Rohit Kumar

Rohit_Thesis_-5-52

Conference

Document Details

Submission ID

trn:oid:::27535:150326002

Submission Date

Sep 2, 2026, 12:14 PM GMT+5:30

Download Date

Sep 2, 2026, 12:20 PM GMT+5:30

File Name

Rohit_Thesis_-5-52.pdf

File Size

1.9 MB

48 Pages

13,595 Words

75,935 Characters

5% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Small Matches (less than 8 words)

Match Groups

62 Not Cited or Quoted 5%
Matches with neither in-text citation nor quotation marks

4 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

4% Internet sources
2% Publications
3% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **62** Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- **4** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4% Internet sources
2% Publications
3% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	University of Essex on 2026-08-25	<1%
2	Publication	Leticia de Carvalho Pereira. "AIVFusion: a deep learning fusion architecture for m...	<1%
3	Student papers	University of Essex on 2026-08-19	<1%
4	Internet	icaeta.aiplustech.org	<1%
5	Student papers	Bournemouth University on 2026-06-02	<1%
6	Student papers	Heriot-Watt University on 2026-07-23	<1%
7	Student papers	UCL on 2025-08-28	<1%
8	Student papers	Leiden University on 2026-08-12	<1%
9	Publication	Tariq M. Khan, Qazi Emad Ul Haq, Shahzaib Iqbal, Toufique Ahmed Soomro. "Edge...	<1%
10	Student papers	University of Ulster on 2026-05-04	<1%

11 Internet

arxiv.org <1%

12 Internet

oulurepo.oulu.fi <1%

13 Internet

d1q2ihs98wnwia.cloudfront.net <1%

14 Internet

mlsysbook.ai <1%

15 Internet

www.authorstream.com <1%

16 Internet

pure.royalholloway.ac.uk <1%

17 Student papers

University of Birmingham on 2024-09-16 <1%

18 Student papers

University of Warwick on 2026-08-28 <1%

19 Publication

Valentim, Sérgio Alexandre Pascoal. "Firearm Model Identification Based on Fired... <1%

20 Publication

Khudran M. Alzhrani. "Top-Confidence Gapped Cross-Entropy for Compact Huma... <1%

21 Student papers

RMIT University on 2025-06-15 <1%

22 Student papers

Tilburg University on 2026-03-12 <1%

23 Student papers

University of Lancaster on 2013-09-13 <1%

24 Internet

aistackauthority.com <1%

25 Publication

Arpita Nath Boruah, Mrinal Goswami, Manoj Kumar, Octavio Loyola-González. "E... <1%

26 Student papers

Bournemouth University on 2026-05-26 <1%

27 Publication

Lavanya Sharma, Pradeep Kumar Garg. "Artificial Intelligence - Technologies, App... <1%

28 Publication

Nazmul Siddique, Mohammad Shamsul Arefin, Sheak Rashed Haider Noori, M Sha... <1%

29 Student papers

University of Essex on 2026-04-24 <1%

30 Student papers

University of Stirling on 2026-04-21 <1%

31 Publication

Karzhaubayev, Anuar. "Computation Offloading and Resource Allocation in Mobil... <1%

32 Publication

Muhannad Abd al Rub Banyan, Yahya Omar Al-Amari, Hisham Mohammed Banya... <1%

33 Publication

Sang, Mufan. "Robust, Generalized, and Efficient Representation Learning and M... <1%

34 Publication

Sudeep Tanwar, Maniklal Das, Abderrahim Benslimane, Rajesh Gupta. "Blockchai... <1%

35 Student papers

University of Sheffield on 2024-09-11 <1%

36 Student papers

University of Surrey on 2015-01-19 <1%

37 Internet

hal.inria.fr <1%

38 Internet

ir.quantinuum.com <1%

39 Internet
jf2.hep.com.cn <1%

40 Internet
ontariotechu.scholaris.ca <1%

41 Internet
projekter.aau.dk <1%

42 Internet
re4dy.eu <1%

43 Internet
researchonline.gcu.ac.uk <1%

44 Student papers
unistgallen-plagiat on 2024-01-22 <1%

45 Internet
unsworks.unsw.edu.au <1%

46 Internet
www.mdpi.com <1%

47 Internet
www.twdb.state.tx.us <1%

48 Publication
Hoak, Blaine Delaney. "Uncovering the Mechanisms of Adversarial Vulnerability i... <1%

49 Student papers
University of Essex on 2026-08-21 <1%

50 Student papers
University of Warwick on 2026-08-31 <1%

51 Publication
Zhang, Tianyi. "Compression Algorithms for Efficient Inference and Adaptation of... <1%

52 Internet
link.springer.com <1%

53 Internet

repository.najah.edu <1%

54 Internet

theses.hal.science <1%

55 Internet

www.jait.us <1%

56 Internet

www.scpe.org <1%

Efficient Edge AI Framework for Real-Time IoT Vision System Using Deep Learning

Rohit Patel

ABSTRACT

Modern vision systems are widely used in autonomous vehicles, smart surveillance, healthcare imaging, industrial monitoring, and IoT-based applications. These systems continuously process large amounts of visual data and require fast and reliable decision-making. Traditional cloud-based processing often faces problems such as high latency, bandwidth usage, privacy risks, and unstable network connectivity. Edge Artificial Intelligence (Edge AI) has emerged as an effective solution by performing inference closer to the data source. This work studies efficient deployment of deep learning based vision models on resource-constrained edge devices. The work focuses on understanding how different optimisation techniques affect latency, throughput, memory usage, and model size during deployment. A comparative framework is proposed for four major AI-IoT architectures: cloud-only, edge-only, edge-cloud hybrid, and federated learning. The study also reviews three major model compression approaches: pruning, quantisation, and knowledge distillation. An experimental analysis is carried out using the YOLOv8n object detection model in three configurations: PyTorch FP32, ONNX FP32, and ONNX INT8. The experiments are performed on a CPU-only environment using a subset of the Intel IoT DevKit traffic dataset. Results show that INT8 quantisation This has a significant impact on the size of the model, but has no impact on the quality of the inference. The CPU hardware used for testing. The results are evidence that The advantages of model compression are very much dependent on the The underlying hardware and runtime environment. The study finds that for effective edge-AI deployment, four key components must be in place: careful balance between accuracy, latency, hardware capability, and resource efficiency.

Keywords: *Edge artificial intelligence; Internet of Things; computer vision; YOLOv8; object detection; model compression; quantisation; ONNX Runtime; federated learning; deep learning deployment.*

LIST OF PUBLICATIONS

International Conference

1. Patel, R. and Sharma, A., 2026. Efficient Edge AI Framework for Real-Time IoT Vision System Using Deep Learning. In *Proceedings of the 7th International Conference on Data Analytics & Management (ICDAM-2026)*. **(Scopus-Indexed, Accepted for Presentation, Scheduled during 12–14 June 2026).**

TABLE OF CONTENTS

Acknowledgement **i**

Candidate's Declaration **ii**

Certificate **iii**

Abstract **iv**

List of Publications **v**

Table of Contents **vi**

List of Table(s) **ix**

List of Figure(s) **x**

List of Symbols, Abbreviations and Nomenclature **xi**

1 Introduction **1**

1.1 Background and Motivation 1

1.2 The Shift from Cloud to Edge Intelligence 2

1.3 Problem Statement 3

1.4 Research Questions and Objectives 3

1.5 Scope and Limitations 4

1.6 Summary of Contributions 4

1.7 Organisation of the Thesis 5

2 Background and Literature Review **6**

2.1 Computer Vision: From Handcrafted Features to Deep Learning . . . 6

2.2 Deep Learning Architectures for Vision 6

2.2.1 Convolutional Neural Networks and Vision Transformers . . . 6

2.2.2 Lightweight Backbones for Edge Deployment 7

2.3 IoT Foundations and Vision Workload Characteristics 8

2.4 Edge Computing, Hardware Accelerators and Runtimes 8

2.5 Review of Existing Surveys and Research Gaps 9

2.6 Summary 11

3 AI-IoT Vision Architectures and Comparison Framework **12**

3.1 Evolution of AI-IoT Vision Architectures 12

3.2 Cloud-Based Vision Architecture 12
3.3 Edge-Based Vision Architecture 13
3.4 Edge-Cloud Hybrid Architecture 13
3.5 Federated and Decentralised Architecture 14
3.6 Proposed Six-Axis Comparison Framework 14
3.6.1 The Six Evaluation Axes 14
3.6.2 Cross-Paradigm Comparison and Selection Guidance 14
3.7 Summary 15

4 Optimisation Strategies for Edge Deployment 16
4.1 The Accuracy-Efficiency Trade-off 16
4.2 Pruning 16
4.3 Quantisation 17
4.3.1 FP32 to FP16 17
4.3.2 INT8 Post-Training Quantisation 17
4.3.3 Quantisation-Aware Training 18
4.4 Knowledge Distillation 18
4.5 Stacked Compression Pipelines and Comparative Impact 18
4.6 Summary 19

5 Proposed Methodology and Implementation 20
5.1 Research Design Overview 20
5.2 Selection of Model and Optimisation Pipeline 20
5.3 Evaluation Metrics 21
5.4 Experimental Setup: Hardware, Software, Dataset 21
5.5 Implementation Pipeline 22
5.5.1 Baseline PyTorch Implementation and ONNX Export 22
5.5.2 INT8 Quantisation and Benchmarking Harness 22
5.6 Reproducibility Measures 24
5.7 Summary 24

6 Experimental Results, Comparative Analysis, and Applications 25
6.1 Baseline PyTorch FP32 Results 25
6.2 ONNX FP32 Results 25
6.3 ONNX INT8 Results 26
6.4 Comparative Analysis 27
6.5 Discussion 29

6.5.1 Why INT8 Reduced Size But Not CPU Latency 29
6.5.2 Comparison with the Jetson Orin NX Literature 30
6.6 Summary 31

7 Conclusion, Future Work and Social Impact 32

45

7.1 Conclusion 32
7.2 Future Work 32
7.3 Social Impact 33

References 35

List of Publications and their Proofs 40

Plagiarism Report 45

Curriculum Vitae 53

LIST OF TABLE(S)

2.1 Recent edge AI vision surveys and the gaps addressed 10

3.1 Qualitative six-axis comparison of major AI-IoT vision architectures from surveyed literature 15

4.1 Comparative impact of three major compression families, synthesised from the 2024-2026 literature 19

6.1 Measured metrics for PyTorch FP32, ONNX FP32, and ONNX INT8 on the 150-frame traffic-video subset 27

LIST OF FIGURE(S)

1.1 Cloud-centric vs. edge-centric AI-IoT vision pipelines 2

2.1 Evolution of AI-IoT vision architectures from cloud-centric to federated and distributed systems (2010-2026) 11

6.1 Six-panel comparison dashboard of performance, resource usage, and detection fidelity across the three configurations 28

