Energy-Aware TinyML for Ambient-Powered, Hardware-Constrained IoT Nodes

Anakha Krishnavilasom Gopalakrishnan

Master's Thesis 2026

Department of Electrical Engineering
Information and Communication Technology, MPICT
Chalmers University of Technology
Gothenburg, Sweden

06 August 2026

Thesis Information

Title

Energy-Aware TinyML for Ambient-Powered, Hardware-Constrained IoT Nodes

Author

Anakha Krishnavilasom Gopalakrishnan

Master's Programme

Information and Communication Technology (MPICT)

Department

Department of Electrical Engineering

Chalmers University of Technology

Examiner

Prof. Giuseppe Durisi

Chalmers University of Technology

Thesis Supervisor

Javad Aliakbari

Chalmers University of Technology

Industrial Supervisor

Navid Amani

Emickers AB

06 August 2026, Gothenburg, Sweden

Abstract

Resource-constrained Internet of Things (IoT) devices are increasingly expected to perform intelligent sensing under strict limitations in memory, computation, and energy. For image-based applications, transmitting raw images can consume substantially more communication energy and time than transmitting a compact latent representation. This thesis investigates lightweight convolutional autoencoders for on-device image compression on microcontroller-class hardware.

A design space of 70 convolutional autoencoder models was evaluated on the MNIST dataset. The models differed in encoder filter configuration, bottleneck dimension, and compression ratio. Reconstruction quality and task utility were measured by normalized mean squared error (NMSE) and downstream classifier accuracy. The baseline models were further optimized using post-training quantization (PTQ), quantization-aware training (QAT), and a two-stage magnitude-based pruning method. Different optimization orders were compared with respect to model size, reconstruction quality, classifier accuracy, and estimated energy.

An analytical energy model was used to estimate microcontroller active energy associated with encoder computation and payload transfer under explicitly stated assumptions. Pareto-front analysis identified non-dominated candidate models offering different trade-offs between reconstruction quality and estimated energy. A selected INT8 encoder was deployed on an Ambiq Apollo3 microcontroller using TensorFlow Lite for Microcontrollers and the NeuralSPOT software stack. Hardware experiments confirmed successful encoder inference and provided measured inference-time results.

The results show that bottleneck size and the choice of optimization method strongly influence the trade-off between reconstruction quality, payload size, and deployment cost. Quantization reduced numerical precision and memory footprint, while unstructured pruning introduced sparsity but did not necessarily reduce execution time on dense microcontroller kernels. The study therefore carefully distinguishes analytical energy estimates from measured hardware results.

This work provides a reproducible framework for comparing learned compression models on resource-constrained embedded devices. The evaluation is limited to MNIST data, analytical communication-energy assumptions, host-side serial data transfer, and a single microcontroller platform. A complete autonomous energy-harvesting and wireless IoT

implementation remains future work.

Keywords: TinyML, Energy-Aware Machine Learning, Autoencoder, Model Compression, Energy Harvesting, Pareto Optimization, IoT

Acknowledgements

This thesis was conducted at the Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden.

First and foremost, I would like to express my sincere gratitude to my examiner, Prof. Giuseppe Durisi, for providing me with the opportunity to work on this research topic and for his valuable guidance throughout the project.

I am really grateful to Prof. Henk Wymeersch for his valuable instructions his continuous support, and to my supervisor, Javad Aliakbari, for his insightful feedback, support, and guidance.

I am especially thankful to my industrial supervisor, Navid Amani at Emickers AB, for sharing his practical knowledge and guidance.

I would also like to acknowledge the resources and support provided by Chalmers University of Technology.

I am also grateful to my family and friends for their great support and encouragement throughout this journey.

Anakha Krishnavilasom Gopalakrishnan
Gothenburg, 06 August 2026

AI Declaration

I acknowledge the use of OpenAI's language model, ChatGPT, for assistance with grammar checking, language editing, creating illustrative figures used in the theoretical background and improving the clarity of this report. All research ideas, methodology, implementation, analysis, all experimental results, plots, tables, and conclusions are based on the author's own work and experiments.

The author reviewed, verified, and takes full responsibility for all the content presented in this thesis.

