed by separate encoders:

- Camera Encoder
  Composed of `img_backbone` and `img_neck`

  The camera branch relies on a Swin Transformer [34] backbone followed by a GeneralizedLSSFPN neck.

  - Backbone (`SwinTransformer`)
    The Swin Transformer provides hierarchical multi-scale image features using shifted window self-attention.
    The configuration used here corresponds to the Swin-T variant:

    $$\text{embed\_dims} = 96, \quad \text{depths} = [2, 2, 6, 2], \quad \text{num\_heads} = [3, 6, 12, 24]$$

  - Neck (`GeneralizedLSSFPN`)
    Produces unified 256-channel features for view transformation.

- LiDAR Encoder
  Composed of `pts_voxel_encoder`, `pts_middle_encoder`, `pts_backbone` and `pts_neck`

  The LiDAR branch follows SECOND [9], a voxel-based 3D detection architecture.

  - Voxel Encoder (`HardSimpleVFE`)
    Implements Voxel Feature Encoding (VFE) through pointwise processing and voxel-level aggregation.
  - Middle Encoder (`BEVFusionSparseEncoder`)
    A sparse 3D CNN module that collapses vertical structure into a dense 2D BEV representation.
  - Backbone (`SECOND`)
    A 2D convolutional backbone operating on the BEV feature map.
  - Neck (`SECONDFPN`)
    A multi-scale FPN module producing BEV features.

Camera-to-BEV Transformation

- View Transformer (`DepthLSSTransform`)

  Implements the Lift, Splat, Shoot mechanism to convert 2D image features into a pseudo-BEV representation based on predicted depth.

BEV Feature Fusion and Refinement

- Fusion Layer (`ConvFuser`)

  Concatenates and refines camera and LiDAR BEV features, producing a unified 256-channel representation.

Task Head (3D Detection)

- Detection Head (`TransFusionHead`)

  A Transformer-based proposal refinement module predicting class scores, centers, dimensions, heights, rotations, and velocities.

Chapter 3

Quantization of BEVFusion

3.1 Neural Network Quantization

3.1.1 Floating-Point Precision vs. Integer Precision

The transition from floating-point (FP) precision to integer precision is a cornerstone of modern neural network optimization, particularly motivated by the necessity for efficient and accurate inference on resource-constrained mobile and edge devices. While current state-of-the-art Convolutional Neural Networks (CNNs) have historically been primarily appraised according to classification or detection accuracy, leading to architectures optimized without primary regard for computational efficiency, successful deployment on platforms such as smartphones or drones mandates small model sizes and low latency [36, 37].

The fundamental difference lies in how numerical values are represented and manipulated during computation, offering stark trade-offs in efficiency, hardware requirements, and accuracy preservation.

Floating-Point Precision (FP32)

Floating-point representations, typically 32-bit (FP32), serve as the conventional precision for training deep neural networks, enabling high representational capacity and allowing weights and biases to be easily adjusted by small gradient amounts.

- High Accuracy and Training Standard: FP32 is the precision in which neural networks are typically trained and initially represented. Models stored in FP32 are often large and over-parameterized by design to extract marginal accuracy improvements.
- Computational Cost: FP32 operations are computationally intensive and energy-consuming. Moving from 32-bit FP to lower integer precision significantly reduces memory footprint and latency. For example, a 32-bit floating-point

addition or multiplication requires substantially more energy and area compared to its 8-bit integer counterpart. Performing inference in FP32 requires processing elements and accumulators that support floating-point logic.

- Limited Numerical Precision in Training: While FP32 is standard, innovations have introduced lower floating-point precisions, such as half-precision (FP16), to accelerate training throughput in AI accelerators, although moving below half-precision has proven challenging without significant tuning [37].

Integer Precision (Fixed-Point Arithmetic)

Integer precision, often referred to as fixed-point arithmetic in the context of inference, involves representing real-valued tensors (weights and activations) as low bit-width integer values. This approach yields significant benefits in efficiency, speed, and hardware compatibility.

Efficiency and Computational Gains

Integer-only arithmetic is inherently more efficient than floating-point inference. The computational cost and memory transfer requirements decrease dramatically when moving to integer representations:

1. Memory and Latency Reduction: Moving from 32-bit FP to 8-bit integer (INT8) decreases the memory overhead for storing tensors by a factor of 4. Critically, the computational cost for matrix multiplication can reduce quadratically, by a factor of 16. Overall, reductions of 4x to 8x in memory footprint and latency are often realized in practice when using low-precision fixed integer values.

2. Energy Efficiency: Low-precision logic offers exponentially better energy efficiency. For instance, INT8 addition is approximately 30 times more energy efficient and 116 times more area efficient than FP32 addition, based on 45nm technology estimates.

3. Hardware Optimization: Integer-only inference leverages fast integer-arithmetic circuits common in CPUs. This is crucial for edge devices, many of which lack dedicated floating-point units or offer highly optimized integer arithmetic capabilities [37].

4. Integer-Arithmetic-Only Inference: This methodology mandates that all computations, including multiplication, addition, and accumulation, be performed using low-precision integer arithmetic without requiring floating-point dequantization during inference. Standard implementation often quantizes weights and activations as 8-bit integers, while retaining bias vectors as 32-bit integers to minimize quantization error propagation [37].

Implementation Complexity and Granularity

Integer quantization schemes, particularly the commonly used uniform affine quantization, rely on three parameters—scale factor (s), zero-point (z), and bit-width (b)—to map real values (r) to an integer grid ($Q(r)$) such that $r = S(q - Z)$.

- Accumulator Precision: While inputs (weights and activations) are quantized to low bit-widths (e.g., INT8), the accumulator utilized in the Multiply-Accumulate (MAC) operation is typically maintained at a higher bit-width, such as 32-bits, to prevent overflow as numerous products accumulate during computation.
- Granularity: Integer quantization can be applied per-tensor or per-channel. Per-channel quantization, where each output channel of a weight tensor receives separate quantization parameters, offers better quantization resolution and accuracy, especially when weight distributions vary widely across channels. Per-tensor quantization is simpler to implement because all accumulators use the same scale factor during multiplication [37].
- Uniform vs. Non-Uniform: Uniform quantization, where resulting quantized values are uniformly spaced, is the *de facto* method because it maps efficiently to general computation hardware. Non-uniform quantization can theoretically capture signal distributions better but is generally harder to deploy efficiently.

Accuracy Trade-offs in Integer Precision

While INT8 quantization is effective, lowering the bit-width increases quantization noise, which can lead to significant accuracy degradation.

- INT8 Performance: Integer-only quantization can preserve end-to-end model accuracy post-quantization. INT8 schemes often result in near floating-point accuracy. Furthermore, integer quantized MobileNets have demonstrated achieving higher accuracy compared to FP MobileNets given the same runtime budget on certain hardware[37].
- Low-Bit Integer Performance (INT4 and Below): Moving below INT8, particularly to 4-bit weights and activations (W4A4), presents greater challenges.
  - Extreme Quantization: Binarization (1-bit) and Ternarization (2-bit) offer the maximum memory reduction (up to 32x for binarization) and can be computed efficiently using specialized bit-wise operations (like XNOR followed by bit-counting). However, achieving high accuracy with extreme quantization typically requires intensive tuning and advanced techniques.

In summary, floating-point precision serves as the fidelity baseline required for robust training, while low-precision integer arithmetic (INT8 being the current

optimal trade-off point) provides the necessary computational and energy efficiency for mass deployment on modern hardware.

3.1.2 Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT)

Since the quantization process introduces noise that can degrade accuracy, two principal strategies(Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT)) have been developed to mitigate this impact while leveraging the hardware efficiency of integer arithmetic.

Post-Training Quantization (PTQ)

Post-Training Quantization (PTQ) is a lightweight and computationally efficient methodology used to convert a pre-trained FP32 network into a fixed-point network without requiring the original training pipeline.

1. Computational Efficiency: The overhead of PTQ is very low and often negligible, as it avoids end-to-end training.
2. Data Requirements: PTQ methods can be data-free or may only require a small, unlabeled calibration set. This approach, known as Zero-Shot Quantization (ZSQ), is particularly important when access to the original training data is limited due to size, proprietary concerns, or security [37].
3. Accuracy Threshold: PTQ is generally sufficient for achieving high accuracy with 8-bit quantization of weights and activations (W8A8)..

PTQ relies on effective techniques, often optimized in a *pipeline*, to set quantization parameters and correct errors introduced by fixed rounding and clipping.

Quantization-Aware Training (QAT)

Quantization-Aware Training (QAT) involves fine-tuning a network while modeling the effects of quantization noise within the training loop.

QAT is typically used when aiming for aggressive low-bit precision (e.g., 4-bit weights and activations, W4A4) where PTQ techniques alone are insufficient to mitigate the large quantization error.

1. Requirements: QAT necessitates longer training times, access to labeled training data, and hyperparameter search, incurring higher costs compared to PTQ.
2. Mechanism (Simulated Quantization): In the forward pass, QAT simulates fixed-point inference on floating-point hardware by injecting quantization blocks into the graph (referred to as fake quantization). This induces quantization effects, allowing the network parameters to adapt [36].

3. Backpropagation and STE: The core challenge in QAT is that the rounding operation used in quantization is non-differentiable. To propagate gradients, QAT utilizes the Straight-Through Estimator (STE), which approximates the gradient of the rounding operator as 1. This allows the training to adjust the weights, even though the forward pass involves discrete integer values [36].

QAT can be implemented and refined following the techniques:

1. Batch Normalization Folding: During QAT, Batch Normalization (BN) must be folded into the preceding linear layer (convolutional or fully connected) to accurately simulate efficient inference behavior. Static folding, where the BN scale and offset are permanently absorbed into the weights and biases at the start of QAT, is a common and effective approach. For per-channel quantization, BN parameters can be merged into the per-channel scale factor, allowing the BN layer to remain intact during training and simplifying deployment conversion.

2. Learnable Quantization Parameters: In modern QAT pipelines, the quantization parameters (scale factor s and zero-point z) are made learnable (real numbers in the training graph) and optimized alongside the weights using gradient descent. Learning these parameters directly leads to higher performance, especially for low-bit quantization, although it may require adjusting the learning rate relative to the network weights.

3. Initialization: Although QAT can eventually close the accuracy gap, initializing the process using successful PTQ techniques (like MSE range estimation) is crucial.

Comparative Summary and Use Case

The choice between PTQ and QAT hinges on the required accuracy and the resource constraints of the application.

Feature	Post-Training Quantization (PTQ)	Quantization-Aware Training (QAT)
Training/Fine-tuning	No re-training required	Requires extensive fine-tuning/re-training
Computational Cost	Very low/negligible	High (same costs as network training)
Data Requirements	Data-free (ZSQ) or small calibration set	Requires full labeled training data set
Core Mechanism	Optimizes fixed parameters and applies corrections	Simulates quantization noise during training using STE
Optimal Use Case	Achieving W8A8 with minimal accuracy loss	Achieving aggressive low-bit precision (e.g., W4A4)
Accuracy (W4A8)	Often incurs larger drops	Generally maintains near FP32 accuracy

In practical deployment, PTQ pipelines are typically the first approach utilized to quickly deploy highly efficient W8A8 models. If the accuracy requirements demand even lower bit-widths (W4A8 or W4A4), the investment in QAT becomes necessary to recover performance.