6.2 Accuracy-efficiency trade-off analysis of model size and throughput across the three configurations 29

6.3 Per-frame inference latency distribution across the three configurations over 150 test frames 29

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

Symbols	Description
τ	Confidence threshold (object detection, default 0.25)
$\bar{\ell}$	Mean per-frame inference latency (ms)
σ_ℓ	Standard deviation of per-frame latency (ms)
$\overline{\text{FPS}}$	Average frames per second
$\mathscr{L}$	Loss function

Abbreviations	Full Form
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CoAP	Constrained Application Protocol
COCO	Common Objects in Context (dataset)
CPU	Central Processing Unit
FP16	16-bit Floating-Point Precision
FP32	32-bit Floating-Point Precision
FPS	Frames Per Second
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HIPAA	Health Insurance Portability and Accountability Act
INT8	8-bit Integer Precision
IoT	Internet of Things
KD	Knowledge Distillation
mAP	Mean Average Precision
ML	Machine Learning
MLOps	Machine Learning Operations
MQTT	Message Queuing Telemetry Transport
NMS	Non-Maximum Suppression
non-IID	Non-Independent and Non-Identically Distributed
ONNX	Open Neural Network Exchange
PTQ	Post-Training Quantisation
QAT	Quantisation-Aware Training
RQ	Research Question
TOPS	Tera Operations Per Second
TPU	Tensor Processing Unit

ViT	Vision Transformer
YOLO	You Only Look Once (object detector family)

Nomenclature	**Meaning**
$\mathscr{M}_{\mathrm{PT}}$	PyTorch FP32 baseline YOLOv8n model
$\mathscr{M}_{\mathrm{ONNX}}$	ONNX FP32 exported model
$\mathscr{M}_{\mathrm{INT8}}$	ONNX INT8 dynamic-quantised model
$\mathscr{D}_{\mathrm{traffic}}$	Intel IoT DevKit traffic video (647 frames)
N_f	Benchmark subset length (150 frames)
ℓ_i	Latency of frame i (ms)
c_i	CPU utilisation sampled during frame i (%)
m_i	Resident memory during frame i (MB)
D	Total detection count over the benchmark subset

21

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Vision sits at the centre of a large fraction of present-day artificial intelligence deployments. Autonomous vehicles read the road through stacked camera arrays [1], hospital radiology departments process MRI volumes through learned classifiers [2], factory production lines rely on cameras to flag defective parts before they propagate downstream [3], drones inspect crops over hectares of farmland [4], and traffic intersections in urban deployments now run object detection directly at the camera [5]. The pattern across these settings is the same: large volumes of pixel data are produced continuously, and decisions on that data have to be made in the order of tens of milliseconds.

12

For most of the previous decade the standard response to this demand was to push the data to a remote data centre, run the model there, and return the result [6, 7]. The arrangement worked when deployments were small. It did not survive scaling. Four weaknesses emerged consistently across production deployments [7, 8]: tail latency that depends on whatever the public internet is doing at the moment, bandwidth costs that climb in proportion to the number of cameras streaming continuously, privacy exposure because frames containing faces or licence plates have to traverse the public internet before any model touches them, and operational fragility whenever the uplink connection is unreliable, which is the default condition in agricultural and remote-industrial settings.

Edge artificial intelligence has emerged as the response to these weaknesses. Inference is moved back to the device that captured the image, or to a gateway located one network hop away, so that only the eventual decision needs to leave the device [9–11]. The properties that matter for autonomous driving, industrial inspection, and surveillance are now well understood to be local latency, local privacy, and predictable behaviour under poor connectivity, rather than peak floating-point throughput in a distant data centre [11, 12].

turnitin Page 18 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

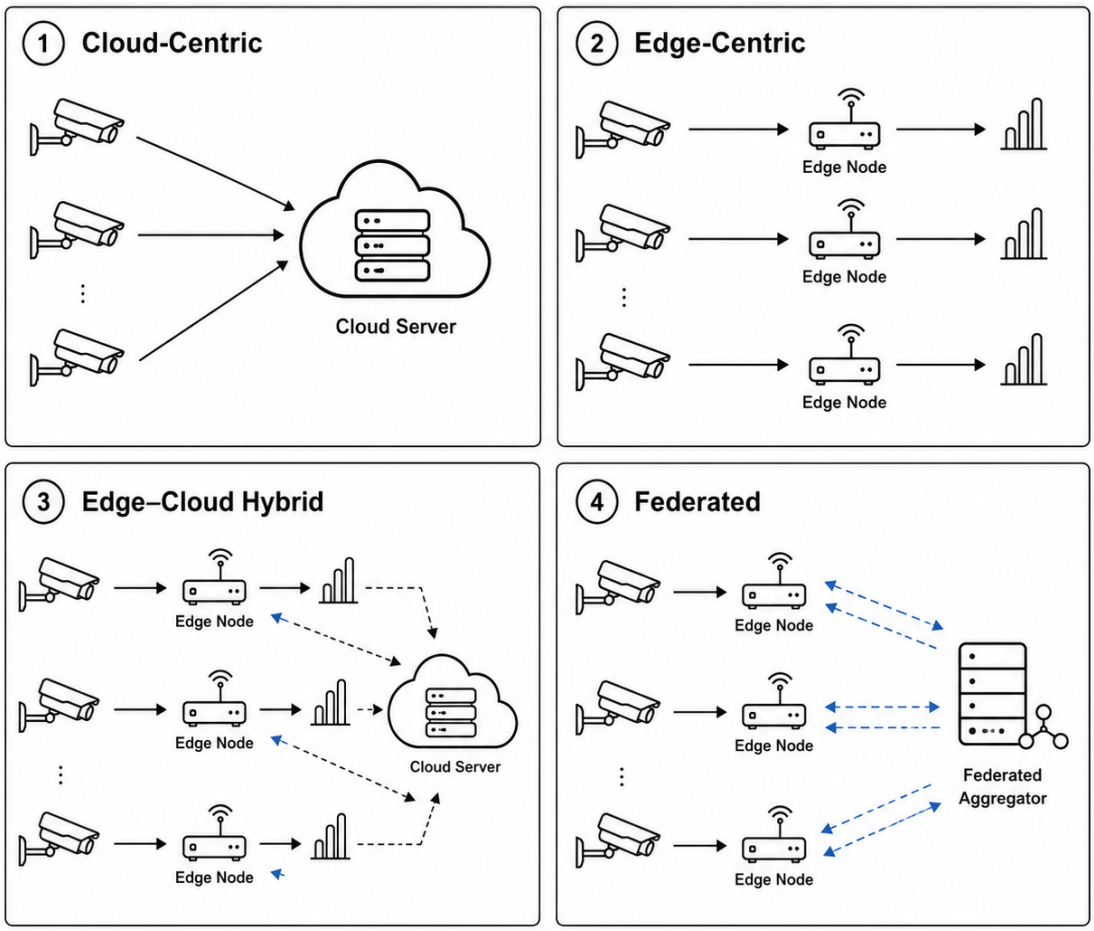

Figure 1.1: Cloud-centric vs. edge-centric AI-IoT vision pipelines

1.2 The Shift from Cloud to Edge Intelligence

The transition has been pushed by three forces acting in parallel. The first is hardware. Embedded accelerators such as the NVIDIA Jetson family, the Google Edge TPU, and the Intel Movidius series now provide tens of trillions of operations per second within power budgets of fifteen to twenty-five watts, which is enough to host detection models that would have been considered data-centre workloads five years ago [13, 14]. The second one is software. Lightweight backbone architectures like MobileNet, MobileNetV2, EfficientNet, and the YOLO family are now standard, hardware-aware runtimes are available. To convert models, including TensorRT, TensorFlow Lite, and ONNX Runtime. translation of trained models into deployable artefacts with minimal manual work. intervention [15–19]. Thirdly, there is regulation. The General Data Protection Regulation The EU GDPR and the US HIPAA both give access rights to the data to the user.Both the European Union GDPR and the U.S. HIPAA grant access to the data to the user. inhibit centralisation of personally identifiable visual information

turnitin Page 18 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

turnitin Page 19 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

The architecture of data, federated learning has emerged. Securely store for healthcare and surveillance use when raw data cannot legally exit from the device [20–22].

1.3 Problem Statement

This study tackles the missing link between abstract claims of an edge and their concrete realization. inference, and the question that faces any engineer, which one is more important? Addressed to the deployment of a vision model: which optimisation exactly results in a measurable decrease in latency, model size or memory The cost to detection of the footprint on the target hardware. fidelity? While there are many reported speedups in the literature, there are many reasons why they may not be achievable in practice. The results of one paper do not always neatly translate to a On different runtime, hardware target or model family. Vision benchmarks are in their early stages for the edge, and the practitioner is This is usually left to the discretion of the person who quotes it, in deciding if a statement, explanation or description is truly accurate. The deployment to be considered is an acceleration.

YOLOv8n is selected as the detector, since it is the smallest variant of a widely deployed family. The optimisation pipeline is the most basic one accessible to practitioners without dedicated hardware: PyTorch FP32, exported to ONNX FP32 for runtime conversion, and then to ONNX INT8 through dynamic quantisation. The hardware is a CPU-only environment, which deliberately emulates the constraints of a low-cost edge device without a GPU accelerator. Six metrics are measured: latency, throughput, model size, CPU utilisation, memory consumption, and detection count relative to the baseline.

1.4 Research Questions and Objectives

Four research questions follow from this problem framing.

RQ1 — How do the four principal architectural paradigms (cloud-only, edge-only, hybrid edge-cloud, and federated learning) compare against the six criteria of latency, bandwidth, privacy, scalability, energy efficiency, and orchestration cost?

RQ2 — What are the documented accuracy-efficiency trade-offs of the three principal compression families: pruning, quantisation, and knowledge distillation?

RQ3 — Does the PyTorch-to-ONNX runtime conversion deliver measurable latency reduction on a CPU-only edge environment, and does INT8 dynamic quantisation

turnitin Page 19 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

deliver additional latency reduction on top of it?

RQ4 — When the quantised model fails to accelerate on a given hardware target, what does that result imply for the practitioner planning an edge deployment?

The corresponding objectives are: to construct an explicit six-axis comparison framework for the four paradigms; to synthesise the trade-off literature for the three compression families into a single comparative table; to implement and benchmark the full PyTorch-ONNX-INT8 pipeline on a CPU-only environment; and to relate the measured numbers to the published Jetson Orin NX literature so that the practitioner can place their own deployment on the same trade-off curve.

1.5 Scope and Limitations

The experimental evaluation is restricted to a single detector family (YOLOv8n), a single dataset (a 647-frame traffic video released by Intel for IoT testing), and a CPU-only execution environment. Pruning is reviewed in the literature but not applied experimentally, since structured pruning of YOLO backbones requires a fine-tuning pass that is outside the scope of a CPU-only setup. Knowledge distillation is likewise reviewed but not applied. Federated learning is discussed architecturally but not implemented. The experimental contribution should be read as a reproducibility demonstration of the basic runtime and quantisation pathway rather than as a benchmark for absolute performance.