Anakha Krishnavilasom Gopalakrishnan
Gothenburg, 06 August 2026

List of Acronyms

AE — Autoencoder

CAE — Convolutional Autoencoder

IoT — Internet of Things

TinyML — Tiny Machine Learning

MCU — Microcontroller Unit

PSNR — Peak Signal to Noise Ratio

SSIM — Structural Similarity Index

MAC — Multiply Accumulate Operation

TFLite — TensorFlow Lite

INT8 — 8-bit Integer

FP32 — 32-bit Floating Point

PTQ — Post Training Quantization

QAT — Quantization Aware Training

Contents

List of Figures

1 Introduction

1.1 Background

The Internet of Things (IoT) has enabled the large-scale deployment of connected sensing devices for applications such as environmental monitoring, industrial automation, precision agriculture, healthcare, and smart cities. Many of these devices operate under strict constraints on energy, memory, and computational resources, particularly when deployed in remote or difficult-to-access locations where battery replacement is costly or impractical. Consequently, improving the energy efficiency of embedded sensing systems has become an important research challenge for extending the operational lifetime of resource-constrained IoT devices [7, 21].

Recent advances in Tiny Machine Learning (TinyML) have made it possible to execute machine learning models directly on low-power microcontrollers, enabling data to be processed locally rather than transmitted to cloud servers. Local inference reduces communication latency, lowers communication overhead, and improves data privacy. However, microcontroller class platforms provide limited processing capability, memory capacity, and energy resources, making the deployment of deep learning models particularly challenging [20, 2].

For image-based sensing applications, wireless communication is often a major contributor to the overall energy consumption because raw images contain a large amount of data. Instead of transmitting complete images, learned image compression using autoencoders provides an attractive alternative by generating a compact latent representation on the sensing device. The encoder performs the compression on the resource-constrained node, while the decoder can be executed on a gateway or edge server to reconstruct the transmitted image. This approach significantly reduces the communication payload while preserving useful information for downstream processing [19, 1].

Several model optimization techniques, including post-training quantization (PTQ), quantization aware training (QAT), and pruning, have been widely adopted to improve the deployment efficiency of deep learning models on embedded hardware. These techniques reduce memory requirements and computational complexity, making TinyML models more suitable for resource-constrained devices. However, the resulting trade-offs between reconstruction quality, model size, computational complexity, and energy consumption depend strongly on both the network architecture and the selected optimization strategy.

This thesis systematically investigates these trade-offs using lightweight convolutional autoencoders for image compression on resource-constrained embedded platforms. A design space of baseline models was evaluated, multiple optimization techniques were compared, and selected models were assessed using an analytical energy model together with hardware deployment on an Ambiq Apollo3 microcontroller. The experimental evaluation presented in this thesis is limited to the implemented hardware platform and the methodology described in Chapter 3. These challenges motivate the review of existing research on learned image compression, TinyML deployment, and model optimization techniques presented in the following section.