3.1.3 Linear Quantization: Scale ($\mathcal{S}$) and Zero Point ($\mathcal{Z}$)

Linear quantization, often referred to as uniform affine quantization or asymmetric quantization, is the standard numerical representation scheme employed in deep learning for transforming floating-point models into fixed-point formats.

The defining characteristic of this quantization method is the affine mapping it establishes between a continuous real-valued domain (r or x) and a finite, discrete integer domain (q or x_{int}), governed primarily by the scale factor (S or s) and the zero-point (Z or z).

Affine Mapping

The fundamental correspondence between the real value r (or x) and the quantized integer value q (or x_{int}) is defined by the affine relationship

$$r = S(q - Z)$$

This equation is implemented in two stages: quantization (mapping r to q) and dequantization (mapping q back to an approximation $\hat{r}$ or $\hat{x}$).

Scale Factor

The scale factor S (or s) is a crucial parameter defining the resolution of the quantization grid. It is typically represented as an arbitrary positive real number and is often stored as a floating-point quantity in software. It fundamentally specifies the step-size of the quantizer.

The scale factor relates the clipping range $[\alpha, \beta]$ and the target bit-width b through

$$S = \frac{\beta - \alpha}{2^b - 1}$$

During inference implementation, the scale factor enables integer-only arithmetic. By assigning a single scale factor (s_w for weights, s_x for activations) to an entire tensor (or channel), these floating-point scales can be factored out of the multiply-accumulate (MAC) summation, allowing the main matrix multiplication operation to be performed purely using low-precision integer arithmetic.

A notable optimization occurs when the scale factor is restricted to a power-of-two, $s = 2^{-k}$. This restriction allows the scaling operation to be implemented

through efficient bit-shifting, potentially increasing hardware efficiency, although it may complicate the trade-off between rounding and clipping errors due to the restricted expressiveness of the scale factor [37, 36].

Zero-Point

The zero-point Z (or z) is an integer value that ensures computational fidelity, particularly concerning the exact representation of zero. It is defined as the quantized value q that corresponds exactly to the real value $r = 0$. It is an integer of the same type as the quantized values q.

Exact representation of zero is crucial for two reasons:

1. Zero padding: Efficient implementation of neural network operators often requires zero-padding of arrays around boundaries. If zero could not be exactly represented, this padding would introduce quantization errors.

2. Activation functions: Operations such as ReLU, which clamp negative values to zero, require the exact representation of zero to avoid inducing quantization error at the layer output.

The quantization process maps a real value x to a clamped integer x_{int}:

$$x_{int} = \text{clamp}\left(\left\lfloor \frac{x}{s} \right\rceil + z;\, 0,\, 2^b - 1\right)$$

The dequantization process approximates the original real value $\hat{x}$:

$$\hat{x} \approx x = s(x_{int} - z)$$

Symmetric and Asymmetric Quantization

The selection of the zero-point Z determines whether the quantization is symmetric or asymmetric, leading to different trade-offs in accuracy and computational overhead.

In asymmetric quantization, the clipping range $[\alpha, \beta]$ is not necessarily symmetric around zero, and consequently the zero-point Z is calculated to be a non-zero integer.

Benefits include increased expressiveness and a tighter clipping range, especially when the input data distribution is skewed, such as non-negative ReLU activations. However, if both weights and activations use asymmetric quantization (i.e., $Z_w \neq 0$ and $Z_x \neq 0$), the resulting multiply-accumulate operation introduces an input-dependent residual term that must be calculated during inference [36]. This may lead to significant overhead in latency and power, making fully asymmetric quantization often inefficient for deployment.

Symmetric quantization simplifies the scheme by restricting the zero-point to $Z = 0$. This reduces the computational overhead associated with managing the zero-point offset during the MAC operation and simplifies implementation.

In practice, it is common to use symmetric quantization for weights (where $Z_w = 0$) and asymmetric quantization for activations (where $Z_x \neq 0$). This combination avoids the costly data-dependent overhead introduced by two non-zero zero-points while still allowing activations, which are frequently non-symmetric, to use a tight non-symmetric range.

In this work, we adopt a Post-Training Quantization (PTQ) approach using asymmetric, per-channel quantization for weights and activations, as it provides a tighter dynamic range representation while maintaining deployment efficiency.

3.2 The Proposed Three-Stage PTQ Technique for BEVFusion

This work introduces a three-stage PTQ technique designed specifically for BEVFusion. The proposed method combines lightweight per-module calibration, fine-grained quantization parameter refinement, and a mixed-precision search procedure to achieve stable and accurate low-bit deployment. Each stage is engineered to minimize memory usage while progressively improving the alignment between floating-point and quantized outputs.

- Stage 1 establishes robust baseline quantization parameters via per-module min/max calibration, avoiding the memory bottlenecks of model-level statistics.
- Stage 2 refines these parameters using per-module multiplicative adjustment factors, optimized to maximize output similarity with minimal overhead.
- Stage 3 employs a multi-objective genetic algorithm to explore mixed-precision configurations that balance accuracy and computational efficiency.

Together, these stages form a cohesive PTQ pipeline that is scalable and hardware-friendly. The following subsections describe each stage in detail.

Target Modules and Scope

The calibration procedure targets all computationally intensive layers within the network, specifically `Conv2D`, `Linear`, and `MatMul` layers. These layers dominate both the compute and memory footprint of BEVFusion [16], making accurate initial quantization essential for effective downstream low-bit deployment.

3.2.1 Stage 1 (Baseline Calibration): Min/Max Statistics for Initial $\mathcal{S}$ and $\mathcal{Z}$ Determination

The calibration strategy used in this stage takes direct inspiration from the PTQ4ViT framework [38], which demonstrated that careful per-channel min/max estimation is crucial for stable low-bit post-training quantization.

In this work the procedure is performed per-module (specifically, at the level of the model's child modules, referred to here as the *main modules*)and can be applied on the whole model. This design choice significantly reduces memory requirements: rather than storing statistics for entire network-wide tensors, only one module at a time is processed, making the method scalable to BEVFusion's large feature maps, at the cost of increasing the calibration time.

Stage 1 of BEVFusion quantization focuses on establishing baseline quantization parameters, specifically the scale ($\mathcal{S}$) and zero-point ($\mathcal{Z}$), for both activations and weights. This initialization is critical to ensure that subsequent integer quantization preserves the dynamic range and numerical fidelity of the network's features, while respecting hardware constraints.

Min/Max Collection Methodology

Due to hardware limitations, storing full activation tensors across the network is infeasible. Instead, only the channel-wise minimum and maximum values are retained and updated during each forward pass. For a given layer, if x_c represents the set of activation values in channel c for a batch, the running min and max are updated as:

$$\min_c \leftarrow \min(\min_c, \min(x_c)), \quad \max_c \leftarrow \max(\max_c, \max(x_c))$$

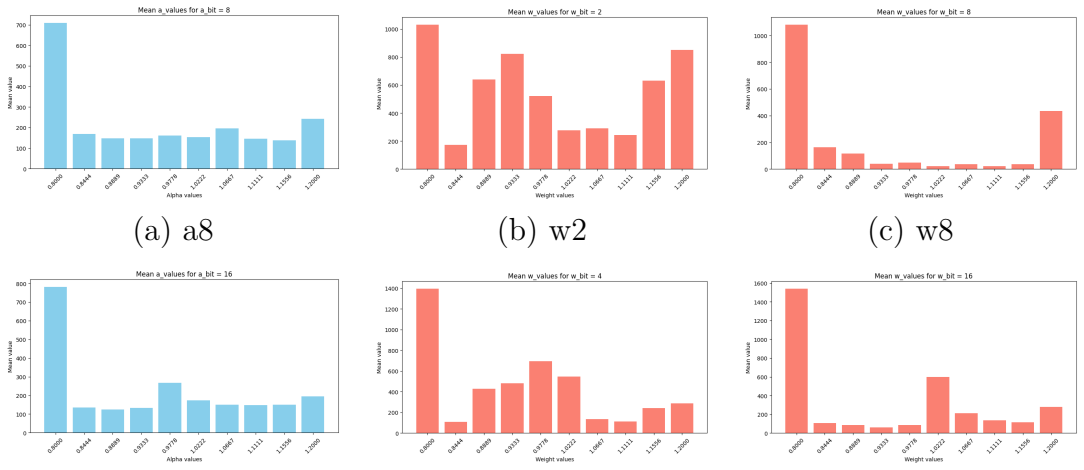
This approach ensures that the dynamic range of each channel is adequately captured without exceeding memory constraints, while enabling efficient computation of scale and zero-point.

Scale and Zero-Point Computation

Once the channel-wise min/max statistics are collected, the scale $\mathcal{S}$ and zero-point $\mathcal{Z}$ are computed based on the target bit-width b using uniform affine quantization:

$$\mathcal{S}_c = \frac{\max_c - \min_c}{2^b - 1}, \quad \mathcal{Z}_c = \text{round}\left(-\frac{\min_c}{\mathcal{S}_c}\right)$$

This calculation supports 16-bit and 8-bit quantization for activations, while weight quantization can go down to 2 bits, allowing extreme compression for memory- or energy-constrained deployments. The quantization parameters computed in this stage serve as the initial values for further fine-tuning and quantization-aware training in subsequent stages.



(a) a8 (b) w2 (c) w8

(d) a16 (e) w4 (f) w16

Figure 3.1: Distributions of optimized α values for all activation and weight bit-widths. Each histogram reflects the frequency of α candidates selected as minimizing MSE during Stage 2 optimization.

Efficiency Considerations

Storing only the per-channel min/max values minimizes memory overhead during calibration. Moreover, the incremental update at each forward pass avoids the need for retaining entire feature maps, which is particularly advantageous for high-resolution BEVFusion inputs. This strategy ensures that calibration is feasible even on GPUs or accelerators with limited memory while still providing a faithful approximation of the activation distribution for subsequent integer quantization.

This baseline calibration stage sets the foundation for accurate, low-bit quantization of BEVFusion while balancing memory efficiency and computational feasibility.

3.2.2 Stage 2 (Per-Module Optimization): Search for Two α Factors to Maximize Output Similarity

This optimization stage also draws inspiration from PTQ4ViT [38], which introduced the concept of refining quantization parameters through multiplicative adjustment factors. All computations, including forward passes and α-selection, are localized to individual layers and grouped in main modules, further reducing memory overhead and enabling the method to scale to BEVFusion's computationally heavy architecture.