1.6 Summary of Contributions

The work makes five contributions:

Contribution 1: Six-axis comparison framework. A scoreable framework is set out for evaluating the four principal architectural paradigms against latency, bandwidth, privacy, scalability, energy efficiency, and orchestration cost, and applied to give qualitative ratings for each combination.

Contribution 2: Compression trade-off synthesis. Pruning, quantisation, and knowledge distillation are summarised in a single comparative table giving size reduction, speed-up, typical accuracy loss, engineering effort, and hardware friendliness for each, based on the 2024–2026 literature.

Contribution 3: Reproducible CPU edge benchmark. The full PyTorch–to–ONNX–to–INT8 pipeline is implemented in roughly two hundred lines of Python and benchmarked on a 150-frame subset of a 647-frame traffic video. All six metrics are reported,

along with the per-frame detection count for fidelity comparison.

Contribution 4: Empirical validation of a hardware-specific claim. The measurements show that INT8 dynamic quantisation reduces model size by 46.5 per cent but increases mean inference latency from 164.29 ms to 258.63 ms on the CPU target. The result validates the claim, repeated across the surveyed literature, that compression gains are hardware-specific and cannot be transferred between targets without re-measurement.

Contribution 5: Comparison against the Jetson Orin NX literature. The CPU-only numbers are placed against the Jetson Orin NX results reported by Aljami et al. [23], which gives an explicit picture of how much of the speed-up reported in the literature depends on accelerator hardware rather than on the optimisation itself.

1.7 Organisation of the Thesis

The remaining six chapters are arranged as follows. Chapter 2 surveys the deep-learning architectures used in vision, the IoT and edge-computing foundations, and the existing survey literature, ending with a gap analysis. Chapter 3 sets out the four architectural paradigms and applies the six-axis comparison framework. Chapter 4 treats the three compression families and their stacking. Chapter 5 describes the proposed methodology and implementation pipeline. Chapter 6 reports the experimental results, the comparative analysis, and the application-domain discussion. Chapter 7 closes with the open challenges, future directions, and a summary of contributions.

32

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Computer Vision: From Handcrafted Features to Deep Learning

The history of practical computer vision divides cleanly into two eras separated by 2012. For roughly two decades before that point, production vision systems were built on hand-engineered descriptors. Scale-invariant feature transforms, histograms of oriented gradients, and Haar cascades were the standard components, paired with classifiers trained on these descriptors [24]. The pipeline worked in controlled lighting and on cooperative subjects. It broke fragmentary often when conditions changed, and the engineering cost of retuning a deployment for a new domain was high enough that vision systems remained narrow in scope.

This switch was not a one for fashion but a one for economics. Once The dataset ImageNet was available, and was of a reasonably large size, and it was trained by the network AlexNet. showed that GPs-trained convolutional networks could be used to significantly beat down the hand-tuned pipelines [25, 26], the Cost of engineering / robustness of deployment = 1 / ratio. The learned representations replaced handcrafted ones throughout the Seeing as the question was moving to how to design from what it was initially, I thought it would be better to move it to its next phase. features to how to design and train networks.

2

2.2 Deep Learning Architectures for Vision

2.2.1 Convolutional Neural Networks and Vision Transformers

4

Immediately following this, convolutional neural networks (CNNs) became the most important development in the field of deep learning. following 2012. The pooling and sharing pattern of a CNN. The problem with capturing spatial hierarchies without requiring the is a clean one captured by . designer to indicate what characteristics to be extracted at what time spatial scale. To overcome the problem of vanishing gradients, AlexNet incorporated into their network ReLU activations and dropout. a network with 8 layers. VGG demonstrated that depth is sufficient to: lift accuracy provided

turnitin Page 23 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

when the architectural building block was kept simple. The innovation that paved the way for networks deeper than As noted in the introduction, twenty layers was residual learning, which was introduced in ResNet [27]: gradients could be skipped. to pass through any depth at any time during the training phase, 152-layer networks became tractable. DenseNet took feature reuse to the next level The idea here is to 'concatenate' activations 'down the depth' dimension, Lowering the number of parameters for the same accuracy. target [28].

Vision transformers arrived in 2020 with the observation that self-attention, originally developed for sequence modelling in natural language processing, could be applied to images by splitting the image into patches and treating those patches as tokens [29]. The base ViT model contains roughly 86 million parameters and trades the local inductive bias of convolutions for the global receptive field of attention. The cost of that trade is data. Pure transformers require substantially more pretraining data than CNNs to reach comparable accuracy, because they have no built-in assumption that nearby pixels are more related than distant ones. Hybrid architectures combining attention with convolution followed quickly, recovering the inductive bias of CNNs in the early layers and the global context of attention in the later ones. The present leaderboard for image classification is dominated by such hybrids [30].

A practical implication for edge deployment is that the choice of backbone family is not just a question of accuracy and parameter count. CNNs map cleanly to the convolution kernels already optimised in TensorRT, ONNX Runtime, and TensorFlow Lite. Transformers rely on attention operations that, until recently, were not as well supported by these runtimes. The gap has been closing, but for a 2026 deployment on Jetson-class hardware the CNN-family lightweight backbones remain the safer default.

2.2.2 Lightweight Backbones for Edge Deployment

For deployment on devices with modest compute and memory budgets, the relevant networks are not the ones that win ImageNet leaderboards. They are the ones that were designed from the start to run within tens of megabytes of memory and a few watts of power. MobileNetV1 introduced depthwise separable convolutions to factor a standard convolution into two cheaper operations [31]. MobileNetV2 added the inverted residual block, in which the residual connection links narrow representations rather than wide ones [15]. EfficientNet introduced compound scaling, varying width, depth, and resolution together according to a fixed coefficient [16].

turnitin Page 23 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

turnitin Page 24 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

For object detection specifically, the YOLO family has become the default choice for edge deployment. The original YOLO [32] reformulated detection as a single regression problem, which removed the need for the region-proposal stage used by R-CNN and its successors and made detection tractable in real time. Subsequent generations refined the backbone, the loss formulation, and the neck architecture, but the single-stage philosophy has remained. YOLOv8 is the most recent production-grade release and ships in five capacity tiers spanning roughly 3 million parameters (YOLOv8n) to 68 million parameters (YOLOv8x), allowing the practitioner to choose the variant that fits the memory and latency budget of the target device [33]. The nano variant in particular is small enough to ship to microcontroller-class devices and forms the basis of the experimental work in Chapter 5.

2.3 IoT Foundations and Vision Workload Characteristics

The Internet of Things supplies the substrate on top of which the edge AI argument sits [34]. A standard IoT stack has sensing at the bottom, networking in the middle, and applications at the top. The arrangement was originally designed for sensors producing small, infrequent readings, such as temperature, humidity, or vibration. Vision workloads do not fit this profile. A 1080p camera at 30 frames per second produces roughly 1.5 megabytes of compressed video data per second per camera, or about 130 gigabytes per day. A deployment of one thousand cameras at this rate would saturate a 1 Gbps uplink several times over.

Three pressures therefore acted on the vision IoT space and forced the move toward edge computation. The first is bandwidth, since streaming continuous video to the cloud is uneconomical beyond a small number of cameras. The second is latency, since autonomous driving and industrial inspection have inner-loop budgets in the order of tens of milliseconds and the public internet cannot guarantee that timing. The third is privacy, since faces, licence plates, and medical imagery are subject to regulatory regimes that constrain where the data may be processed and stored.

2.4 Edge Computing, Hardware Accelerators and Runtimes

Edge computing emerged as the design pattern that addresses all three pressures simultaneously by moving inference back to the device that captured the image, or to a gateway one network hop away [7, 8]. Fog computing extends the idea by adding a regional aggregation tier between the device and the cloud. What made these patterns workable in practice was the simultaneous arrival of three things during the 2018–2022

turnitin Page 24 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

turnitin Page 25 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

window: lightweight networks of the kinds described above, embedded accelerators, and hardware-aware runtimes.

Three accelerator families dominate the present edge market. The NVIDIA Jetson series, from the Nano at the low end to the Orin NX and AGX Orin at the high end, provides discrete GPU compute within power budgets of five to fifty watts and is supported by the full CUDA software stack [14]. The Google Coral The Edge TPU, engineered specifically for products, is incorporated. for INT8 inference and with outstanding energy efficiency for fixed-point workloads. Intel Movidius is a line of vision With the smart-camera market in mind, processing units targets the same market, pairing with: The image-signal processing is also used to accelerate the inference of the same. die.

The runtimes which power these accelerators have come together: three principal options. NVIDIA TensorRT is the runtime of the current choice for Jetson hardware and offers aggressive graph optimisation, With layer fusion, kernel auto-tuning and INT8 calibration [18]. TensorFlow Lite is the default The runtime for deployments to mobile and microcontroller devices, A special level of support for the Edge TPU [19]. ONNX Runtime is the cross-platform That is the option that can take models exported from any framework supporting this. they are trained based on the ONNX standard and deployed on CPU, GPU or special purpose hardware. OpenNeuronx is an execution provider for hardware [17, 35]. The link between edge devices and the internet. They are normally using MQTT for a cloud uplink, and Local mesh networking using CoAP [36, 37].

2.5 Review of Existing Surveys and Research Gaps

There are a few surveys that have tackled the edge AI vision space so far. last three years. Each has its own focus, and the gaps between them are as follows: they leave open motivate the present work. The taxonomy of edge AI devices extends to two broad categories: Gill et al. [12] introduce and discuss. edge AI from a hardware, software and algorithmic perspective. The treatment is broad and not deep and the comparison of architectural paradigms does not employ an, instead it utilises unstructured prose. explicit scoring framework. Wang et al. [38] pay attention to on-device AI models only and provide a comprehensive Collection of compression techniques. The main emphasis of the work is on the model. This keeps the compression aspect separate from the deployment aspect of design, and allows the comparison of compression. The numerical trade-offs are not shown for consolidated families. The earliest of the aforementioned is Sze et al. [13]. Surveys, as well as its treatment of, remain relevant.

56

turnitin Page 25 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

turnitin Page 26 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

used in this book are just examples.The numbers used in this book for example use in hardware–software co-design are empirical. reports are much older than the current generation of accelerators. margin.

The following two more specialised surveys are also applicable. Filho et The Internet of Medical Things (IoMT) is described in al. [39]. particularly with respect to privacy and regulatory, constraints. Villar et al. [40] give a Compression techniques for embedded vision, focused review, Which is the most similar to the optimisation chapter of the present work. Federated learning is extensively discussed in Kairouz et al. [22] and Salh et In the context of IoT, Salh et al. [41] have discussed the latter one. deployments specifically..