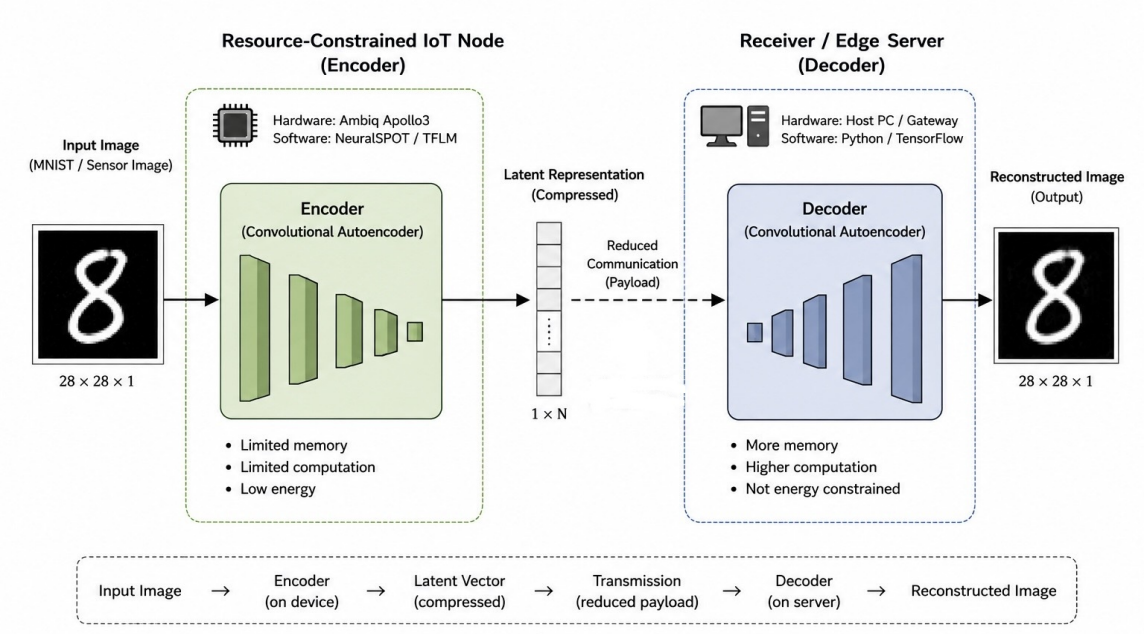

Figure 1: Overview of the proposed TinyML image compression framework. The encoder executes on the embedded device, while the decoder reconstructs the image at the receiver.(Created by the author)

1.2 Related Work

Previous research related to this thesis can be grouped into three main areas: TinyML deployment on microcontroller-class devices, learned image compression using autoencoders, and model optimization techniques such as quantization and pruning. This section reviews these areas and identifies the gap addressed in this thesis.

Learned image compression using convolutional autoencoders has received considerable attention. These models learn compact latent representations that preserve the information required for reconstruction while significantly reducing the amount of data that

must be transmitted. Compared with traditional codecs, learned compression can adapt its representation to the data distribution and has demonstrated competitive performance at low bit rates [19, 1].

Although deep autoencoders provide promising compression performance, their computational and memory requirements often exceed the capabilities of resource-constrained devices. Consequently, substantial research has focused on model optimization techniques. PTQ converts trained floating-point parameters to lower numerical precision after training, while QAT incorporates quantization effects during training. Network pruning removes redundant parameters or filters to reduce model complexity [8, 11, 16].

Several studies have demonstrated the successful deployment of quantized neural networks on embedded hardware using TensorFlow Lite for Microcontrollers and related TinyML frameworks. These studies report significant reductions in memory footprint and computational cost while maintaining acceptable inference performance for classification and detection tasks. However, the effectiveness of a given optimization strategy depends strongly on the network architecture, hardware platform, and application requirements [4, 2].

For image-compression applications, most existing studies primarily evaluate reconstruction quality or compression ratio. Relatively few works jointly investigate reconstruction quality, deployment efficiency, communication payload size, and energy cost on the same resource-constrained platform. In particular, the combined effects of quantization and pruning on convolutional autoencoders, together with both analytical energy estimation and hardware deployment, have received limited attention.

This thesis addresses this gap by systematically evaluating a design space of lightweight convolutional autoencoders, applying multiple optimization strategies (including different ordering of quantization and pruning), estimating microcontroller active energy under explicit assumptions, and demonstrating deployment feasibility of a selected INT8 encoder on an Ambiq Apollo3 microcontroller using TensorFlow Lite for Microcontrollers.

1.3 Research Gap and Motivation

Despite significant progress in TinyML, learned image compression, and model optimization, several challenges remain for deploying deep learning models on resource-constrained embedded platforms. Existing studies have primarily focused on improving either compression performance or model efficiency independently. Comparatively fewer studies

have jointly investigated the trade-offs between image reconstruction quality, model complexity, communication payload size, and energy consumption within a unified evaluation framework for embedded deployment.

Furthermore, previous research has largely evaluated model optimization techniques such as PTQ, QAT and pruning in isolation. Limited attention has been given to systematically comparing these techniques including different optimization orders using a common baseline architecture and a consistent evaluation methodology. This makes it difficult to identify the most suitable deployment strategy for a given resource-constrained application.

Motivated by these limitations, this thesis investigates lightweight convolutional autoencoders for image compression on embedded devices. A systematic design-space exploration was performed by evaluating multiple baseline architectures together with different model optimization techniques. Reconstruction quality, classifier accuracy, model size, and estimated energy consumption were jointly analyzed to understand the trade-offs between compression performance and deployment efficiency.