Stage 2 of BEVFusion quantization builds upon the baseline calibration established in Stage 1. After initial scale ($\mathcal{S}$) and zero-point ($\mathcal{Z}$) values are determined, this stage focuses on per-layer optimization of the quantization parameters by introducing multiplicative adjustment factors α_w and α_a for weights and activations,

respectively. The objective is to minimize the discrepancy between the original floating-point outputs and their quantized counterparts, thereby preserving feature fidelity during low-bit deployment.

Scope

For each layer, both weights and activations are refined independently through α_w and α_a, allowing the quantization function to adapt to layer-specific dynamic ranges that were only coarsely captured in Stage 1. Importantly, each (α_w, α_a) pair is associated with a specific weight/activation bit-width configuration (e.g., W2A8, W4A8), so the optimization is performed separately for each quantization scenario.

Optimization Procedure and MSE Objective

For each batch of representative input samples, the floating-point output Y_{fp} and the quantized output Y_{q} (after applying $\alpha\mathcal{S}$ and $\mathcal{Z}$) are compared using a mean squared error (MSE) metric:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\Big(Y_{\text{fp},i} - Y_{\text{q},i}\Big)^2$$

The search for optimal α_w and α_a is performed via a discrete grid search. Each candidate value in the search grid is applied multiplicatively to the corresponding weight or activation scale $\mathcal{S}$, and the resulting MSE is recorded. To manage memory constraints, only the occurrence counts of each discrete α value that produces minimal MSE in a batch are retained. The optimization is repeated independently for each quantization bit-width pair (e.g., W2A8, W4A8), ensuring that the selected α factors are tailored to the corresponding low-bit configuration.

Weighted Average Application

After iterating over all batches, the final α_w and α_a applied to each layer are computed as weighted averages of the candidates, using the frequency of occurrence as weights:

$$\alpha_{\text{final}} = \frac{\sum_i(\text{occurrence}_i \cdot \alpha_i)}{\sum_i \text{occurrence}_i}$$

This ensures that the most consistently effective α values across batches have greater influence, while still respecting memory limitations by avoiding full-batch retention of intermediate outputs.

Efficiency Considerations

By leveraging a discrete search grid and recording only occurrence counts, this procedure avoids the need to store full activation tensors for each candidate α, which

is critical for high-resolution BEVFusion inputs. The method strikes a balance between fine-grained per-module quantization optimization and practical memory usage.

Analysis of α Distributions Across Bit-Widths

To better understand the behavior of the multiplicative factors α_w and α_a optimized in Stage 2, six histograms are provided (Figures 3.1). Each α rescales the quantization scale, so the distribution reveals how much correction was required beyond the Stage 1 min/max initialization.

Across all bit-widths, the most frequent value is $\alpha = 0.8$, indicating that the Stage 1 dynamic range tends to be slightly overestimated and benefits from uniform compression. Below is the interpretation for each bit-width.

Activation 8-bit (a8). A dominant peak at 0.8 shows that 8-bit activations only require moderate scale reduction. Since 8-bit quantization already preserves fine granularity, small corrections suffice.

Activation 16-bit (a16). Despite near-lossless 16-bit quantization, this histogram again peaks sharply at 0.8, suggesting that the Stage 1 range overestimation is consistent across bit-widths.

Weight 8-bit (w8). A main peak at 0.8 and a secondary peak at 1.2 reveal module heterogeneity. Most modules benefit from shrinking the scale, but some require expansion due to an underestimated range in Stage 1.

Weight 4-bit (w4). The distribution peaks at 0.8 but also forms a smooth, bell-shaped cluster centered near 0.98, reflecting limited precision at 4 bits: many modules need only minimal tuning, while others favor scale reduction.

Weight 16-bit (w16). A strong peak at 0.8 with smaller peaks at 1.02 and 1.2 indicates fine-grained module-specific adjustments enabled by 16-bit precision.

Weight 2-bit (w2). The noisiest distribution due to extreme quantization constraints. Although irregular, the highest bin still occurs at 0.8, showing that range compression is beneficial even in ultra-low precision.

3.2.3 Stage 3 (MPQ Search): Introduction to the Genetic Algorithm Search Space

Stage 3 of BEVFusion quantization uses a multi-objective genetic algorithm (NSGA-II) to explore the combinatorial search space of Mixed-Precision Quantization

(MPQ) assignments across all quantized modules. This stage can be indipendent from α_w and α_a factors optimized in Stage 2 and focuses on identifying promising bit-width configurations (e.g., W2A8, W4A8) that balance model fidelity and resource efficiency.

Each module can select from a set of candidate bit-widths for weights and activations, forming a large search space. NSGA-II is chosen because it is significantly faster than traditional MPQ optimization techniques, though it does not guarantee the globally optimal solution. Instead, it efficiently finds configurations that tend toward optimality by evolving populations of candidate solutions using selection, crossover, and mutation guided by multi-objective criteria such as accuracy preservation and memory or computational cost.

The output of this stage is a set of candidate MPQ assignments per module, each may be associated with the previously optimized α factors. Detailed GA metrics and MPQ results are reported in the following chapter.

3.3 Results of Uniform ("Flat") Quantization

This section reports the performance of BEVFusion under uniform quantization settings, where all layers of the network use the same bit-width for weights and activations. The following notation is used:

- **NQ:** Non-quantized floating-point model.
- **W*b*A*a*Q:** Uniform quantization with b-bit weights and a-bit activations. Activations are always 8-bit for 2/4/8-bit weights, and 16-bit for 16-bit weights.
- **W*b*A*a*Q with α (XAQ):** Same configuration with per-module scaling factors α_w and α_a.

The evaluation uses the standard BEVFusion metrics (NDS, mAP, mATE, mASE, mAOE, mAVE, mAAE) and two model-size indicators (total bits and relative compression ratio).

The goal of this section is to highlight how accuracy changes across different uniform quantization depths, with particular emphasis on the practically relevant **8-bit** and **4-bit** models.

3.3.1 Performance of the *int*8 Uniform Model

Table 3.1 reports the results for the 8-bit model (W8A8). This configuration delivers accuracy nearly identical to the non-quantized baseline, while reducing the model size by a factor of **4**×. Because the accuracy drop is negligible, 8-bit quantization can be considered a *"free" compression step* for BEVFusion.

Overall, the *int*8 configuration demonstrates that BEVFusion can be quantized to 8 bits with negligible impact on performance.

Table 3.1: Performance metrics of the uniform *int*8 model.

	NDS	mAP	mATE	mASE	mAOE	mAVE	mAAE	total	relative	rapport
NQ	0.5605	0.6130	0.2969	0.7103	1.5530	0.2671	0.1861	1.17e+09	1.00	1
8Q	0.5612	0.6140	0.2966	0.7100	1.5547	0.2648	0.1865	2.92e+08	0.25	4
8AQ	0.5554	0.6047	0.3000	0.7107	1.5552	0.2721	0.1864	2.92e+08	0.25	4

3.3.2 Performance of the *int*4 Uniform Model

Table 3.2: Performance metrics of the uniform *int*4 model.

	NDS	mAP	mATE	mASE	mAOE	mAVE	mAAE	total	relative	rapport
NQ	0.5605	0.6130	0.2969	0.7103	1.5530	0.2671	0.1861	1.17e+09	1.00	1
4Q	0.5545	0.6038	0.3002	0.7087	1.5577	0.2841	0.1814	1.46e+08	0.13	8
4AQ	0.5547	0.6043	0.3040	0.7095	1.5610	0.2790	0.1824	1.46e+08	0.13	8

Table 3.2 shows the results for the 4-bit model (W4A8). This configuration compresses the model by **8**× compared to floating-point while suffering only a mild reduction in NDS and mAP (approximately 1.7%). This makes 4-bit uniform quantization an attractive operating point for memory-constrained deployments.

The results confirm that 4-bit quantization provides a strong compression-accuracy trade-off while maintaining acceptable performance.

3.3.3 Summary of Other Bit-Widths: *int*16 and Ternary (*int*2)

Table 3.3: Global comparison of uniform quantization configurations.

	NDS	mAP	mATE	mASE	mAOE	mAVE	mAAE	total	relative	rapport
NQ	0.5605	0.6130	0.2970	0.710	1.553	0.267	0.186	1.17e+09	1.00	1
2Q	0.0349	0.0249	0.923	0.933	1.056	0.976	0.944	7.31e+07	0.06	16
2AQ	0.0903	0.0340	0.707	0.863	1.170	0.948	0.750	7.31e+07	0.06	16
4Q	0.5545	0.6038	0.300	0.709	1.558	0.284	0.181	1.46e+08	0.13	8
4AQ	0.5547	0.6043	0.304	0.710	1.561	0.279	0.182	1.46e+08	0.13	8
8Q	0.5612	0.6140	0.297	0.710	1.555	0.265	0.187	2.92e+08	0.25	4
8AQ	0.5554	0.6047	0.300	0.711	1.555	0.272	0.186	2.92e+08	0.25	4
16Q	0.5602	0.6124	0.297	0.710	1.554	0.267	0.186	5.84e+08	0.50	2
16AQ	0.5606	0.6129	0.296	0.710	1.552	0.267	0.186	5.84e+08	0.50	2

The 16-bit configuration (W16A16) shows accuracy nearly identical to the floating-point baseline while halving the model size. It serves mainly as a sanity check for

the quantization pipeline.

The ternary 2-bit configuration (W2A8) yields a 16× reduction in model size but introduces severe accuracy degradation, even with α correction as shown in Table 3.3. This confirms that uniform 2-bit quantization is unsuitable for BEVFusion and motivates mixed-precision strategies evaluated later.