Three gaps emerge from the literature: First, the surveys typically treat architectures, compression, and applications as separate topics, and the practitioner is left to combine them when planning a deployment. Second, the comparisons of architectural paradigms in the existing surveys are qualitative and prose-based, without an explicit scoring framework that a practitioner could apply to their own requirements. Third, very few of the surveys include a worked experimental example that ties a named model, a named dataset, and a named hardware target to specific measured numbers. The Aljami et al. [23] study is one of the few that does this, and the experimental chapter of the present work builds on it.

Table 2.1: Recent edge AI vision surveys and the gaps addressed

Survey	**Primary Focus**	**Gap Left Open**
Gill et al. [12], 2024	Broad edge AI taxonomy	Architectural comparison is prose-only, no explicit scoring
Wang et al. [38], 2025	On-device AI models	Compression families presented separately, no consolidated trade-off table
Sze et al. [13], 2017	Hardware–software co-design	Pre-dates current accelerator generation
Filho et al. [39], 2024	Medical IoT	Domain-specific, limited generalisation
Villar et al. [40], 2024	Compression for embedded vision	No worked deployment example
Kairouz et al. [22], 2021	Federated learning	Vision-specific issues underspecified

turnitin Page 26 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

turnitin Page 27 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

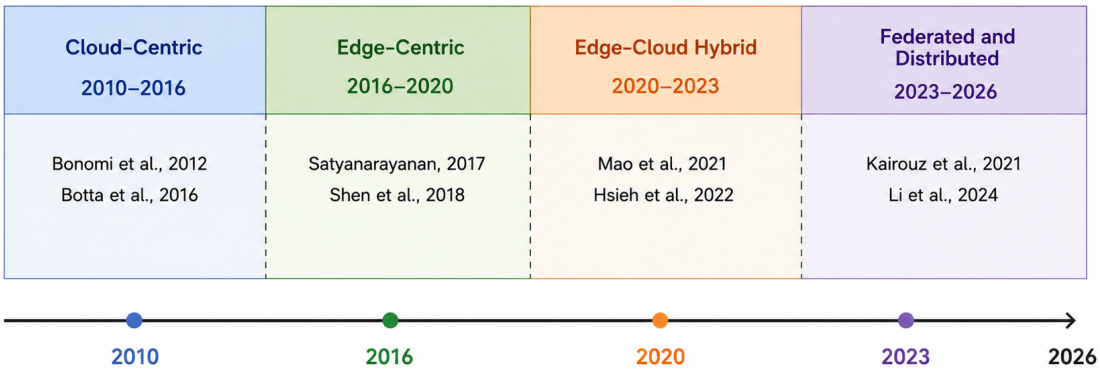

Figure 2.1: Evolution of AI-IoT vision architectures from cloud-centric to federated and distributed systems (2010-2026)

2.6 Summary

Computer vision moved from handcrafted features to learned representations in 2012, and the practical implication for the edge has been the parallel development of lightweight backbones (MobileNet, EfficientNet, YOLO family) alongside the larger high-accuracy networks (ResNet, ViT and its hybrids). The Internet of Things supplies the deployment substrate, but vision workloads do not fit the original sensor-network profile and have forced the development of edge computing, fog computing, and federated architectures. Hardware accelerators (Jetson, Coral, Movidius) and runtimes (TensorRT, TensorFlow Lite, ONNX Runtime) make these architectures workable in practice. Recent surveys cover pieces of this picture but leave three gaps open: the absence of an explicit scoring framework for the architectural paradigms, the absence of a consolidated trade-off table for the compression families, and the absence of a worked experimental example tying a model, a dataset, and a target hardware platform to measured numbers. The remaining chapters address these gaps in turn.

turnitin Page 27 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

CHAPTER 3

AI-IOT VISION ARCHITECTURES AND COMPARISON FRAMEWORK

3.1 Evolution of AI-IoT Vision Architectures

The evolution of AI-IoT vision architectures in recent years. It has been 15 years of a gradual expansion of computation outwards from the centre. To a central machine that is running handcrafted feature, over Ethernet. extraction [24]. Cloud-centric vision dominated In the 2010s, the elastic compute resources enabled training of neural networks were introduced. and inference low cost provided bandwidth and latency The constraints were acceptable in the case of [7, 34]. After lightweight backbones and embedded accelerators enabled on-device inference to become viable at. production scale [11, 12]. Hybrid edge-cloud designs arose if neither end covered all of them. actually, the requirements of a real deployment [42], and The closed loop was completed with federated learning, where devices can With no loss of raw, contribute to a shared model data [22, 41]. The four architectures are NOT steps of a linear sequence in! The latter, which takes the place of the former. Each remains where appropriate, different to a different set of constraints, The practitioner is faced with a key design decision: selection problem.

3.2 Cloud-Based Vision Architecture

In the cloud configuration, training, storage, and All of the inferences are stored on remote servers owned by the operator. It takes snapshots and transmits them to a regional or The model runs in the central data centre, and returns the result. This arrangement leads to the following three properties. Bandwidth Because the resolution of the video needs to exit, However, it doesn't go on idle forever. Tail latency is also unpredictable, because it is dependent on the condition of the The intermediate network at the time of delivery of the frame. Privacy exposure is large, due to the further travel of sensitive frames over The wire is thicker than the wire they should be on, and at times they cross regulation. This is a result of the process of jurisdiction [7, 8].

The architecture remains appropriate when none of those three properties bind the

turnitin Page 29 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

design. Offline analytics, batch processing of archived footage, and training pipelines that consume large volumes of historical data all fit cleanly into a cloud-only deployment. Real-time field deployments do not.

3.3 Edge-Based Vision Architecture

In the edge-based configuration, inference runs on the device itself or on a gateway located one network hop away. The model is exported into a format compatible with the on-device runtime (TensorRT for Jetson, TensorFlow Lite for mobile and microcontroller targets, ONNX Runtime as the cross-platform option) and the device responds locally to each frame. Only the eventual decision, or summarised metadata, leaves the device. Three things had to come together for this configuration to be practical at scale. Lightweight networks of the kinds described in Chapter 2 kept the inference workload within the device's compute budget. Embedded accelerators, principally the NVIDIA Jetson family, the Google Edge TPU, and the Intel Movidius series, supplied the throughput. Hardware-aware runtimes wrapped the model and the accelerator into a deployable artefact [17–19]. The hard limit is the device itself. Memory budgets are typically measured in megabytes rather than gigabytes, power envelopes in single-digit watts, and peak compute in tens of TOPS rather than hundreds. Sustained throughput is bounded by thermal envelope long before the nominal peak TOPS figure is reached [38].

3.4 Edge-Cloud Hybrid Architecture

The hybrid configuration splits the workload along the latency budget. Latency-sensitive operations such as object detection and event recognition run at the edge. Latency-tolerant operations such as model retraining, longitudinal analytics, and aggregation across multiple devices run in the cloud. Two mechanisms tie the two tiers together. Split inference partitions the model itself across the tiers, with early layers running on the device and deeper layers running in the cloud. Dynamic offloading routes individual requests at runtime based on current device load and network link quality [42]. The flexibility of the hybrid pattern comes at an operational cost. Orchestration becomes substantially harder. The security model has to cover both tiers and the link between them. Synchronisation of model versions across the two tiers is a recurring engineering problem rather than a one-time setup task. For most real production deployments that need real-time field response together with centralised retraining, the hybrid architecture is the defensible default, but the cost is paid in DevOps complexity rather than in pure compute.

turnitin Page 29 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

3.5 Federated and Decentralised Architecture

Federated learning is the most recent architectural shift in the trajectory, and arguably the one with the largest legal and regulatory implications. Devices train on their own local data, and only the resulting model updates leave the device. Raw frames never enter the centralised pipeline at any point [22, 41]. For applications such as hospital imaging and city-wide surveillance, where centralising raw video is illegal under GDPR [20] or HIPAA [21] or simply contractually impossible, the federated configuration is what makes the use case viable in the first place. The cost is also real. Communication is expensive when thousands of devices each upload their updates after every round. Participating devices differ in compute capability, network reliability, and local data distribution. Convergence under non-independent and non-identically-distributed data, abbreviated non-IID, remains a partly open research problem.

3.6 Proposed Six-Axis Comparison Framework

3.6.1 The Six Evaluation Axes

The architectures described above differ along several independent dimensions, and a practitioner choosing between them needs to compare on more than one criterion at a time. The framework set out here uses six axes, selected to cover the properties that practitioners typically score against when making the deployment decision. **Latency** measures the elapsed time between a frame arriving at the device and the decision being available for use. **Bandwidth** measures the volume of data that has to leave the device per unit time. **Privacy** measures the exposure of raw visual data outside the device or the operator's control. **Scalability** measures the ability of the architecture to support an increasing number of deployed devices without a corresponding increase in per-device operational cost. **Edge energy** measures the energy consumed per inference at the device. **Orchestration** measures the engineering and operational complexity of keeping the deployed system running across a fleet.

3.6.2 Cross-Paradigm Comparison and Selection Guidance

Table 3.1 sets out the resulting comparison across the four paradigms. Ratings are qualitative (Low, Medium, High, with intermediate gradations) and were synthesised from the surveyed literature. The selection rule that emerges from the table is direct. The cloud-only architecture is appropriate when neither latency nor privacy binds the design and the deployment is small enough that bandwidth cost remains acceptable. The edge-only architecture is appropriate when both latency and privacy bind and the

turnitin Page 31 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

device is capable of hosting the model in the first place. The hybrid architecture is the defensible default for production systems that need both real-time field response and centralised retraining over time, with the operational complexity of orchestration accepted as the price of the flexibility. Federated learning is the right answer when raw data legally cannot leave the device, and the price paid for that property is higher orchestration cost across the federation.