The experimental evaluation is intentionally limited to the MNIST dataset and deployment of the encoder on an Ambiq Apollo3 microcontroller. Communication energy is estimated using an analytical energy model under explicitly stated assumptions, while hardware validation focuses on demonstrating successful encoder deployment and measuring inference time. Consequently, the results should be interpreted as an evaluation of deployment-efficient learned image compression for resource-constrained embedded systems rather than a complete validation of an autonomous batteryless or wireless IoT platform.

The findings of this study are intended to provide practical guidance for selecting lightweight autoencoder architectures and optimization strategies under different resource and energy constraints, while establishing a foundation for future investigations that incorporate real image datasets, wireless communication, and energy-harvesting embedded systems.

1.4 Research Questions and Scope

The primary objective of this thesis is to investigate the deployment of lightweight convolutional autoencoders for image compression on resource-constrained embedded platforms. The study evaluates the trade-offs between image reconstruction quality, model complexity, communication efficiency, and deployment cost while assessing the effective-

ness of different model optimization techniques.

To achieve this objective, the following research questions are addressed:

1. **RQ1:** How do different convolutional autoencoder architectures, with varying filter configurations, bottleneck dimensions, and compression ratios, affect image reconstruction quality, compression performance, and deployment efficiency on resource-constrained embedded platforms?

2. **RQ2:** What are the effects of PTQ, QAT, magnitude-based pruning, and their different combinations and optimization orders on reconstruction quality, model size, computational complexity, and estimated energy consumption?

3. **RQ3:** Can an optimized INT8 convolutional autoencoder be successfully deployed on an Ambiq Apollo3 microcontroller while satisfying the memory and inference-time constraints of the target platform?

To answer these research questions, a systematic design-space exploration was performed using the MNIST dataset. Lightweight convolutional autoencoder models with different architectural configurations were first evaluated as baseline models. The selected model optimization techniques were then applied and compared using normalized mean squared error (NMSE), classifier accuracy, model size, and estimated energy consumption. Pareto-front analysis was subsequently used to identify suitable deployment candidates, and a selected INT8 encoder was deployed on an Ambiq Apollo3 microcontroller to verify deployment feasibility and measure inference time.

The scope of this thesis is limited to the evaluation of convolutional autoencoders using the MNIST dataset, analytical estimation of encoder computation and communication energy under explicitly stated assumptions, and hardware deployment of the encoder on a single embedded platform. The study does not experimentally evaluate wireless communication, camera-based image acquisition, autonomous energy-harvesting operation, or runtime adaptive model selection. These aspects are considered promising directions for future work.

1.5 Main Contributions

The main contributions of this thesis are summarized as follows:

1. A systematic design-space exploration of lightweight convolutional autoencoders for image compression on resource-constrained embedded platforms. Multiple baseline

architectures with different encoder filter configurations, bottleneck dimensions, and compression ratios were evaluated to characterize the trade-offs between reconstruction quality, compression performance, total energy consumption and deployment efficiency.

2. A comparative evaluation of model optimization techniques, including PTQ, QAT, magnitude-based pruning, and their different optimization orders. The optimized models were assessed using NMSE, classifier accuracy, model size, and estimated energy consumption.

3. An analytical evaluation framework that jointly estimates the energy associated with encoder computation and communication of the compressed latent representation. Pareto-front analysis was applied to identify non-dominated operating points under different resource and energy constraints.

4. Experimental validation of deployment feasibility by implementing a selected INT8 convolutional autoencoder on an Ambiq Apollo3 microcontroller using TensorFlow Lite for Microcontrollers and the NeuralSPOT software stack. Hardware experiments demonstrated successful deployment of the encoder and provided measured inference-time results on the target platform.

1.6 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 presents the theoretical background of the study. It introduces the fundamental concepts of TinyML and edge intelligence, energy-aware embedded systems, convolutional autoencoders, model optimization techniques, and Pareto optimization that provide the scientific foundation for this work.

Chapter 3 describes the research methodology. It presents the proposed TinyML image compression framework, the convolutional autoencoder architecture, the experimental setup, the model optimization techniques, the analytical energy model, the hardware platform, and the deployment methodology adopted in this study.