3.3.4 Per Class Comparison Across All Uniform Bit-Widths

Table 3.4: Per Class metrics for non-quantized model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.8520	0.1780	0.7500	1.5870	0.2710	0.1960
truck	0.5750	0.3450	0.7940	1.6080	0.2480	0.2270
bus	0.7040	0.3420	0.8610	1.5400	0.4510	0.2630
trailer	0.4000	0.5260	0.8630	1.6110	0.2190	0.1470
construction_vehicle	0.2370	0.7560	0.6880	1.5870	0.1180	0.3140
pedestrian	0.8410	0.1360	0.3290	1.5770	0.2310	0.0870
motorcycle	0.6610	0.1980	0.7900	1.5260	0.3730	0.2450
bicycle	0.5150	0.1630	0.8080	1.6340	0.2260	0.0100
traffic_cone	0.6720	0.1330	0.3310	-	-	-
barrier	0.6730	0.1930	0.8880	1.3070	-	-

Table 3.5: Per Class metrics for ternary($int2$) uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
truck	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
bus	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
trailer	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
construction_vehicle	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
pedestrian	0.2490	0.2270	0.3330	1.5030	0.8080	0.5510
motorcycle	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
bicycle	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
traffic_cone	0.0000	1.0000	1.0000	-	-	-
barrier	0.0000	1.0000	1.0000	1.0000	-	-

A closer inspection of the per-class metrics reveals how quantization affects different object categories (Tables 3.4-3.12). The non-quantized model (Table 3.4) already shows a large performance spread between categories, with high AP values

Table 3.6: Per Class metrics for ternary(*int*2) with alpha scaling uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
truck	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
bus	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
trailer	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
construction_vehicle	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
pedestrian	0.2480	0.2350	0.4700	1.4600	0.7660	0.4160
motorcycle	0.0000	0.8020	0.8750	1.2810	1.0250	0.4470
bicycle	0.0000	0.4510	0.8010	1.5760	0.7940	0.1330
traffic_cone	0.0770	0.2080	0.5770	-	-	-
barrier	0.0150	0.3700	0.9090	1.2110	-	-

Table 3.7: Per Class metrics for *int*4 uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.8470	0.1810	0.7490	1.6040	0.3160	0.1930
truck	0.5650	0.3400	0.7930	1.6060	0.2570	0.2240
bus	0.7110	0.3330	0.8600	1.5390	0.4930	0.2550
trailer	0.3600	0.5550	0.8560	1.6370	0.2180	0.1450
construction_vehicle	0.2420	0.7480	0.6830	1.5840	0.1090	0.3080
pedestrian	0.8300	0.1400	0.3260	1.5910	0.2460	0.0800
motorcycle	0.6490	0.2040	0.7900	1.5430	0.3810	0.2380
bicycle	0.4970	0.1700	0.8050	1.6290	0.2520	0.0090
traffic_cone	0.6610	0.1360	0.3360	-	-	-
barrier	0.6770	0.1960	0.8900	1.2870	-	-

for *car* (0.852), *pedestrian* (0.841), and *motorcycle* (0.661), while challenging classes such as *construction_vehicle* (0.237) and *trailer* (0.400) lag behind.

Introducing ternary quantization (*int*2) leads to a drastic degradation (Tables 3.5 and 3.6). Most vehicle-related classes collapse to AP = 0, and their geometric metrics saturate at the worst possible values (ATE, ASE, AOE, AVE = 1.0), confirming that the network fails entirely to detect or localize them. Only the *pedestrian* class retains some functionality, with AP = 0.249 without scaling and a nearly identical 0.248 with scaling; nonetheless, its orientation and velocity metrics severely degrade. Alpha scaling provides partial recovery for a few categories—e.g., *motorcycle*, *bicycle*, and *traffic_cone*— but the model remains non-viable at 2 bits.

At 4 bits (Tables 3.7 and 3.8), performance returns much closer to the baseline. All classes regain their characteristic AP ranking structure, and the deviations

Table 3.8: Per Class metrics for *int*4 with alpha scaling uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.8480	0.1820	0.7490	1.6030	0.3110	0.1920
truck	0.5630	0.3460	0.7930	1.6100	0.2520	0.2250
bus	0.7130	0.3320	0.8600	1.5380	0.4700	0.2620
trailer	0.3620	0.5510	0.8570	1.6400	0.2110	0.1510
construction_vehicle	0.2430	0.7820	0.6910	1.5910	0.1140	0.3060
pedestrian	0.8310	0.1400	0.3310	1.5920	0.2460	0.0800
motorcycle	0.6520	0.2020	0.7880	1.5340	0.3840	0.2360
bicycle	0.5030	0.1720	0.8040	1.6450	0.2430	0.0080
traffic_cone	0.6600	0.1350	0.3330	-	-	-
barrier	0.6670	0.1960	0.8890	1.2960	-	-

Table 3.9: Per Class metrics for *int*8 uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.8520	0.1780	0.7500	1.5880	0.2710	0.1970
truck	0.5770	0.3460	0.7940	1.6090	0.2470	0.2280
bus	0.7100	0.3440	0.8610	1.5410	0.4440	0.2670
trailer	0.4000	0.5270	0.8630	1.6080	0.2170	0.1480
construction_vehicle	0.2350	0.7530	0.6860	1.5890	0.1160	0.3120
pedestrian	0.8420	0.1350	0.3290	1.5790	0.2310	0.0870
motorcycle	0.6650	0.1980	0.7900	1.5260	0.3730	0.2450
bicycle	0.5130	0.1640	0.8070	1.6460	0.2190	0.0090
traffic_cone	0.6740	0.1290	0.3310	-	-	-
barrier	0.6730	0.1920	0.8880	1.3080	-	-

from the non-quantized model become small (typically within 1-2% AP for most classes). Harder categories such as *trailer* and *construction_vehicle* remain slightly more affected, but geometric errors stay nearly unchanged. Alpha scaling has only marginal effects at this bit width.

The 8-bit quantization results (Tables 3.9 and 3.10) are essentially indistinguishable from the baseline for all classes. AP, ATE, ASE, and AOE values differ only in the third decimal place, confirming that the model behaves as if unquantized. Even the most sensitive categories, such as *bicycle* and *construction_vehicle*, preserve nearly the same performance.

Finally, the 16-bit quantized models (Tables 3.11 and 3.12) match the non-quantized results exactly, as expected, since the quantization error becomes negligible at this precision. No class shows any statistically meaningful deviation.

Table 3.10: Per Class metrics for *int*8 with alpha scaling uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.8490	0.1850	0.7500	1.5860	0.2790	0.1990
truck	0.5770	0.3470	0.7950	1.6090	0.2520	0.2280
bus	0.7090	0.3390	0.8610	1.5390	0.4540	0.2610
trailer	0.3980	0.5270	0.8620	1.6140	0.2150	0.1490
construction_vehicle	0.2400	0.7470	0.6920	1.6070	0.1110	0.3240
pedestrian	0.8290	0.1430	0.3240	1.5800	0.2310	0.0870
motorcycle	0.6400	0.2020	0.7890	1.5240	0.4020	0.2350
bicycle	0.4850	0.1740	0.8080	1.6290	0.2340	0.0080
traffic_cone	0.6540	0.1400	0.3360	-	-	-
barrier	0.6670	0.1960	0.8890	1.3090	-	-

Table 3.11: Per Class metrics for *int*16 uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.8520	0.1780	0.7500	1.5870	0.2700	0.1960
truck	0.5760	0.3460	0.7940	1.6070	0.2500	0.2280
bus	0.7050	0.3460	0.8610	1.5410	0.4540	0.2660
trailer	0.4010	0.5240	0.8630	1.6060	0.2160	0.1470
construction_vehicle	0.2370	0.7530	0.6860	1.5910	0.1170	0.3130
pedestrian	0.8400	0.1350	0.3280	1.5790	0.2310	0.0870
motorcycle	0.6620	0.1990	0.7910	1.5260	0.3740	0.2410
bicycle	0.5090	0.1630	0.8080	1.6400	0.2210	0.0090
traffic_cone	0.6710	0.1330	0.3310	-	-	-
barrier	0.6710	0.1930	0.8880	1.3070	-	-

Overall, the per-class analysis demonstrates three key points:

1. ternary uniform quantization is too coarse to preserve object-level information,
2. 4-bit weights and 8-/16-bit activations retain nearly all of the baseline performance regardless of scaling
3. 8-bit and 16-bit uniform quantization are effectively lossless for this model across all categories.

Table 3.12: Per Class metrics for *int*16 with alpha scaling uniform model

Object Class	Class	AP	ATE	ASE	AOE	AVE
car	0.8520	0.1780	0.7500	1.5870	0.2710	0.1970
truck	0.5760	0.3450	0.7940	1.6080	0.2480	0.2270
bus	0.7080	0.3450	0.8610	1.5410	0.4530	0.2640
trailer	0.4010	0.5240	0.8630	1.6080	0.2160	0.1470
construction_vehicle	0.2380	0.7470	0.6870	1.5780	0.1170	0.3130
pedestrian	0.8400	0.1350	0.3290	1.5790	0.2310	0.0870
motorcycle	0.6610	0.1980	0.7910	1.5240	0.3800	0.2410
bicycle	0.5090	0.1640	0.8080	1.6370	0.2200	0.0100
traffic_cone	0.6720	0.1310	0.3320	-	-	-
barrier	0.6720	0.1930	0.8880	1.3060	-	-

Chapter 4

Mixed-Precision Search Algorithm and Results

4.1 Introduction to MPQ Search Algorithms

4.1.1 Sensitivity-Based and Heuristic-Based Search Techniques

The development and application of Multi-objective Evolutionary Algorithms (MOEAs) to many-objective problems (MaOPs)—defined as optimization problems involving four or more conflicting objectives—has led to the proliferation of specialized search techniques often categorized as preference-based (implicitly handling **sensitivity** to objective differences) or methods transforming the original problem (which encompass several **heuristic** search methodologies).

Sensitivity-Based Techniques

As the number of objectives increases in MaOPs, standard Pareto dominance loses effectiveness because most solutions become non-dominated. Sensitivity-based techniques restore selection pressure by refining preference relations to account for the magnitude or structure of objective differences [39].

Crisp Preference Relations These methods use additional metrics to compare solutions:

1. $(1-k)$**-dominance**: Counts objectives where $\mathbf{x}$ is better, equal, or worse than $\mathbf{x}'$, introducing k-optimality. For $k = 0$, it reduces to standard Pareto dominance [39].

2. **Favour relation and SCO**: $\mathbf{x}$ is favoured over $\mathbf{x}'$ if it is better in more objectives. SCO establishes a partial order despite non-transitivity.

3. **ϵ-preferred**: Compares solutions by counting objectives exceeding a threshold ϵ; ties are broken by the favour relation.

4. **Preference ordering (PO_k, $PO_{k,z}$)**: Measures efficiency across all k-objective subspaces, allowing discrimination among mutually non-dominated solutions.

5. **$-\epsilon$-DOM ranking**: Uses *mepsd*, the minimum ϵ to make $\mathbf{x}'$ weakly dominate $\mathbf{x}$. The rank is the minimum *mepsd* over all comparisons.

6. **Expansion dominance**: Adjusts dominance areas via a parameter S, providing finer solution ranking.

Fuzzy Preference Relations These extend crisp approaches using membership functions:

1. **$(1 - k_F)$-dominance**: Fuzzy version of $(1 - k)$-dominance with trapezoidal membership functions μ to quantify negligible differences.

2. **Fuzzy-dominance-driven GA (FDD-GA)**: Calculates degrees of dominance (μ_a and μ_p); fuzzy rank of $\mathbf{x}$ is the maximum degree of being dominated.

Heuristic-Based Search Techniques

Heuristic-based methods address MaOP challenges such as computational cost and visualization complexity by transforming the problem into a related one that is easier to solve using existing MOEA frameworks [39].

Scalarization-Based Methods convert a MaOP into single- or fewer-objective problems via aggregation or decomposition.

Objective Aggregation:

1. Desirability index: Maps objectives/constraints to desirability functions (0-1) and aggregates them per category, reducing the number of objectives.

2. Correlation-based aggregation: Groups objectives to maximize intra-group correlation, combining each group into a single fitness value ($m' < m$).