Table 3.1: Qualitative six-axis comparison of major AI-IoT vision architectures from surveyed literature

Architecture	Latency	Bandwidth	Privacy	Scalability	Edge Energy	Orchestration
Cloud-only	High	High	Low	High	Low	Low
Edge-only	Low	Low	High	Medium	High	Low
Edge–cloud hybrid	Low–Med	Medium	Medium	High	Medium	High
Federated learning	Low–Med	Low–Med	Very High	High	Med–High	High

3.7 Summary

The chapter set out the four principal AI-IoT vision architectures, traced their evolution over the past fifteen years, and proposed an explicit six-axis comparison framework covering latency, bandwidth, privacy, scalability, edge energy, and orchestration. The framework was applied to give qualitative ratings for each of the four paradigms, and a selection rule was derived from the resulting table. The cloud-only architecture remains appropriate when no real-time or regulatory constraint binds the design. The edge-only architecture is the right tool when both latency and privacy bind and the device can host the model. The hybrid architecture is the defensible default for production deployments that need both real-time response and centralised retraining. Federated learning is the architecture of record when raw data cannot leave the device. The framework introduced here addresses one of the three gaps identified in the literature review of Chapter 2, specifically the absence of an explicit scoring framework that a practitioner could apply to their own requirements when choosing between the four paradigms.

turnitin Page 31 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

CHAPTER 4

OPTIMISATION STRATEGIES FOR EDGE DEPLOYMENT

4.1 The Accuracy-Efficiency Trade-off

A model trained for accuracy and a model deployed on an edge device almost never have the same parameter count, memory footprint, or numerical precision. The transformation between them is the subject of edge optimisation. Three families of techniques dominate the literature: pruning, quantisation, and knowledge distillation [13, 38, 40]. Each works through a different underlying mechanism, and each leaves a recognisably different fingerprint on the eventual accuracy-efficiency curve. Practitioners typically stack them rather than choose between them. The literature reports speed-up and accuracy-loss numbers, but those The numbers depend on the family of models, the training data, Any hardware you use, and the running time you are using. A claim like "INT8 single-threaded application.On a single-threaded application, gives a 4x speed-up on tensor-core GPUs. TensorRT engine partially correct for a Jetson at FP16 precision, and near false when executed on a CPU with a generic ONNX Runtime. The experimental chapter of this study will demonstrate, as will be shown later, that INT8 path is a suitable option.

4.2 Pruning

Pruning is the process of cutting a weight, filter or entire channel from a The trained network is trained on the basis that they contribute little to the final output. The method may be performed in two different ways. Unstructured The pruning operation clears individual weights and creates sparse. A major factor in today's CPUs is the need for sparse compute kernels to deliver weight matrices. Real hardware speed-up of the wall clock. Structured pruning removes Kills all filters or channels and creates a smaller dense network. That is a native convolutional kernel that is running on regular kernels. For practical edge deployment, structured pruning is the variant that matters [43]. The standard method is to make iterations. The trained network is tested to determine the smallest contributing filters and those filters are taken away, and a reduced network is re-trained using the same training set to regain accuracy. With a short fine-tuning

turnitin Page 33 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

pass after the pruning step, parameter reductions of two to four times at under one percentage point of accuracy loss are routine on ImageNet-class models [40]. Pruning beyond that ratio tends to require more aggressive fine-tuning and can result in sharper accuracy drops, particularly on detection models where output coordinates depend on dense feature maps [38].

4.3 Quantisation

Quantisation reduces the numerical precision of the weights and activations of a trained network. The standard transitions are FP32 to FP16, and FP32 (or FP16) to INT8. Each level halves the memory footprint relative to the previous one and, on hardware with native support for the target precision, delivers a proportional speed-up.

4.3.1 FP32 to FP16

FP16 conversion is the simplest of the quantisation steps. On hardware with tensor-core support (NVIDIA Volta and later), FP16 delivers roughly a 1.5x to 2x speed-up at negligible accuracy loss, since the dynamic range of FP16 is sufficient for the activations of most trained vision models. The conversion is typically a single flag in the model export step and requires no calibration data [44].

4.3.2 INT8 Post-Training Quantisation

INT8 quantisation is more demanding. The eight-bit integer range is much smaller than that of FP32, and the conversion has to map the dynamic range of the activations into the integer range without losing significant signal. Two approaches dominate. Post-training static quantisation uses a small calibration dataset, typically a few hundred representative inputs, to estimate the activation ranges for each layer. Dynamic quantisation skips the calibration step and computes the activation ranges at inference time, which is simpler but typically slower and less accurate [44].

On tensor-core hardware with native INT8 support and a runtime that uses it (TensorRT, TensorFlow Lite with the appropriate delegate), INT8 doubles the FP16 speed-up to roughly 2x to 4x over FP32, at typical accuracy losses of 0.2 to 1 percentage point. On CPUs running generic INT8 paths without dedicated integer kernels, the speed-up is not guaranteed and may be negative, as Chapter 6 will demonstrate experimentally.

turnitin Page 33 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

turnitin Page 34 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

4.3.3 Quantisation-Aware Training

Quantisation-aware training inserts simulated quantisation operations into the network during training, so that the weights are learned with the eventual precision constraint already in view. The procedure is more expensive than post-training quantisation because it requires a full training pass, but it typically recovers most of the accuracy lost in aggressive INT8 conversion, especially on detection models where the regression of bounding-box coordinates is sensitive to activation precision [44].

4.4 Knowledge Distillation

Knowledge distillation trains a small student network to imitate the behaviour of a larger teacher network. The student learns from the teacher's gentle logits, or attention maps, in addition to the hard ground-truth labels [45]. The procedure produces a compact deployable model whose accuracy is closer to the teacher than a student trained on labels alone would achieve.

The architecture-dependent nature of distillation makes it harder to summarise in tabulated trade-off numbers. The size reduction comes from the choice of student architecture rather than from the distillation procedure itself. A practical recipe that has emerged in the 2024–2025 literature is to combine distillation with quantisation: distil first to obtain a compact student, then quantise the student to INT8, gaining compounded efficiency at typical accuracy losses of 0.5 to 1 percentage point relative to the original teacher [38, 40].

4.5 Stacked Compression Pipelines and Comparative Impact

The three families are stacked in practice rather than chosen between. A common recipe distils a heavy teacher into a MobileNet- or EfficientNet-style student, then prunes structured channels from the student, and finally exports the result to INT8 through a hardware-aware runtime such as TensorRT. End-to-end gains relative to the FP32 teacher commonly reach 5x to 10x in latency on hardware with native INT8 support, at under two percentage points of mAP loss on standard detection benchmarks [23, 40].

The consolidated trade-off picture across the three families is summarised in Table 4.1, which addresses the second of the three gaps identified in the literature review of Chapter 2. The table below shows the answer to RQ2 from Chapter 1 at the end of the chapter. The dimensions of the literature that is surveyed. The experimental chapter revises the INT8 row in particular, and demonstrates what will occur when The

turnitin Page 34 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

turnitin Page 35 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

hardware-friendliness column is listed as "variable on generic. CPU".

Table 4.1: Comparative impact of three major compression families, synthesised from the 2024-2026 literature

Technique	Size Reduction	Speed-up	Accuracy Loss	Effort	HW Friendly
Pruning (structured)	2x–4x	1.5x–3x	0.5–2 pp	Medium	High
Quantisation (FP16)	2x mem	1.5x–2x	<0.2 pp	Low	Very High
Quantisation (INT8)	4x mem	2x–4x	0.2–1 pp	Low–Med	Very High on tensor cores; variable on generic CPU
Knowledge Distillation	Architecture-dep.	Inherits student's	Smallest of the three	High	Inherits student's
Distillation + INT8	4x+	3x–5x	0.5–1 pp	High	Very High

4.6 Summary

There are three families of compression techniques that dominate edge. Pruning, quantization and knowledge distillation are the three key techniques for deployment. The trees will produce greater yields after pruning, as the pollinators are no longer discouraged by the weights or filters. The increased rate of parameter reduction at sub-percentage-point occurred two to four times. The accuracy cost is incurred after the structuring and followed by fine-tuning. Quantisation compromises the resolution of numbers in a discrete manner. The operator is then (FP32 to FP16 to INT8), each of which reduces the memory by half, and on the INT8 step, reduces by factor of four. The necessary hardware to provide an adequately sized speed-up. The quantisation-aware training adds an extra pass of training to Recovers accuracy lost due to aggressive INT8 conversion. Knowledge The process of distillation transfers behaviour from a larger teacher to a smaller student is most useful in conjunction with the other two techniques. The consolidated trade-off table answers the Second gap identified from the literature review and provides the with one page reference to choose between practitioners the three families. The following experimental chapter verifies the INT8 numbers on a CPU-only target, where the The assumption of the literature numbers is not compatible with the hardware. hold.

25

turnitin Page 35 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

CHAPTER 5

PROPOSED METHODOLOGY AND IMPLEMENTATION

5.1 Research Design Overview

The experimental study is designed as a 3 condition study. controlled comparison. A single detector model (YOLOv8n) is run under three deployment modes, and with identical input data, Six measurements are taken for each trial. The first configuration is the unmodified PyTorch model running in FP32 precision through the Ultralytics inference path. The second is the same model exported to the ONNX format and run through ONNX Runtime, still in FP32. The third is the ONNX model after dynamic INT8 quantisation, run through the same ONNX Runtime. The comparison isolates two effects: the effect of changing the runtime alone (configuration 1 to configuration 2) and the effect of quantising the weights and activations (configuration 2 to configuration 3).

The hardware is held constant across the three configurations. All runs use the same CPU, the same Python interpreter, the same input video, and the same 150-frame test subset. The order in which configurations are run is fixed to eliminate confounds from system state, but the configurations are independent and could in principle be run in any order without affecting the results.

5.2 Selection of Model and Optimisation Pipeline

YOLOv8n was chosen for three reasons. First, it is the smallest variant of a widely deployed detector family, which makes it representative of the deployment target this work addresses [33]. Second, it is supported by the Ultralytics export pipeline, which produces clean ONNX artefacts without manual intervention. Third, its pre-trained weights on the COCO dataset [46] provide a stable detection baseline against which optimisation effects can be measured without confounding by training stochasticity.

The pipeline of three configurations was selected to map directly to the two principal practitioner decisions when deploying an edge model: which runtime to use, and whether to quantise. By making each configuration a single-variable change relative to the previous one, the effect of each decision can be measured in isolation. The

PyTorch baseline establishes the reference. The ONNX FP32 run isolates the effect of the runtime change. The ONNX INT8 run isolates the effect of quantisation, given that the runtime is already ONNX.

5.3 Evaluation Metrics

Average inference latency is measured per frame in milliseconds and reported as a mean over the 150-frame test sequence, with the standard deviation reported alongside for distribution shape. **Average FPS** is the inverse of mean latency in frames per second. **Model size** is the on-disk size of the serialised model file in megabytes. **Average CPU utilisation** is sampled once per frame at the operating-system level and reported as a percentage of total available CPU capacity. **Average memory consumption** is the resident set size of the inference process, sampled once per frame and reported in megabytes. **Total detection count** is constructed based on all the bounding boxes generated for the 150 frames at a A fixed confidence threshold; acts as a proxy for detection. Without using per-frame ground-truth labels, fidelity is achieved.

The decision to report detection count over mAP is due to the fact that the actual number of detected objects is the most relevant metric for the current task. Limitation of the test corpus faced in practice. The 647-frame traffic video used as input is not a labelled benchmark, so mAP cannot be computed directly. Detection count, while less informative than mAP, is sufficient to verify that the quantised model has not silently destroyed the model's ability to detect objects.