Chapter 4 presents and analyzes the experimental results. The performance of the baseline models, quantized models, pruned models, and combined optimization strategies is evaluated in terms of reconstruction quality, classifier accuracy, model size, estimated energy consumption, and hardware deployment results. Pareto-front analysis is then used

to identify deployment-efficient operating points.

Chapter 5 discusses the findings of the study by interpreting the experimental results, comparing them with related work, highlighting the limitations of the proposed approach, and discussing their implications for resource-constrained TinyML deployment.

Chapter 6 concludes the thesis by summarizing the main findings, answering the research questions, highlighting the primary contributions of the work, and suggesting directions for future research.

2 Theoretical Background

2.1 TinyML and Edge AI

TinyML is a branch of embedded artificial intelligence that enables machine learning models to execute directly on resource-constrained microcontrollers. Unlike conventional cloud-based machine learning, where data are transmitted to remote servers for processing, TinyML performs inference locally on the embedded device. Local inference reduces communication latency, lowers bandwidth requirements and enables real-time decision-making in applications where reliable network connectivity is unavailable or intermittent [20, 2].

Typical TinyML platforms are characterized by limited computational capability, memory capacity, and strict energy budgets. Many commercially available microcontrollers provide only a few hundred kilobytes of RAM and flash memory while operating at very low power. These constraints make the direct deployment of conventional deep learning models impractical and require lightweight neural network architectures together with model optimization and efficient deployment strategies [17, 13].

Edge Artificial Intelligence (Edge AI) extends this concept by performing computation close to the data source rather than relying entirely on cloud computing. Instead of transmitting all sensed data, Edge AI processes information locally and communicates only the required results or compressed representations. This approach reduces communication overhead, decreases response latency, and improves system reliability and privacy [18, 22].

Despite these advantages, deploying deep learning models on resource-constrained embedded platforms remains challenging because of limited processing capability, memory availability, and energy resources. Consequently, lightweight network architectures and model optimization techniques such as quantization and pruning are commonly employed to reduce computational complexity, memory footprint, and inference cost while maintaining acceptable model performance [11, 8].

In this thesis, TinyML provides the deployment platform for implementing lightweight convolutional autoencoders on an Ambiq Apollo3 microcontroller. The encoder performs image compression directly on the embedded device, while the more computationally demanding decoder executes on the receiver side. This architecture reduces the communication payload while enabling efficient deployment on resource-constrained hardware,

which is the primary focus of this thesis.

2.2 Convolutional Autoencoders

Autoencoders are a class of unsupervised neural networks designed to learn compact representations of input data by reconstructing the original input after passing it through a low-dimensional latent space. A typical autoencoder consists of two main components: an encoder and a decoder. The encoder compresses the input into a compact latent representation, while the decoder reconstructs the original input from this representation. During training, the network learns to minimize the reconstruction error between the input and the reconstructed output, thereby preserving the most informative features of the data [6, 9].

Convolutional Autoencoders (CAEs) extend conventional autoencoders by replacing fully connected layers with convolutional layers. Since convolutional layers exploit the spatial correlations present in images, CAEs generally achieve better reconstruction performance while requiring fewer trainable parameters than fully connected architectures. These characteristics make convolutional autoencoders particularly suitable for image compression and embedded vision applications [15, 19].

For image compression, the encoder transforms the input image into a compact latent representation whose dimensionality determines the compression ratio. A smaller latent space generally provides higher compression but may reduce reconstruction quality because of information loss. Conversely, a larger latent representation preserves more image information but increases the communication payload. Selecting an appropriate bottleneck dimension therefore requires balancing reconstruction quality against communication efficiency.

The decoder reconstructs the image from the latent representation. During training, the encoder and decoder are jointly optimized using a reconstruction loss function so that the latent representation captures the most relevant image features. After training, the encoder can be deployed on a resource-constrained embedded device, while the decoder executes on a more capable receiver or edge server. This partitioning significantly reduces the amount of transmitted data and makes convolutional autoencoders well suited for TinyML-based image compression.