Decomposition Approaches:

1. MOEA/D: Assigns scalarization functions and weight vectors to individuals, performing selection and crossover within neighborhoods; adaptive strategies may switch scalarization methods based on Pareto front convexity.

2. MSOPS / MSOPS-II: Evaluates solutions against target vectors via weighted min-max methods. MSOPS-II dynamically generates targets and reduces complexity.

Dimensional Reduction remove redundant or weakly conflicting objectives:

1. PCA-based reduction: Identifies principal components to retain objectives with largest contributions.

2. Greedy reduction: Iteratively selects an objective subset minimizing dominance-structure error δ.

3. Unsupervised feature selection: Uses correlation among non-dominated solutions to discard least-conflicting objectives.

Indicator-Based MOEAs transform MaOPs into single-objective problems by optimizing quality indicators:

1. IBEA: Compares solutions using dominance-preserving binary indicators (e.g., $I_{\epsilon+}$, I_{HD}).

2. SMS-EMOA: Maximizes hypervolume using non-dominated sorting; high dimensions require approximation heuristics.

Space Partitioning,ϵR-EMO: Alternates between full-objective iterations and non-overlapping reduced subspaces, solving a series of related subproblems.

Summary of Sensitivity- and Heuristic-Based Methods

Sensitivity-based techniques and heuristic-based transformations provide complementary strategies for addressing the scalability issues of MOEAs in many-objective optimization. Sensitivity-enhanced preference relations restore selection pressure when Pareto dominance becomes ineffective, while heuristic approaches simplify the problem through scalarization, objective reduction, indicator optimization, or space partitioning. Each method introduces its own search bias, influencing convergence and diversity. As a result, adaptive scalarization schemes, dynamic objective-reduction mechanisms, and hybrid combinations remain key directions for improving MaOP performance.

NSGA-II [40] is designed for problems with fewer objectives, it remains a standard reference due to:

1. Well-understood behavior: Its selection based on Pareto sorting and crowding distance provides a stable comparison point for assessing improvements introduced by new preference relations.

2. Clear exposure of MaOP challenges: NSGA-II quickly loses discriminatory power as dimensionality increases, making it useful for illustrating where sensitivity-based and heuristic methods offer advantages.

3. Availability and extensibility: Its simple, modular structure and widespread implementation facilitate integration of modified ranking criteria or objective-reduction strategies.

Thus, NSGA-II is chosen for this work not for superior many-objective performance, but for its reliability and interpretability against which enhanced techniques can be systematically evaluated.

4.1.2 The Need for a Cost Function Balancing Accuracy Preservation and Bit-Cost Minimization

Designing mixed-precision quantization algorithms for deep neural networks is essentially a balancing act. On one hand, we want to shrink the model or reduce its computational cost; on the other, we need to keep its accuracy high enough for the task. In real-world applications like real-time detection, robotics, or autonomous driving, quantization often has to be aggressive to meet strict speed or hardware limits. But if we push compression too far, accuracy suffers; if we focus too much on accuracy, we lose the benefits of quantization. This trade-off becomes even harder as models grow larger and more complex.

To navigate this trade-off, we need a cost function that properly balances compression and accuracy. Without it, the optimization process may drift toward overly compact models that no longer work well, or toward overly precise models that are still too expensive to deploy. A well-constructed cost helps the algorithm explore useful parts of the search space instead of being dominated by just one objective.

In practice, different phases of quantization may require shifting the focus between maintaining general model stability and emphasizing task-specific performance. As the search progresses, the cost function should highlight the components that matter most for the final task, while still allowing compression in less critical areas.

Overall, a balanced cost mechanism works much like preference handling in many-objective optimization. It helps the optimizer distinguish between candidate solutions and avoids getting stuck when simple accuracy or compression metrics aren't enough. With this guidance, the quantization process can find models that meet strict efficiency requirements without compromising their functional reliability.

4.2 The Proposed Genetic Algorithm (GA) for MPQ Optimization

4.2.1 Rationale for Employing NSGA-II

The Nondominated Sorting Genetic Algorithm II (NSGA-II [40]) has become one of the most influential and widely adopted Multi-objective Evolutionary Algorithms (MOEAs) due to its balance of computational efficiency, convergence reliability, and diversity preservation. In contrast to earlier nondominated sorting-based approaches, NSGA-II introduces key methodological advances that make it particularly suitable for complex multi-objective optimization problems where maintaining a well-distributed set of high-quality Pareto-optimal solutions is essential.

Motivational Considerations

Classical MOEAs such as the original NSGA exhibited several limitations—high computational cost, the absence of elitism, and reliance on problem-dependent diversity parameters—that restricted their scalability and robustness. The rapid growth of solution sets in multi-objective search requires not only efficient ranking mechanisms but also effective selection pressure to guide the population toward the Pareto-optimal front. NSGA-II addresses these needs through a principled integration of algorithmic enhancements that reduce complexity, strengthen convergence guarantees, and enable parameterless density estimation.

Computational Efficiency and Fast Sorting A central motivation for adopting NSGA-II lies in its fast nondominated sorting procedure, which reduces the computational burden from $O(MN^3)$ to $O(MN^2)$ by maintaining for each candidate both a domination count and a list of dominated solutions. This improvement is particularly significant for studies involving moderately large populations or multiple objectives, where computational overhead can otherwise become prohibitive.

Elitism and Convergence Assurance NSGA-II incorporates an explicit elitist strategy by forming an intermediate population that merges parents and offspring before selection. This guarantees that high-quality solutions, once discovered, are retained across generations. The elitism mechanism plays a pivotal role in accelerating convergence toward the true Pareto-optimal front and preventing the loss of valuable nondominated solutions—an issue common in earlier MOEA designs.

Parameterless Diversity Preservation Instead of relying on a user-defined sharing parameter, NSGA-II employs a crowding-distance-based density estimator and a crowded-comparison operator. This approach promotes uniform spread along the Pareto front without additional tuning, thereby reducing algorithmic sensitivity

to arbitrary parameter choices. By ranking solutions first by nondomination level and subsequently by crowding distance, NSGA-II maintains selection pressure while encouraging exploration of less populated regions of the objective space.

4.2.2 Optimization Setup: Search Space, Encoding, and Operators

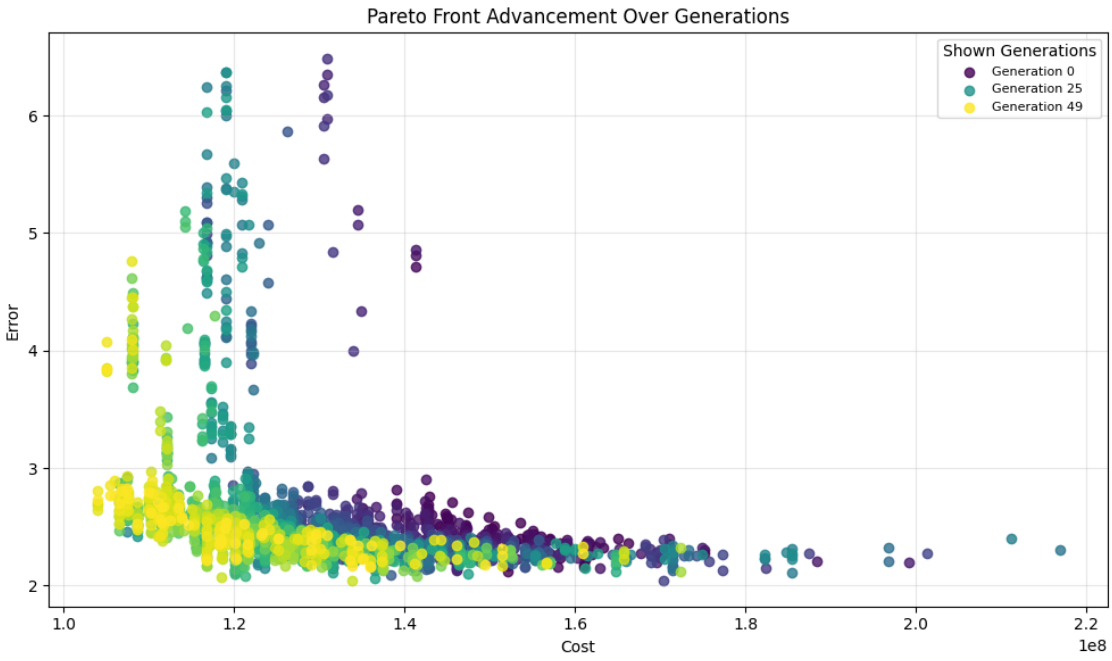


Figure 4.1: Evolution of population distributions across 50 NSGA-II generations. Each point corresponds to one individual evaluated in terms of total bit-cost and bounding-box error. The progression illustrates convergence toward the Pareto front.

In order to apply NSGA-II to the quantization-aware design problem, it is essential to clearly define the search space, solution representation, and the evolutionary operators governing population dynamics. The following subsections describe the configuration adopted in this study, emphasizing how individuals are encoded and how variation operators are tailored to the discrete nature of quantization bit assignments.

Search Space and Individual Encoding

The search space is defined by the set of allowable quantization bitwidths for each quantized layer in the network. In this work, each layer may take one of the discrete values $\{2, 4, 8, 16\}$ bits. Accordingly, an individual in the population is represented as a fixed-length chromosome whose genes correspond to the bitwidth assignments

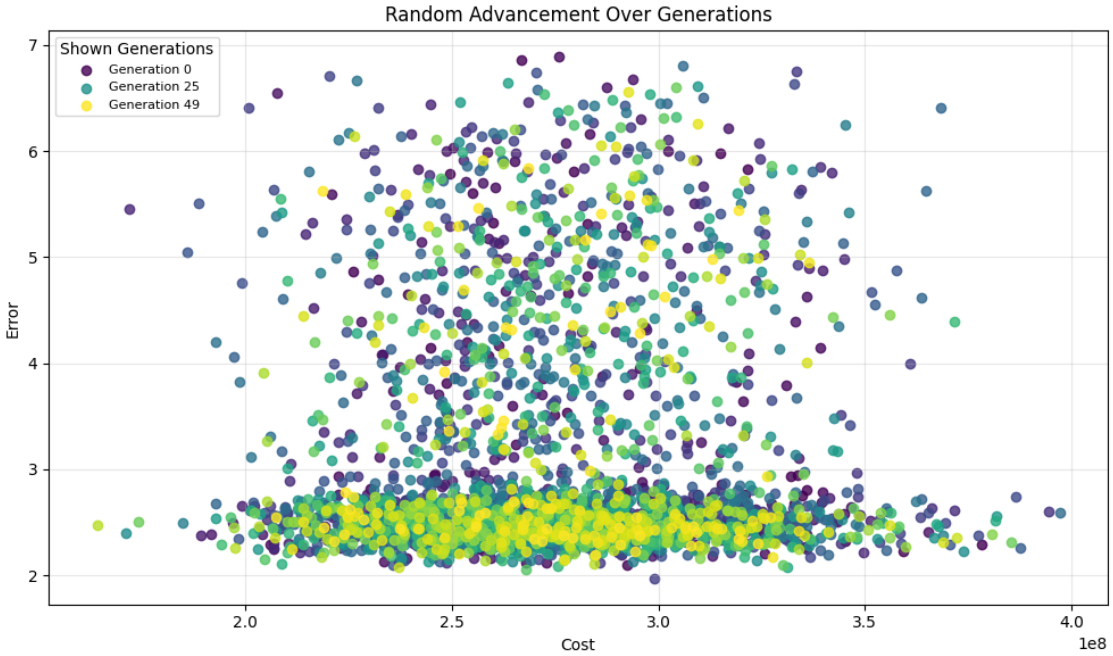


Figure 4.2: Evolution of randomly generated populations across 50 generations, evaluated using the same bit-cost and accuracy-error metrics as NSGA-II.

of the quantized layers. This encoding provides a direct mapping between evolutionary search and architectural design choices: each candidate solution specifies a complete quantization configuration, enabling NSGA-II to explore trade-offs between accuracy, latency, and model size.