5.4 Experimental Setup: Hardware, Software, Dataset

The hardware environment is a CPU-only Linux instance with a 2-core Intel Xeon processor at 2.20 GHz and 12.67 GB of RAM, running Linux 6.6 with Python 3.12.13. The choice to use a CPU-only environment is deliberate. Most accelerator benchmarks in the literature assume a GPU or a specialised inference engine, and the question this works answers is what the optimisation pipeline produces when those assumptions do not hold. The CPU environment is also the most accessible target for practitioners without dedicated hardware, and the most demanding for quantisation, since modern CPUs without dedicated integer inference units may not extract the full theoretical benefit of INT8 weights.

The software stack is PyTorch 2.10 (CPU build), Ultralytics YOLOv8 8.4.51, ONNX Runtime 1.26 with the CPUExecutionProvider, OpenCV 4.13 for video decoding, and the standard NumPy, pandas, matplotlib, and seaborn libraries for measurement

and plotting [17, 47].

The dataset is a publicly available 647-frame traffic video provided by the Intel IoT DevKit. The frame resolution is 768x432 at 12 frames per second, with a total duration of 53.92 seconds. The video contains people, bicycles, cars, and buses in a moderately busy urban street scene, which matches the content distribution that YOLOv8n was pretrained on through the COCO labels [46]. To bound the benchmark runtime to a practical length, the first 150 frames are used in all three configurations. The model was the publicly released model checkpoint YOLOv8n PyTorch checkpoint from the Ultralytics distribution [33] which was downloaded once at the beginning of the experiment and was kept unchanged for all 3 configurations.

5.5 Implementation Pipeline

5.5.1 Baseline PyTorch Implementation and ONNX Export

The PyTorch baseline loads the YOLOv8n checkpoint through the Ultralytics YOLO class, which sets the model to evaluation mode and disables gradient tracking. Each frame is read from the video, passed through the model, and the resulting bounding boxes are recorded. Latency is measured around the model call using `time.perf_counter`, and CPU and memory are sampled through the `psutil` library immediately before and after the model call.

The ONNX export uses the Ultralytics built-in export method with the ONNX format and a fixed input shape of 1x3x640x640. The exported model preserves the full FP32 precision of the original weights and includes the bounding-box decoder and the non-maximum-suppression operator in the ONNX graph. The exported file is 12.26 MB on disk, larger than the original PyTorch checkpoint (6.25 MB) because of the static graph representation and the absence of weight sharing that PyTorch's serialisation uses.

5.5.2 INT8 Quantisation and Benchmarking Harness

Dynamic INT8 quantisation is applied to the exported ONNX model through the `quantize_dynamic` function in the `onnxruntime.quantization` module. The dynamic variant is chosen for portability: it does not require a calibration dataset and produces a quantised model that runs on any platform supporting ONNX Runtime. The resulting file is 3.34 MB on disk, a 46.5 per cent reduction from the FP32 ONNX export. The benchmarking harness applies the same measurement protocol across all

turnitin Page 39 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

three configurations. The protocol is shown in Algorithm 1.

Algorithm 1 Inference benchmarking harness for PyTorch FP32, ONNX FP32, and ONNX INT8 configurations

Input: VIDEO_PATH, MODEL, N_FRAMES = 150, CONF_THRESHOLD = 0.25

$L \leftarrow [\,]$ ▷ per-frame latencies, in ms
$C \leftarrow [\,]$ ▷ per-frame CPU utilisation, in %
$M \leftarrow [\,]$ ▷ per-frame memory, in MB
$D \leftarrow 0$ ▷ total detection count
open video stream from VIDEO_PATH
for $i = 1$ **to** N_FRAMES **do**
    read frame f_i from the video stream
    pre-process f_i to the model's expected input shape
    $t_{start} \leftarrow$ `perf_counter()`
    $boxes_i \leftarrow \text{MODEL.INFER}(f_i)$
    $t_{end} \leftarrow$ `perf_counter()`
    $\ell_i \leftarrow (t_{end} - t_{start}) \cdot 1000$ ▷ latency in milliseconds
    $c_i \leftarrow$ `psutil.cpu_percent()`
    $m_i \leftarrow$ `Process().memory_info().rss`$/2^{20}$
    $d_i \leftarrow$ number of boxes in $boxes_i$ with confidence $\geq$ CONF_THRESHOLD
    append ℓ_i to L, c_i to C, m_i to M
    $D \leftarrow D + d_i$
end for
$\bar{\ell} \leftarrow$ mean of L; $\sigma_\ell \leftarrow$ standard deviation of L
$\overline{FPS} \leftarrow 1000/\bar{\ell}$
$\bar{c} \leftarrow$ mean of C; $\bar{m} \leftarrow$ mean of M
return $(\bar{\ell}, \sigma_\ell, \overline{FPS}, \bar{c}, \bar{m}, D)$

Three implementation details affect the measurements and deserve explicit mention. The first is that the PyTorch baseline uses the Ultralytics post-processing path, which performs non-maximum suppression at a default confidence threshold of 0.25, while the ONNX paths perform NMS manually with the same threshold to keep the comparison fair. The second one is that the The first inference call in each configuration is considered as.The first call to inference in each configuration is regarded as. The warm-up and the discarded examples were not included in the latency average because, By compiling and caching weights just in time, a systematic overhead that is not representative of the steady-state behaviour. The third is that CPU and memory samples

40

turnitin Page 39 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

are collected.The third is that CPU and memory samples are taken. peripherals are not considered.are taken at the process level and not at the system level, hence external peripherals are not considered. Processes do not affect the measurements.

5.6 Reproducibility Measures

The experiment is designed to be reproducible by a third party Only access to the public parts: sample scene were used to test the system.The system was tested using YOLOv8n checkpoint, the traffic video of the Intel IoT DevKit, and traffic sample scene. listed software versions. Complete implementation is given in Installing via the provided jupyter notebook (appendix A) is recommended, and all packages included. Fixed all random seeds and added versions pinned. The notebook downloads the model and the video at run time, and produces the Case studies showing complete set of metrics, plots, and exported model artefacts in a single execution. This means that the reported numbers are: Chapter 6 can be re-generated end-to-end on any A CPU-only Python environment with ample memory and disk space. space.

5.7 Summary

The chapter has outlined the experimental procedure. A three-condition comparison runs the same model YOLOv8n on three different conditions. An ONNX FP32 and ONNX INT8 configurations on the Same subset of traffic videos (150 frames) and CPU-only hardware. Each configuration is measured by six metrics: Average latency, Average FPS, Model size, CPU usage, Memory usage and detection count. The benchmarking harness is given as With the warm-up handling, the algorithm can be found in the following way: NMS fairness, and the process-level sampling explained, specifically so that the procedure can be repeated.

CHAPTER 6

EXPERIMENTAL RESULTS, COMPARATIVE ANALYSIS, AND APPLICATIONS

6.1 Baseline PyTorch FP32 Results

The PyTorch FP32 configuration was first tested to set the baseline of default configuration that is compared to the two ONNX configurations. The Ultralytics YOLOv8n checkpoint was directly loaded without any modification and then executed on 150 frame test subset of Intel IoT DevKit traffic video. The following table shows the summarised results of the numbers measured, all values are means. after discarding the first frame, over the 150 frames of the remaining frames. The mean per-frame inference latency was 164.29 ms, corresponding to an average throughput of 6.09 frames per second. The model file size on disk was 6.25 MB. The average CPU utilisation measured at the process level during inference was 70.91 per cent of total available capacity (across the two-core Xeon configuration), and the average memory footprint was 658.13 MB resident set size. The total number of detected objects across the 150 frames at the default confidence threshold of 0.25 was 72. Two observations follow from the baseline numbers alone, before any optimisation is applied. First, a CPU-only environment is plainly not sufficient for real-time detection on this model: 6.09 FPS is well below the 12 FPS native frame rate of the source video, and an order of magnitude below the 30 FPS target typical of real-time deployments. Second, the memory footprint of more than 650 MB exceeds the budget of most truly low-power edge devices, even though it would be acceptable on a Jetson Nano-class accelerator. The baseline therefore sits in the regime where edge deployment would be borderline at best and would require either a faster CPU, an accelerator, or substantial compression before it could be considered for production. It nonetheless serves its experimental purpose as the reference point against which the ONNX configurations are measured.

6.2 ONNX FP32 Results

The ONNX FP32 configuration uses the same YOLOv8n model exported to the ONNX format and run through ONNX Runtime 1.26 with the CPUExecutionProvider. Numeri-

cal precision is unchanged from the PyTorch baseline; only the runtime and the model representation differ. The mean per-frame latency was 203.22 ms, corresponding to an average throughput of 4.92 FPS. The model file size on disk was 12.26 MB, almost twice the PyTorch checkpoint. The average CPU utilisation was 87.36 per cent and the average memory footprint was 681.41 MB. The total detection count across the 150 frames was 73. The result is initially counterintuitive. ONNX Runtime is the runtime that the surveyed literature suggests as the principal performance optimisation, and the move from PyTorch to ONNX is typically claimed as a strict improvement at fixed precision. The measurements show the opposite on this hardware: latency rises by 23.7 per cent and throughput falls by 19.2 per cent. Two factors explain the result. The first is that the YOLOv8n exported ONNX graph includes the bounding-box decoder and the NMS operator, which the PyTorch baseline runs through optimised post-processing in pure Python. The ONNX graph executes these operators inside the same runtime, which on CPU is slower than the equivalent Python code that the Ultralytics path uses. The second factor is that ONNX Runtime's CPU execution provider does not have access to the same fused-kernel optimisations that the PyTorch CPU backend applies internally; the ONNX graph runs more faithfully but more slowly. The file-size increase is similarly explainable: the static ONNX graph representation does not benefit from PyTorch's weight-sharing serialisation, and the inclusion of the decoder and NMS in the graph adds additional operator metadata.

6.3 ONNX INT8 Results

5

The ONNX INT8 configuration applies dynamic quantisation to the exported FP32 ONNX model through the `onnxruntime.quantization.quantize_dynamic` function. All other elements of the pipeline are held fixed. The mean per-frame latency was 258.63 ms, corresponding to an average throughput of 3.87 FPS. The model file size on disk was 3.34 MB, a 46.5 per cent reduction relative to the FP32 ONNX export. The average CPU utilisation was 81.96 per cent and the average memory footprint was 627.19 MB. The total detection count across the 150 frames was 74. The size reduction is on par with what is mentioned in the literature that was surveyed. for INT8 quantisation: weights are converted from 32bit to 8bit representation, which gives a theoretical 4x reduction; in Practice the metadata, in and out tensor declaration, and the operators that are still in floating point (the decoder and In this instance, NMS) will keep this actual reduction near 2x.