In this thesis, lightweight convolutional autoencoders are used to compress MNIST images before transmission. Multiple architectures with different encoder filter configurations,

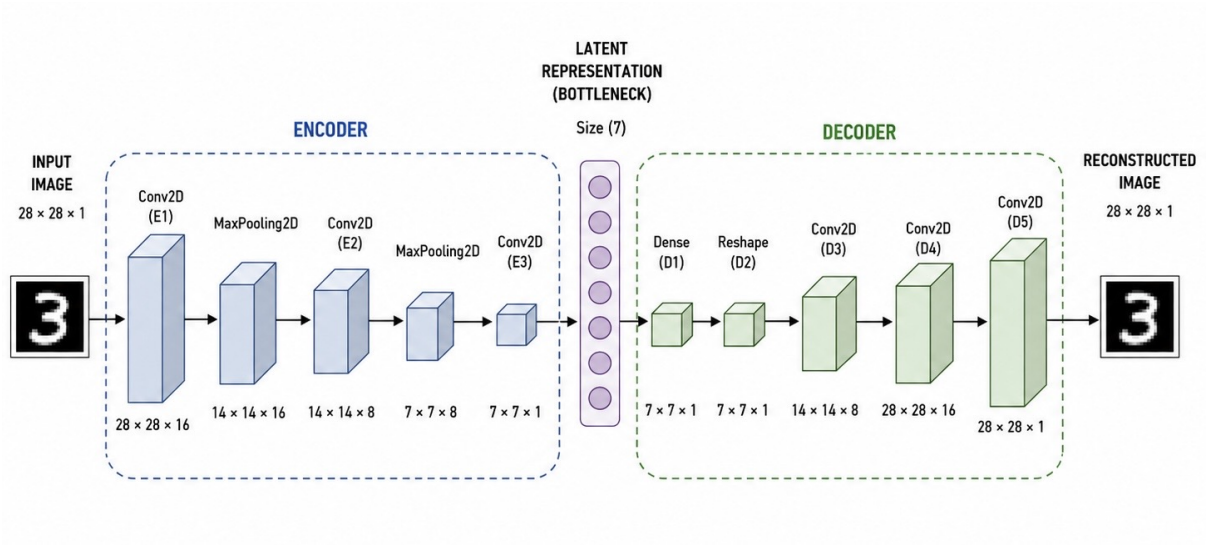

Figure 2: General architecture of a convolutional autoencoder used for image compression. The encoder compresses the input image into a compact latent representation, which is transmitted to the receiver where the decoder reconstructs the original image. Illustration created by the author based on [9, 15, 19].(Created by author)

bottleneck dimensions, and compression ratios are systematically evaluated to investigate the trade-offs between reconstruction quality, communication efficiency, and deployment cost. The selected encoder is subsequently deployed on an Ambiq Apollo3 microcontroller, while the decoder is executed on the receiver, reflecting the deployment strategy adopted throughout this work.

2.3 Model Optimization Techniques

Deep learning models are typically developed and trained using floating-point arithmetic on high performance computing platforms. However, directly deploying these models on resource-constrained embedded devices is often impractical because of limited memory capacity, computational capability, and energy budget. Consequently, model optimization techniques are widely used to reduce model size, computational complexity, and inference cost while maintaining acceptable model performance.

Among the various optimization techniques proposed in the literature, quantization and pruning have become two of the most widely adopted approaches for TinyML applications [8, 11, 16].

Quantization reduces the numerical precision of model parameters and activations, typically from 32-bit floating-point (FP32) to 8-bit integer (INT8) representations. This decreases memory usage and enables more efficient integer arithmetic on microcontrollers.