In total, each individual chromosome contains 108 genes, corresponding to the 108 layers subject to quantization. Since each gene may independently take one of the four discrete bitwidth values 2,4,8,16, the overall size of the search space becomes

$$|\Omega| = 4^1 08 = 2^2 16 \approx 1.0 \times 1065$$

a combinatorial domain of astronomical scale. This emphasizes the necessity of an evolutionary approach: exhaustive enumeration or grid-based search is entirely infeasible.

Mutation Operator

Given the discrete and ordered structure of the quantization levels, mutation plays a central role in ensuring both local refinement and global exploration. Two mutation strategies were considered during the experimentation process:

In the first configuration, mutation acted by randomly selecting a subset of genes and replacing their values with any other bitwidth from the admissible set. This

provides strong exploratory power but may lead to abrupt changes that disrupt promising solution structures.

To improve stability and better exploit neighborhood information, a second mutation strategy was adopted in later iterations. This refined operator restricts gene alterations to adjacent bitwidth levels, effectively allowing only multiplicative increases or decreases by a factor of two. For example, a gene encoded as 4 bits may mutate to 2 or 8 bits, but not directly to 16 bits. This constraint preserves the discrete hierarchy of quantization choices while enabling smoother transitions across the search landscape.

Crossover Operator

Crossover is applied to promote the exchange of structural information between parent solutions. In this setup, a simple yet effective recombination mechanism is employed: each offspring inherits approximately half of its genes from each parent. By splitting the chromosome at the midpoint (or another predetermined division), the operator constructs new individuals that combine quantization patterns from both contributors. This promotes diversity while maintaining interpretability and consistency with the fixed-length encoding.

4.2.3 The Fitness Functions

To effectively guide NSGA-II through the mixed-precision quantization search space, two complementary fitness functions are defined. These objectives reflect the dual goals of reducing model complexity while preserving the predictive fidelity essential for downstream tasks. Together, they form the multi-objective optimization framework within which NSGA-II evaluates and ranks candidate quantization configurations.

Fitness Objective 1: Bit-Cost Minimization

The first fitness function quantifies the total bit-cost associated with a given mixed-precision assignment. For each layer with weight tensor $\mathbf{W}$ quantized to b bits, the cost is computed as

$$F_{\text{bits}} = \sum_{l} b_l \cdot |\mathbf{W}_l|,$$

where $|\mathbf{W}_l|$ denotes the number of parameters in layer l and $b_l \in \{2,4,8,16\}$ specifies its assigned precision. Minimizing F_{bits} encourages the algorithm to identify quantization patterns that significantly reduce memory footprint and computational overhead. This objective provides NSGA-II with a direct incentive to favor low-precision choices, particularly in layers with large parameter counts.

Fitness Objective 2: Accuracy Preservation

The second fitness function measures the discrepancy between the full-precision model and its quantized counterpart. This objective evolves across iterations of the optimization process to more accurately reflect the factors that influence performance:

Cosine-Based Functional Error (Initial Stage). During early exploration, accuracy preservation is evaluated using the cosine similarity between original and quantized module outputs:

$$F_{\cos} = 1 - \cos(f_\theta(\mathbf{x}),\ f_{\theta_b}(\mathbf{x}))\,.$$

This formulation captures high-level representational distortions in activation space and provides a stable, task-agnostic signal that helps preserve functional consistency across quantized layers. By penalizing directional changes in activation vectors, this objective identifies layers where precision must be maintained to prevent early-stage degradation.

Task-Specific Bounding-Box Error (Refinement Stage). As the optimization progresses, the fitness function transitions to a task-specific measure aligned with the performance requirements of detection architectures. Rather than comparing intermediate activations, the discrepancy is defined directly over the semantic components of the predicted bounding boxes:

$$F_{\text{box}} = \alpha\, E_{\text{center}} + (1 - \alpha)\, E_{\text{yaw+size}}.$$

Empirical evaluation indicates that center-point localization and geometric attributes contribute approximately equally to detection performance degradation; consequently, the weighting parameter is set to $\alpha \approx 0.5$. This refinement ensures that the search prioritizes quantization configurations that maintain downstream detection quality, focusing precision where it yields the greatest impact on model utility.

The computational cost of evaluating a single candidate further reinforces this limitation. Computing the task-specific bounding-box error for one mixed-precision configuration requires approximately 7 seconds, meaning that even a vanishingly small fraction of the full search space would take longer than the age of the universe to evaluate. NSGA-II thus provides a practical mechanism for navigating this immense design space efficiently by focusing evaluations on promising regions of the Pareto landscape rather than attempting full enumeration.

Multi-Objective Integration in NSGA-II

NSGA-II leverages both fitness objectives simultaneously, treating them as independent axes in the Pareto optimization landscape. The bit-cost objective promotes compactness, while the accuracy objective—either cosine-based or task-specific, depending on the iteration—ensures consistency with full-precision behavior. Through nondominated sorting and crowding-based selection, the algorithm balances these competing pressures, producing a diverse set of solutions that span the trade-off frontier.

Together, F_{bits} and the accuracy-preservation objective constitute a robust multi-objective formulation tailored to mixed-precision quantization. By beginning with a general functional similarity measure and later shifting to domain-specific semantic error, the fitness functions guide NSGA-II from broad exploratory search toward fine-grained architectural refinement. This staged approach ensures that the resulting quantization schemes achieve both high compression efficiency and strong task-level performance.

To visualize the behavior of the evolutionary process under these two objectives, Fig. 4.1 illustrates the distribution of all individuals across 50 generations of NSGA-II[40]. Each generation contains 100 individuals, plotted according to their total bit-cost (x-axis) and the bounding-box-derived accuracy error (y-axis). Early generations exhibit wide dispersion and suboptimal trade-offs, whereas successive iterations reveal a clear migration toward the Pareto front. This pattern reflects the combined effect of nondominated sorting, elitism, and crowding-distance-based diversity preservation in steadily guiding the population toward high-quality compromises between compactness and task-level fidelity.

In contrast to the structured progression observed in Fig. 4.1, the random-generation baseline in Fig. 4.2 demonstrates no meaningful movement toward the Pareto front. Although purely random sampling occasionally produces individuals with low accuracy error, these candidates almost always achieve such performance by relying on excessively large bitwidth assignments, resulting in substantially higher total bit-costs. Because random search lacks any form of selection pressure, nondominated sorting, or elitism, the sampled individuals remain scattered in dominated regions of the objective space without developing coherent trade-offs. This comparison underscores that the seemingly competitive low-error solutions found by chance are not viable, and that guided evolutionary pressure is essential for discovering efficient mixed-precision quantization configurations.

4.3 Optimal MPQ Results and Network Analysis

This section reports the outcomes of the multi-objective mixed-precision quantization (MPQ) framework derived through NSGA-II. We first characterize the Pareto-optimal trade-offs achieved during the search process, then present the detailed

bit-allocation of the final mixed-precision configuration, and finally compare full-precision, uniform quantization, and the full suite of MPQ variants. The analysis highlights the distinct roles played by whole-model search ("C" variants), module-wise optimization (non-"C" variants), error-oriented and weight-oriented objectives ("E" and "W"), and the influence of quantization schemes (alpha-aware "A" versus min–max "Q"). The introduction of a distance-error objective ("DE") further refines the search by aligning optimization pressure with downstream detection performance.

4.3.1 Presentation of the Pareto Front: Optimal Trade-offs

The Pareto front constructed by NSGA-II encapsulates a diverse spectrum of quantization configurations that optimally balance accuracy preservation and bit-cost reduction. Each solution in the front represents a nondominated trade-off: no candidate can be improved in accuracy without incurring greater memory or computational cost.

A clear structure emerges in the distribution of solutions across the front. Uniform 4-bit quantization (4AQ) provides a strong lower-bound on bit-cost while preserving the majority of the full-precision model's performance. Error-oriented MPQ variants (E-type) populate central regions of the front, characterized by moderate bit-cost and strong preservation of NDS, mAP, and geometric detection metrics. Conversely, weight-oriented configurations (W-type) explore the extreme compression region, achieving significant bit-cost reductions at the expense of accuracy—suitable for highly resource-constrained deployment.

Full-model NSGA-II searches (C variants) produce more coherent precision distributions and improved Pareto smoothness compared to module-wise searches. Finally, the incorporation of the distance-error objective in MPQ-DE_CAE yields the most favorable global trade-off, demonstrating that task-aligned fitness formulations meaningfully influence convergence.

4.3.2 The Final Mixed-Precision Model: Detailed Layer-by-Layer Bit Assignment

From the set of Pareto-optimal solutions, the MPQ-DE_CAE configuration emerges as the most balanced architecture when jointly considering accuracy and efficiency. This configuration benefits from: (i) full-model optimization (C), ensuring interactions across modules are captured; (ii) alpha-aware quantization (A), which enhances dynamic range representation; (iii) error-oriented objectives (E), prioritizing accuracy retention; and (iv) distance-error measurement (DE), which directly reflects downstream detection sensitivity.

The resulting mixed-precision pattern exhibits a structured allocation of bitwidths

Table 4.1: Performance comparison of full-precision (NQ), uniform quantization (4AQ), and all MPQ variants. "C" indicates full-model NSGA-II search, while non-"C" variants use module-wise search. "E" and "W" denote error-oriented and weight-oriented objectives, respectively. "A" indicates alpha-aware quantization and "Q" min–max quantization. "DE" introduces a distance-error objective.