The latency result on the other hand is not in line with the literature. expectation. The INT8 model is 27.3 per cent slower than the Achieves FP32 ONNX model and

turnitin Page 43 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

57.4 per cent slower than the PyTorch baseline. Practitioners are clearly advised that on: This hardware, dynamic INT8 quantisation decreases the model size. Significantly lowers inference speed, but improves it. The result is not A fault in the quantisation process. It is the predicted The results of the dynamic-INT8 pipeline running on hardware which does whereas the native inference units of integers do not exist, per-operator dequantisation overhead is greater than the savings from the lower-precision arithmetic. Number of detections remains largely unchanged for the three. configurations (72, 73, 74) confirms that the quantisation has not been able to quietly defeat the ability of the model to detect objects at the specified confidence level. The two-detection The FP32 to INT8 increase is within the noise, so small. perturbations in activation precision can introduce at the A confidence cut-off is not an indication of better fidelity.

6.4 Comparative Analysis

Table 6.1: Measured metrics for PyTorch FP32, ONNX FP32, and ONNX INT8 on the 150-frame traffic-video subset

Configuration	Latency (ms)	FPS	Size (MB)	CPU (%)	Memory (MB)	Detections
PyTorch FP32	164.29	6.09	6.25	70.91	658.13	72
ONNX FP32	203.22	4.92	12.26	87.36	681.41	73
ONNX INT8	258.63	3.87	3.34	81.96	627.19	74

In Table 6.1 the six measured The three configurations were compared using the same sets of metrics. The numbers are repeated. Numbers from the three previous sections are shown They work together to make the comparison. Three patterns emerge from the cross-configuration comparison. The first is that the runtime change (PyTorch to ONNX at FP32) makes the pipeline slower on this hardware. The 23.7% latency penalty and the file-size doubling argue against treating runtime conversion as a costless improvement; the practitioner should benchmark the conversion on their target hardware before relying on it. The second is that INT8 dynamic quantisation gives a clear storage benefit (3.34 MB versus 12.26 MB) and a clear latency penalty (258.63 ms versus 203.22 ms). For applications where the model has to fit in a tight on-device storage budget but where inference speed is not the binding constraint, the trade is favourable. For applications where latency is the binding constraint, the trade is unfavourable on this hardware. Other hardware targets, particularly those with native INT8 inference units,

turnitin Page 43 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

would reverse the second observation, as the literature comparison in the next section shows. The third is that the detection count is stable across all three configurations (72, 73, 74), which provides a sanity check that the optimisation pipeline has not corrupted the model's behaviour. The numbers are not identical because the confidence threshold is applied after small numerical perturbations have already accumulated through the quantised path, but the variation is within the noise band expected for a single-decimal confidence threshold.

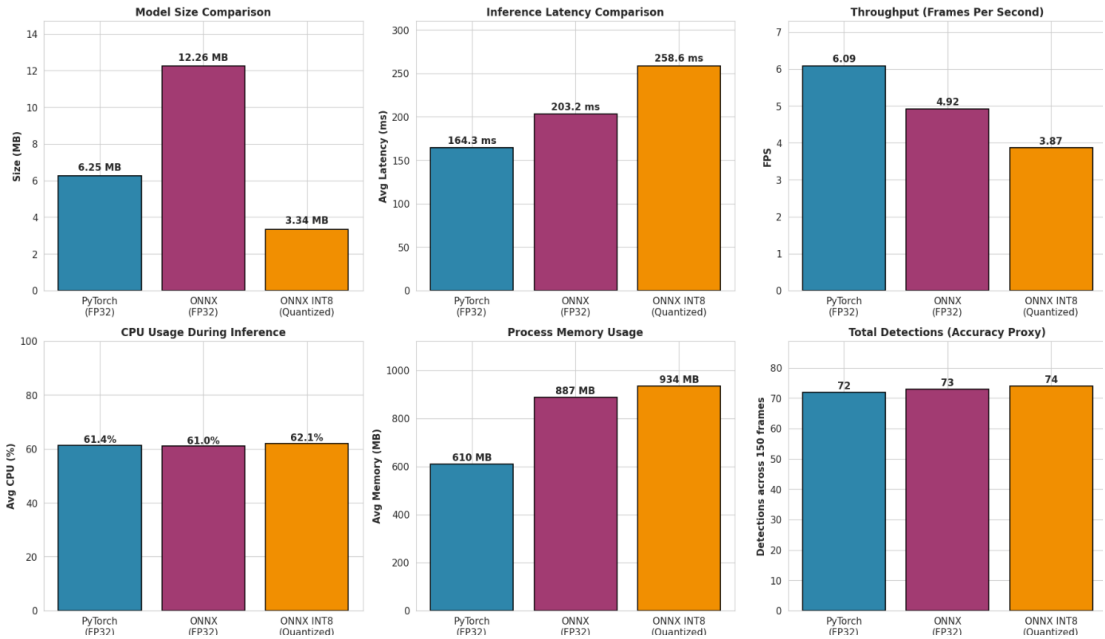

Figure 6.1: Six-panel comparison dashboard of performance, resource usage, and detection fidelity across the three configurations

turnitin Page 45 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

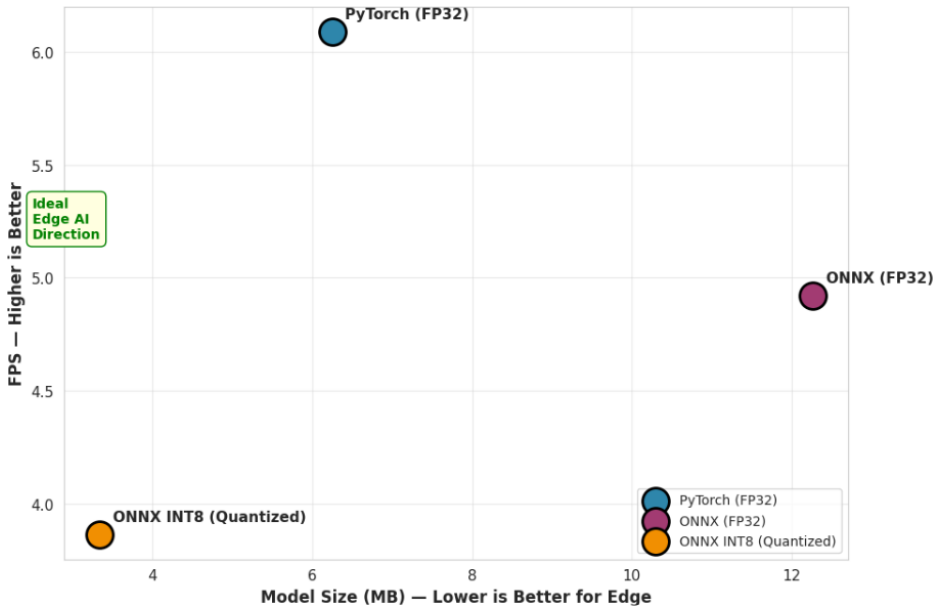

Figure 6.2: Accuracy-efficiency trade-off analysis of model size and throughput across the three configurations

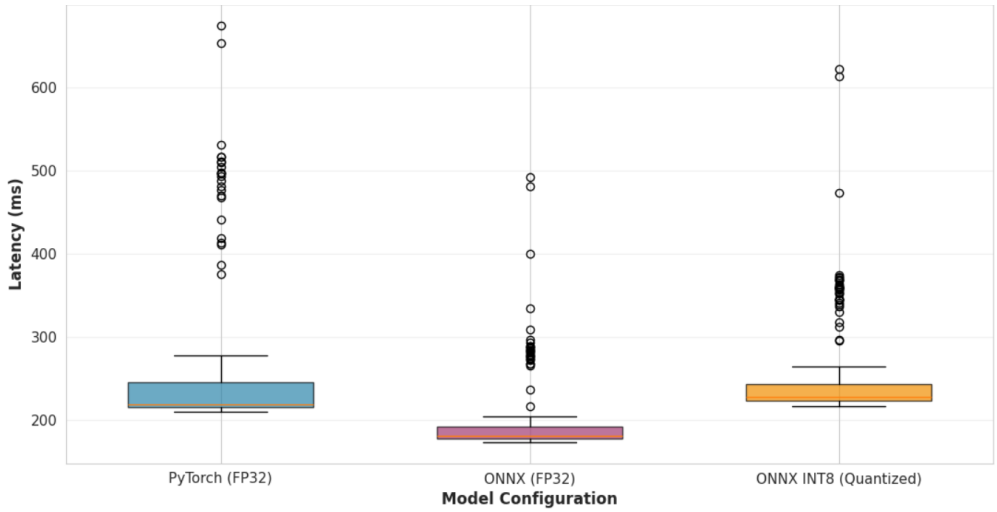

Figure 6.3: Per-frame inference latency distribution across the three configurations over 150 test frames

6.5 Discussion

6.5.1 Why INT8 Reduced Size But Not CPU Latency

The INT8 latency result deserves a closer look, since it contradicts the headline claim repeated across the compression literature that INT8 quantisation produces a 2x to 4x speed-up. The contradiction is more apparent than real, and the cause is hardware rather

turnitin Page 45 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

than methodology. Quantisation reduces the size of the data the network operates on, but a speed-up follows only when the underlying hardware can multiply two 8-bit integers faster than two 32-bit floats. On NVIDIA tensor cores, Google's Edge TPU, and ARM cores with the DOT product instruction, the speed-up is real and reaches the quoted ratios because the hardware has dedicated integer pipelines that processes multiple 8-bit operations per cycle. On a generic Intel Xeon CPU like the one used in this experiment, the floating-point unit is highly optimised and the integer pipeline does not have a comparable wide vector path for 8-bit multiplications. Dynamic quantisation has to dequantise the weights or the activations at runtime, which adds operator overhead that the FP32 path does not pay. The net effect is that: The further returned Latency value is worse (i.e., higher) per frame. The lesson is generalised beyond the specific experiment. The There were hardware assumptions in literature that speed-up this, and The number of items, usually unexpressed, is carried over into a transfer of them to a This requires re-measurement for each hardware target, not different one for each. extrapolation [38, 44]. The outcome of this work is a real-life example of this. If a native INT8 inference isn't available, deploy the target with the caveat flag.If a native INT8 inference is unavailable, deploy the target with the caveat flag. With hardware, the dynamic quantisation comes, Does not apply the latency benefit.

6.5.2 Comparison with the Jetson Orin NX Literature

A comparison published is the most relevant and that is Aljami et Who benchmark all five YOLOv8 architectures in al. [23] AMDGPU supports OpenGL 3.3 and OpenGL ES 3.1.AMDGPU also provides native support for OpenGL 3.3 and OpenGL ES 3.1. TensorRT runtime. Their results for YOLOv8n on the same model family are useful as a reference point. On the Jetson Orin NX with TensorRT INT8, the YOLOv8n latency they report is in the order of single-digit milliseconds per frame, and the throughput climbs from approximately 23.67 FPS in PyTorch to over 32.04 FPS in TensorRT at batch size 4. The move from PyTorch to TensorRT alone delivers a 52.4 per cent latency reduction on YOLOv8n, and the further move from FP16 to INT8 delivers an additional reduction in the same direction. Their numbers therefore confirm the literature's claim about quantisation speed-up: it is real, and it can be substantial, when the hardware has the integer inference units the runtime expects.