	NDS	mAP	mATE	mASE	mAOE	mAVE	mAAE	total	relative	rapport
NQ	0.5605	0.6130	0.297	0.710	1.553	0.267	0.186	1.17e+09	1.0000	1.00
4AQ	0.5547	0.6043	0.304	0.710	1.561	0.279	0.182	1.46e+08	0.1250	8.00
MPQ-AE	0.5166	0.5654	0.319	0.716	1.514	0.418	0.208	1.70e+08	0.1456	6.87
MPQ-AW	0.2050	0.1442	0.539	0.784	1.260	0.828	0.521	1.04e+08	0.0886	11.28
MPQ-CAE	0.5162	0.5679	0.316	0.713	1.553	0.439	0.209	1.38e+08	0.1177	8.50
MPQ-CAW	0.4217	0.4080	0.344	0.710	1.556	0.573	0.197	1.32e+08	0.1129	8.85
MPQ-CQE	0.5084	0.5441	0.312	0.714	1.533	0.414	0.198	1.42e+08	0.1218	8.21
MPQ-CQW	0.3847	0.3474	0.331	0.708	1.547	0.648	0.203	1.32e+08	0.1132	8.84
MPQ-DE_CAE	0.5422	0.5874	0.305	0.712	1.556	0.310	0.189	1.21e+08	0.1033	9.68
MPQ-QE	0.5053	0.5583	0.322	0.715	1.549	0.492	0.209	2.39e+08	0.2044	4.89
MPQ-QW	0.1521	0.0971	0.553	0.800	1.298	1.232	0.610	1.07e+08	0.0916	10.92
MPQ-RCAE	0.4949	0.5239	0.345	0.742	1.549	0.389	0.194	1.49e+08	0.1271	7.87
MPQ-RCAW	0.4413	0.4420	0.352	0.715	1.548	0.519	0.212	1.31e+08	0.1124	8.90
MPQ-RCQE	0.4878	0.5222	0.328	0.729	1.528	0.473	0.203	1.76e+08	0.1509	6.63
MPQ-RCQW	0.4515	0.4708	0.344	0.728	1.541	0.563	0.204	1.29e+08	0.1105	9.05

across the network. Early feature-extraction layers and heads responsible for localization, orientation, and size prediction consistently retain higher precision (8–16 bits), reflecting their sensitivity to quantization noise. Mid-level fusion and transformation layers operate effectively at 4–8 bits, while later or redundant pathways are aggressively quantized to 2–4 bits. This heterogeneous precision assignment is difficult to achieve through manual design or uniform quantization and highlights the nuanced layer-wise importance captured through NSGA-II.

For reference, the genome of the MPQ-DE_CAE configuration is:

$$
\begin{aligned}
&(4, 4, 16, 4, 8, 4, 8, 2, 8, 4, 8, 4, 2, 8, 4, 4, 4, 4, 2, 2, 4, 2, 4,\\
&4, 4, 2, 4, 4, 4, 4, 8, 4, 4, 4, 8, 2, 2, 4, 16, 8, 2, 2, 2, 8, 2, 16,\\
&2, 4, 2, 8, 2, 4, 8, 8, 2, 4, 4, 8, 8, 4, 16, 4, 8, 2, 2, 4, 8, 2, 2,\\
&2, 4, 4, 4, 8, 2, 4, 2, 4, 4, 2, 4, 2, 8, 2, 4, 2, 4, 2, 8, 4, 8, 4, 4,\\
&16, 4, 4, 4, 16, 8, 4, 4, 4, 8, 4, 4, 4, 4, 4)
\end{aligned}
$$

A full mapping of these bit values to their corresponding network layers is provided in Appendix 5.3.2. This appendix enables reproducibility and facilitates layer-level analysis of quantization sensitivity, providing insights into which components benefit most from higher precision and which can be safely compressed.

4.3.3 Final Comparison: Full-Precision vs. Uniform Quantization vs. Optimal MPQ

Table 4.1 summarizes the performance of all evaluated quantization schemes. Several conclusions can be drawn:

Accuracy–Efficiency Behavior. Full-precision (NQ) achieves the highest accuracy but incurs the largest bit-cost. Uniform 4-bit quantization (4AQ) achieves an 8× compression with minimal degradation. The best MPQ configuration, MPQ-DE_CAE, surpasses 4AQ in accuracy while delivering an even greater compression ratio (approximately 9.7×).

Influence of Optimization Strategy. Error-oriented MPQ variants (E) consistently outperform weight-oriented ones (W) on accuracy metrics. Full-model search (C) dominates module-wise search, confirming the importance of capturing global interactions. Min–max-only quantization (Q) performs slightly below alpha-aware quantization (A), though it remains competitive in several metrics.

Effect of Fitness Formulation. Replacing cosine similarity with distance-error (DE) markedly improves downstream task performance. MPQ-DE_CAE achieves the highest accuracy among all MPQ variants while maintaining one of the lowest bit-costs, illustrating the benefit of semantically aligned fitness functions.

Best Overall Trade-off. Among all evaluated configurations, MPQ-DE_CAE provides the most compelling balance:

- NDS of 0.5422 (only ≈3% below full precision),
- mAP of 0.5874,
- total bit-cost of 1.21×10^8,
- relative cost of 0.1033,
- and a rapport score of 9.6786, the highest accuracy-to-cost ratio.

This demonstrates that mixed-precision quantization guided by NSGA-II can exceed uniform quantization in both accuracy and compression, providing a principled route to highly efficient yet performance-preserving model deployments.

4.4 Embedded System Feasibility Analysis

4.4.1 Projected Memory Reduction

The primary benefit of mixed-precision quantization lies in its ability to tailor the bitwidth of each layer to its actual sensitivity, enabling substantial compression without uniformly sacrificing precision. It is important to note that the reported memory savings apply *only to the layers that undergo quantization*; non-quantized components (e.g., certain normalization, positional encoding, or auxiliary heads) contribute additional overhead and therefore limit the total achievable reduction when considering the full model.

Given the total bit-costs reported in Table 4.1, the memory footprint of the quantized portions can be approximated as

$$\text{Memory}_{\text{quantized}} = \frac{F_{\text{bits}}}{8} \quad \text{bytes},$$

where F_{bits} represents the accumulated bit-cost across all quantized layers.

A comparison across quantization strategies highlights several observations:

- **Uniform 4-bit quantization (4AQ)** achieves an approximate 8× reduction for the quantized layers relative to their full-precision form, serving as a strong baseline.
- **Mixed-precision weight-focused configurations (W variants)** achieve the smallest bit-costs, corresponding to up to 11× compression for the quantized subset of the model, though at the cost of notable performance degradation.
- **Error-oriented mixed-precision configurations (E variants)** maintain competitive accuracy while achieving 6–9× compression of the quantized layers, demonstrating effective precision allocation.
- **The MPQ-DE_CAE configuration** offers the best global trade-off, reducing the quantized-layer memory footprint to roughly 10% of its full-precision equivalent while preserving near-baseline detection performance.

Despite these substantial reductions, the *complete* model—including both quantized and non-quantized components—remains relatively large by current standards for embedded and edge AI hardware. Modern deployments increasingly target sub-10 MB total footprints for real-time applications, placing additional pressure on architectural pruning, operator fusion, or hybrid compression strategies beyond quantization alone.

These observations emphasize that while MPQ provides meaningful compression for hardware-limited settings, further model-level optimization is required to meet the strict memory and compute budgets of contemporary embedded platforms.

Chapter 5

Conclusions and Future Work

5.1 Summary of Findings

In this work, we proposed a general post-training quantization (PTQ) methodology for transformer-based detection models. The method is designed to be architecture-agnostic and can be applied to a wide range of large-scale transformer networks, not limited to BEVFusion[16]. By selectively quantizing layers based on their sensitivity, we achieve meaningful memory reductions relative to the quantized layers, while preserving accuracy. While the primary goal was to develop a flexible and generalizable quantization approach, we also analyzed the feasibility of deploying these models on resource-constrained embedded platforms. Even with PTQ, the overall model remains relatively large for current edge devices, indicating that additional compression techniques or architectural adaptations may be needed for deployment [22, 19].

5.2 Contributions

The key contributions of this thesis are:

- A general PTQ framework that can be applied across various transformer architectures to reduce memory footprint while maintaining performance.
- Integration of a multi-objective search strategy (NSGA-II) to identify optimal quantization configurations that balance accuracy and resource efficiency [40].
- Demonstration that the method is broadly applicable and not tied to any specific architecture like BEVFusion.

- Insights into the practical constraints of deploying large transformer models on embedded platforms, serving as guidance for future optimization efforts.

5.3 Future Work

5.3.1 Extension to Other Large Transformer Architectures

Future research can extend this PTQ methodology to other transformer-based models, including Vision Transformers (ViT), Swin Transformers, and hybrid architectures [2, 34]. Each architecture exhibits different layer sensitivities, suggesting that tailored quantization profiles could further optimize the trade-off between memory efficiency and accuracy. Combining PTQ with quantization-aware training or model pruning could improve the feasibility of deploying these networks on edge devices without compromising their general applicability.

5.3.2 Exploration of Alternative Optimization Strategies

The current approach employs a multi-objective genetic algorithm (NSGA-II) to identify optimal quantization configurations. Future work could explore new evaluation metrics within the GA framework to better capture model efficiency, robustness, and deployment constraints. Additionally, alternative search paradigms for layer-wise α selection, such as reinforcement learning, Bayesian optimization, or differentiable search methods, could offer faster convergence and more flexible exploration of the quantization search space. These extensions would further enhance the generality and practicality of the proposed PTQ methodology.

List of Figures

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[3] J. Mao, S. Shi, X. Wang, and H. Li, "3d object detection for autonomous driving: A comprehensive survey," *International Journal of Computer Vision*, vol. 131, no. 8, pp. 1909–1963, 2023.

[4] R. Qian, X. Lai, and X. Li, "3d object detection for autonomous driving: A survey," *Pattern Recognition*, vol. 130, p. 108796, 2022.

[5] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang, "3d object detection with pointformer," 2021.

[6] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020.

[7] Y. Wang, V. C. Guizilini, T. Zhang, Y. Wang, H. Zhao, and J. Solomon, "Detr3d: 3d object detection from multi-view images via 3d-to-2d queries," in *Conference on robot learning*, pp. 180–191, PMLR, 2022.

[8] J. Mao, Y. Xue, M. Niu, H. Bai, J. Feng, X. Liang, H. Xu, and C. Xu, "Voxel transformer for 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3164–3173, October 2021.

[9] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, 2018.

[10] H. Sheng, S. Cai, N. Zhao, B. Deng, Q. Liang, M.-J. Zhao, and J. Ye, "Ct3d++: Improving 3d object detection with keypoint-induced channel-wise transformer," 2024.

[11] P. Sun, M. Tan, W. Wang, C. Liu, F. Xia, Z. Leng, and D. Anguelov, "Swformer: Sparse window transformer for 3d object detection in point clouds," in *European Conference on computer vision*, pp. 426–442, Springer, 2022.

[12] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point

sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

[13] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Q. Yu, and J. Dai, "Bevformer: learning bird's-eye-view representation from lidar-camera via spatiotemporal transformers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[14] K.-C. Huang, T.-H. Wu, H.-T. Su, and W. H. Hsu, "Monodtr: Monocular 3d object detection with depth-aware transformer," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4012–4021, 2022.