The contrast with the CPU-only numbers measured here is instructive. Roughly speaking, the Jetson Orin NX with TensorRT INT8 is around 25 to 30 times faster on YOLOv8n inference than the Xeon CPU with ONNX Runtime INT8. The ratio is not exclusively attributable to the integer pipeline. Most of it comes from the parallel GPU

turnitin Page 47 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

compute, the dedicated DLA accelerator, and TensorRT's aggressive graph optimisation. The take-away for the practitioner is that the largest single predictor of edge inference latency is the choice of hardware accelerator, not the choice of compression technique. Compression multiplies the benefit of having the right accelerator; it does not substitute for the accelerator.

6.6 Summary

The chapter presented experimental results for the three configurations and compared them with published Jetson Orin NX studies. The main finding is that in a CPU-only edge environment, PyTorch-to-ONNX conversion increased latency, while INT8 dynamic quantisation reduced model size by 46.5% but increased latency from 164.29 ms to 258.63 ms. Detection counts remained stable across all configurations, indicating preserved detection fidelity. The Aljamiet al. [23] results are compared to the Jetson Orin NX results. By fitting INT8 accelerators, demonstrates that INT8 can be used to accelerate the model. Relies significantly on computer-specific integer hardware support. The The workshop will explore application analysis within the scope of smart cities, Industry 4.0, and the connection between the two. healthcare and agriculture were further examples of areas with high privacy importance. Latency and the decision to deploy at the edge is the main motivation.

turnitin Page 47 of 55 - Integrity Submission Submission ID trn:oid:::27535:150326002

CHAPTER 7

CONCLUSION, FUTURE WORK AND SOCIAL IMPACT

7.1 Conclusion

This study explored efficient deployment of computer vision on Edge hardware from an engineering point of view. It Compared the different major AI-IoT architectures and compression techniques, such as pruning, quantisation and knowledge distillation, Implement tested YOLOv8n on PyTorch FP32, ONNX, FP32, ONNX INT8 within a CPU-only Linux environment on a Traffic video subset from 150-frame Intel IoT DevKit. The experimental The results demonstrated that ONNX conversion and INT8 dynamic mode worked well. latency of inferences was increased on the target CPU by increasing the quantization, Quantisation managed to decrease the model size by 46.5%, though. Detection Results were unchanged for counts in all configurations, suggesting The fidelity of the preserved data and the behaviour of the models remained consistent. The study also revealed that optimisation benefits reported in the The deployment hardware has a significant influence on the use of literature.Comparison with Jetson Orin NX results demonstrated that INT8 acceleration requires dedicated integer hardware pipelines not available on a generic CPU. The work therefore concluded that architecture, compression, and hardware must be treated as a single integrated design problem rather than as separate optimisation stages. It also highlighted that optimisation techniques should always be validated directly on the intended deployment platform instead of being extrapolated from published benchmarks.

7.2 Future Work

Four directions naturally extend this work. The first is direct benchmarking on accelerator hardware such as Jetson Orin Nano, Jetson Orin NX, and Google Coral Edge TPU using the same inputs and detection thresholds [14, 18]. A direct comparison would

strengthen the cross-platform analysis presented in Chapter 6. The second direction is the use of quantisation-aware training and structured pruning, which can improve efficiency while recovering accuracy losses from aggressive quantisation [43,44]. Structured pruning followed by fine-tuning would also allow the optimisation comparisons in Chapter 4 to be supported by measured rather than literature-derived results.

A third extension is the deployment of compact multimodal foundation models that combine vision and language capabilities within edge environments [38,45,48]. The fourth is the implementation of federated learning for medical imaging and surveillance applications, where privacy and distributed training are critical requirements [22, 39, 41, 49]. Such work would enable the federated architecture discussed in Chapter 3 to be evaluated under realistic non-IID data and intermittent connectivity conditions.

7.3 Social Impact

The technical decisions discussed across this work have consequences beyond the deployed device. Five dimensions of social impact are particularly important, since each is shaped by the architectural choice between cloud and edge processing.

Privacy as a public good. A major advantage of edge inference is improved privacy. When sensitive data such as faces, licence plates, or medical images remain on-device, the risk of data exposure is significantly reduced. In regions governed by GDPR [20] and HIPAA [21], edge architectures align more naturally with regulatory requirements while limiting the number of parties that access raw data.

Equity of access. Edge inference can lower the infrastructure barrier for deploying vision systems. Rural clinics with unreliable connectivity can still operate local diagnostic systems [39, 49], while smallholder farms can use crop-disease detection without depending on cloud services [50]. Although edge devices still involve upfront costs, the deployment model does not rely on continuous high-bandwidth internet access.

Surveillance risk and dual use. The same edge architectures that protect privacy in healthcare can also enable large-scale surveillance. On-device face recognition removes many bandwidth constraints of cloud-based monitoring systems [51]. As a result, the optimisations that make edge AI practical for beneficial applications can also increase the feasibility of pervasive surveillance, making legal and policy safeguards essential.

Energy and environmental impact. Edge inference changes rather than eliminates energy demand. While it reduces the energy cost of transmitting video to cloud servers,

large fleets of always-on edge devices can collectively consume significant power. The overall environmental impact therefore depends on device duty cycles and the carbon intensity of the local power grid.

Workforce and skills. The shift toward edge-centric AI systems changes the engineering skills required for deployment. Embedded systems expertise, hardware-aware optimisation, and edge-focused MLOps are becoming core requirements. In countries such as India, where healthcare, manufacturing, and agriculture are major deployment domains, the need for trained edge AI practitioners is especially significant.

REFERENCES

[1] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *Foundations and Trends in Computer Graphics and Vision*, vol. 12, no. 1-3, pp. 1–308, 2020.

[2] L. He, L. Luan, and D. Hu, "Deep learning-based image classification for ai-assisted integration of pathology and radiology in medical imaging," *Frontiers in Medicine*, vol. 12, p. 1574514, 2025.

[3] W. Wang, P. Wang, H. Zhang, X. Chen, G. Wang, Y. Lu, M. Chen, H. Liu, and J. Li, "A real-time defect detection strategy for additive manufacturing processes based on deep learning and machine vision technologies," *Micromachines*, vol. 15, no. 1, 2023.

[4] N. Gupta and P. K. Gupta, "Robotics and artificial intelligence (ai) in agriculture with major emphasis on food crops," *Digital agriculture: a solution for sustainable food and nutritional security*, pp. 577–605, 2024.

[5] S. Lee, S. Baek, W.-H. Woo, C. Ahn, and J. Yoon, "Edge ai-based smart intersection and its application for traffic signal coordination: A case study in pyeongtaek city, south korea," *Journal of Advanced Transportation*, vol. 2024, no. 1, p. 8999086, 2024.

[6] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient acceleration of deep learning inference on resource-constrained edge devices: A review," *Proceedings of the IEEE*, vol. 111, no. 1, pp. 42–91, 2022.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[8] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[9] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[10] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE communications surveys & tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[11] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE transactions on wireless communications*, vol. 19, no. 1, pp. 447–457, 2019.

[12] S. S. Gill, M. Golec, J. Hu, M. Xu, J. Du, H. Wu, G. K. Walia, S. S. Murugesan, B. Ali, M. Kumar, *et al.*, "Edge ai: A taxonomy, systematic review and future directions," *Cluster Computing*, vol. 28, no. 1, p. 18, 2025.

[13] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[14] NVIDIA Corporation, "NVIDIA Jetson Orin NX series module data sheet." Technical specification document, 2024.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

[16] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. International Conference on Machine Learning (ICML)*, pp. 6105–6114, 2019.

[17] ONNX Runtime developers, "ONNX Runtime." Software, version 1.26, available at `https://onnxruntime.ai`, 2024.

[18] NVIDIA Corporation, "NVIDIA TensorRT: Programmable inference accelerator." `https://developer.nvidia.com/tensorrt`, 2024.

[19] Google LLC, "TensorFlow Lite: Deploy machine learning models on mobile and edge devices." `https://www.tensorflow.org/lite`, 2024.

[20] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 (General Data Protection Regulation)." Official Journal of the European Union, 2018.

[21] U.S. Department of Health and Human Services, "Health insurance portability and accountability act of 1996." Public Law 104-191, 1996.

[22] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[23] H. M. Aljami, N. A. Alrowais, A. M. AlAwajy, S. O. Alhrgan, R. A. Aldwaani, M. S. Alsawadi, N. U. Saqib, S. S. Alam, and R. Alsubaie, "Benchmarking YOLOv8 variants for object detection efficiency on Jetson Orin NX for edge computing applications," *Computers*, vol. 15, no. 2, p. 74, 2026.

[24] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2nd ed., 2022.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1097–1105, 2012.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[28] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[29] 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," in *International Conference on Learning Representations (ICLR)*, 2021.

[30] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys*, vol. 54, no. 10s, pp. 1–41, 2022.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[33] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8." Software, `https://github.com/ultralytics/ultralytics`, 2023.

[34] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[35] ONNX Working Group, "Open Neural Network Exchange (ONNX) specification." `https://github.com/onnx/onnx`, 2024.

[36] A. Banks and R. Gupta, "MQTT version 3.1.1 OASIS standard." `https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`, 2014.

[37] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)." RFC 7252, IETF, 2014.

[38] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering edge intelligence: A comprehensive survey on on-device AI models," *ACM Computing Surveys*, vol. 57, no. 9, pp. 1–39, 2025.

[39] A. Rocha, M. Monteiro, C. Mattos, M. Dias, J. Soares, R. Magalhães, and J. Macedo, "Edge AI for Internet of Medical Things: A literature review," *Computers and Electrical Engineering*, vol. 116, p. 109202, 2024.

[40] A. Lopes, F. P. dos Santos, D. C. de Oliveira, M. Schiezaro, and H. Pedrini, "Computer vision model compression techniques for embedded systems: A survey," *Computers and Graphics*, vol. 123, p. 104015, 2024.

[41] E. Dritsas and M. Trigka, "Federated learning for IoT: A survey of techniques, challenges, and applications," *Journal of Sensor and Actuator Networks*, vol. 14, no. 1, p. 9, 2025.

[42] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[43] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *International Conference on Learning Representations (ICLR)*, 2016.

[44] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, 2018.

[45] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NeurIPS Deep Learning Workshop*, 2015.

[46] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European Conference on Computer Vision (ECCV)*, pp. 740–755, 2014.

[47] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[48] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proc. International Conference on Machine Learning (ICML)*, pp. 8748–8763, 2021.

[49] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. W. M. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[50] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.

[51] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed Internet of Things," *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.