[15] C. Zhou, L. Yu, A. Babu, K. Tirumala, M. Yasunaga, L. Shamis, J. Kahn, X. Ma, L. Zettlemoyer, and O. Levy, "Transfusion: Predict the next token and diffuse images with one multi-modal model," 2024.

[16] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. Rus, and S. Han, "Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation," 2024.

[17] X. Chen, T. Zhang, Y. Wang, Y. Wang, and H. Zhao, "Futr3d: A unified sensor fusion framework for 3d detection," in *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 172–181, 2023.

[18] Y. Li, Y. Chen, X. Qi, Z. Li, J. Sun, and J. Jia, "Unifying voxel-based representation with transformer for 3d object detection," *Advances in Neural Information Processing Systems*, vol. 35, pp. 18442–18455, 2022.

[19] F. Wang, M. Zhang, X. Wang, X. Ma, and J. Liu, "Deep learning for edge computing applications: A state-of-the-art survey," *IEEE Access*, vol. 8, pp. 58322–58336, 2020.

[20] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[21] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[22] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021.

[23] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," 2020.

[24] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *Computer Vision – ECCV 2008* (D. Forsyth, P. Torr, and A. Zisserman, eds.), (Berlin, Heidelberg), pp. 44–57, Springer Berlin Heidelberg, 2008.

[25] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer*

Vision and Pattern Recognition (CVPR), June 2016.

[26] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder, "The mapillary vistas dataset for semantic understanding of street scenes," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[27] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, T. Darrell, *et al.*, "Bdd100k: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, vol. 2, no. 5, p. 6, 2018.

[28] P. Wang, X. Huang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The apolloscape open dataset for autonomous driving and its application," *IEEE transactions on pattern analysis and machine intelligence*, vol. 1, 2019.

[29] Z. Che, G. Li, T. Li, B. Jiang, X. Shi, X. Zhang, Y. Lu, G. Wu, Y. Liu, and J. Ye, "D^2-city: A large-scale dashcam video dataset of diverse traffic scenarios," 2019.

[30] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, 2012.

[31] A. Patil, S. Malla, H. Gang, and Y.-T. Chen, "The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9552–9557, 2019.

[32] A. Patil, S. Malla, H. Gang, and Y.-T. Chen, "The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9552–9557, 2019.

[33] J. Mei, A. Z. Zhu, X. Yan, H. Yan, S. Qiao, L.-C. Chen, and H. Kretzschmar, "Waymo open dataset: Panoramic video panoptic segmentation," in *Computer Vision – ECCV 2022* (S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, eds.), (Cham), pp. 53–72, Springer Nature Switzerland, 2022.

[34] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021.

[35] M. Contributors, "MMDetection3D: OpenMMLab next-generation platform for general 3D object detection." `https://github.com/open-mmlab/mmdetection3d`, 2020.

[36] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[37] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-power computer vision*, pp. 291–326, Chapman and Hall/CRC, 2022.

[38] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization," in *European conference on computer vision*, pp. 191–207, Springer, 2022.

[39] C. Von Lücken, B. Barán, and C. Brizuela, "A survey on multi-objective evolutionary algorithms for many-objective problems," *Computational optimization and applications*, vol. 58, no. 3, pp. 707–756, 2014.

[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

Appendix: Quantized Layer Genome

This appendix contains the complete set of quantized layers used in the model. Each table corresponds to a top-level module (e.g. `img_backbone`, `img_neck`, `view_transform`, etc.). When the tables are read sequentially, they form the complete `genome of the individual` in the evolutionary setting, the position is indicated by the ID in the tables.

Table 1: img_backbone – Stage 0 Quantized Layers

ID	Layer	Type
1	patch_embed.projection	conv2d
2	stages.0.blocks.0.attn.w_msa.qkv	linear
3	stages.0.blocks.0.attn.w_msa.proj	linear
4	stages.0.blocks.0.attn.w_msa.q_k_matmul	matmul
5	stages.0.blocks.0.attn.w_msa.attn_v_matmul	matmul
6	stages.0.blocks.0.ffn.layers.0.0	linear
7	stages.0.blocks.0.ffn.layers.1	linear
8	stages.0.blocks.1.attn.w_msa.qkv	linear
9	stages.0.blocks.1.attn.w_msa.proj	linear
10	stages.0.blocks.1.attn.w_msa.q_k_matmul	matmul
11	stages.0.blocks.1.attn.w_msa.attn_v_matmul	matmul
12	stages.0.blocks.1.ffn.layers.0.0	linear
13	stages.0.blocks.1.ffn.layers.1	linear
14	stages.0.downsample.reduction	linear

Table 2: img_backbone – Stage 1 Quantized Layers

ID	Layer	Type
15	stages.1.blocks.0.attn.w_msa.qkv	linear
16	stages.1.blocks.0.attn.w_msa.proj	linear
17	stages.1.blocks.0.attn.w_msa.q_k_matmul	matmul
18	stages.1.blocks.0.attn.w_msa.attn_v_matmul	matmul
19	stages.1.blocks.0.ffn.layers.0.0	linear
20	stages.1.blocks.0.ffn.layers.1	linear
21	stages.1.blocks.1.attn.w_msa.qkv	linear
22	stages.1.blocks.1.attn.w_msa.proj	linear
23	stages.1.blocks.1.attn.w_msa.q_k_matmul	matmul
24	stages.1.blocks.1.attn.w_msa.attn_v_matmul	matmul
25	stages.1.blocks.1.ffn.layers.0.0	linear
26	stages.1.blocks.1.ffn.layers.1	linear
27	stages.1.downsample.reduction	linear

Table 3: img_backbone – Stage 2 Quantized Layers

ID	Layer	Type
28	stages.2.blocks.0.attn.w_msa.qkv	linear
29	stages.2.blocks.0.attn.w_msa.proj	linear
30	stages.2.blocks.0.attn.w_msa.q_k_matmul	matmul
31	stages.2.blocks.0.attn.w_msa.attn_v_matmul	matmul
32	stages.2.blocks.0.ffn.layers.0.0	linear
33	stages.2.blocks.0.ffn.layers.1	linear
34	stages.2.blocks.1.attn.w_msa.qkv	linear
35	stages.2.blocks.1.attn.w_msa.proj	linear
36	stages.2.blocks.1.attn.w_msa.q_k_matmul	matmul
37	stages.2.blocks.1.attn.w_msa.attn_v_matmul	matmul
38	stages.2.blocks.1.ffn.layers.0.0	linear
39	stages.2.blocks.1.ffn.layers.1	linear
40	stages.2.blocks.2.attn.w_msa.qkv	linear
41	stages.2.blocks.2.attn.w_msa.proj	linear
42	stages.2.blocks.2.attn.w_msa.q_k_matmul	matmul
43	stages.2.blocks.2.attn.w_msa.attn_v_matmul	matmul
44	stages.2.blocks.2.ffn.layers.0.0	linear
45	stages.2.blocks.2.ffn.layers.1	linear
46	stages.2.blocks.3.attn.w_msa.qkv	linear
47	stages.2.blocks.3.attn.w_msa.proj	linear
48	stages.2.blocks.3.attn.w_msa.q_k_matmul	matmul
49	stages.2.blocks.3.attn.w_msa.attn_v_matmul	matmul
50	stages.2.blocks.3.ffn.layers.0.0	linear
51	stages.2.blocks.3.ffn.layers.1	linear
52	stages.2.blocks.4.attn.w_msa.qkv	linear
53	stages.2.blocks.4.attn.w_msa.proj	linear
54	stages.2.blocks.4.attn.w_msa.q_k_matmul	matmul
55	stages.2.blocks.4.attn.w_msa.attn_v_matmul	matmul
56	stages.2.blocks.4.ffn.layers.0.0	linear
57	stages.2.blocks.4.ffn.layers.1	linear
58	stages.2.blocks.5.attn.w_msa.qkv	linear
59	stages.2.blocks.5.attn.w_msa.proj	linear
60	stages.2.blocks.5.attn.w_msa.q_k_matmul	matmul
61	stages.2.blocks.5.attn.w_msa.attn_v_matmul	matmul
62	stages.2.blocks.5.ffn.layers.0.0	linear
63	stages.2.blocks.5.ffn.layers.1	linear
64	stages.2.downsample.reduction	linear

Table 4: img_backbone – Stage 3 Quantized Layers

ID	Layer	Type
65	stages.3.blocks.0.attn.w_msa.qkv	linear
66	stages.3.blocks.0.attn.w_msa.proj	linear
67	stages.3.blocks.0.attn.w_msa.q_k_matmul	matmul
68	stages.3.blocks.0.attn.w_msa.attn_v_matmul	matmul
69	stages.3.blocks.0.ffn.layers.0.0	linear
70	stages.3.blocks.0.ffn.layers.1	linear
71	stages.3.blocks.1.attn.w_msa.qkv	linear
72	stages.3.blocks.1.attn.w_msa.proj	linear
73	stages.3.blocks.1.attn.w_msa.q_k_matmul	matmul
74	stages.3.blocks.1.attn.w_msa.attn_v_matmul	matmul
75	stages.3.blocks.1.ffn.layers.0.0	linear
76	stages.3.blocks.1.ffn.layers.1	linear

Table 5: img_neck – Lateral and FPN Convolution Layers

ID	Layer	Type
77	lateral_convs.0.conv	conv2d
78	lateral_convs.1.conv	conv2d
79	fpn_convs.0.conv	conv2d
80	fpn_convs.1.conv	conv2d

Table 6: view_transform – Quantized Convolution Layers

ID	Layer	Type
81	dtransform.0	conv2d
82	dtransform.3	conv2d
83	dtransform.6	conv2d
84	depthnet.0	conv2d
85	depthnet.3	conv2d
86	depthnet.6	conv2d
87	downsample.0	conv2d
88	downsample.3	conv2d
89	downsample.6	conv2d

Table 7: fusion_layer – Quantized Convolution Layer

ID	Layer	Type
90	0	conv2d

Table 8: pts_backbone – Block 0 Quantized Convolution Layers

ID	Layer	Type
91	blocks.0.0	conv2d
92	blocks.0.3	conv2d
93	blocks.0.6	conv2d
94	blocks.0.9	conv2d
95	blocks.0.12	conv2d
96	blocks.0.15	conv2d

Table 9: pts_backbone – Block 1 Quantized Convolution Layers

ID	Layer	Type
97	blocks.1.0	conv2d
98	blocks.1.3	conv2d
99	blocks.1.6	conv2d
100	blocks.1.9	conv2d
101	blocks.1.12	conv2d
102	blocks.1.15	conv2d

Table 10: pts_neck – Deblock Convolution Layers

ID	Layer	Type
103	deblocks.0.0	conv2d

Table 11: bbox_head – Layers

ID	Layer	Type
104	shared_conv	conv2d
105	heatmap_head.0.conv	conv2d
106	heatmap_head.1	conv2d
107	decoder.0.ffn.layers.0.0	linear
108	decoder.0.ffn.layers.1	linear