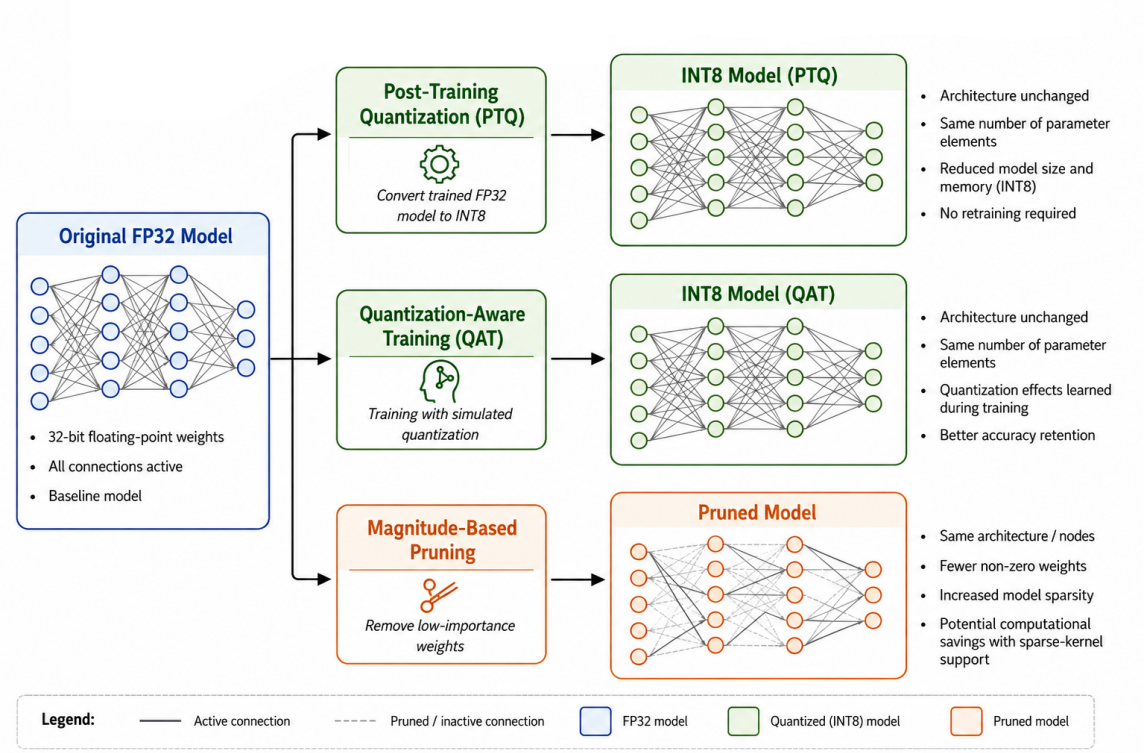

Figure 3: Overview of the model optimization techniques investigated in this thesis. A baseline FP32 model is optimized using PTQ, QAT, or magnitude-based pruning. PTQ and QAT produce INT8 models with reduced memory requirements, while pruning removes less important weights to reduce model complexity. (Illustration created by the author)

Two quantization approaches are considered in this thesis:

- **PTQ** converts a trained floating-point model to lower precision after training. It is simple to apply and requires no retraining, although it may result in some degradation in model performance.
- **QAT** simulates quantization effects during training, enabling the model to adapt to reduced numerical precision and generally preserving higher reconstruction quality after quantization.

Pruning removes parameters or filters that contribute little to the final prediction. In this work, a magnitude-based pruning strategy is employed, where weights with small absolute values are removed according to a predefined threshold. Pruning reduces model complexity and memory requirements, although unstructured pruning does not necessarily lead to faster execution on dense microcontroller kernels.

Quantization and pruning can also be combined to further improve deployment efficiency. For example, pruning may be applied before quantization, or quantization may be followed

by pruning. The effectiveness of each optimization strategy and its ordering depends on the neural network architecture, deployment constraints, and target hardware platform.

In this thesis, PTQ, QAT, and magnitude-based pruning are investigated both individually and in combination. Their impact is evaluated with respect to reconstruction quality, classifier accuracy, model size, and estimated energy consumption in order to identify deployment-efficient convolutional autoencoder models.

Figure 3 illustrates the model optimization techniques investigated in this thesis. Starting from a baseline floating-point model, PTQ, QAT, and magnitude-based pruning are applied to improve deployment efficiency on resource-constrained embedded devices.

2.3.1 PTQ

PTQ is a model optimization technique that converts a trained floating-point neural network into a lower-precision representation after training has been completed. Unlike other optimization methods, PTQ does not require retraining of the model, making it a simple and computationally efficient approach for deployment on resource-constrained embedded devices [11, 16].

The most common form of PTQ converts FP32 weights and activations into INT8 representations. This significantly reduces the model memory footprint and enables efficient integer arithmetic during inference. As a result, PTQ decreases storage requirements, reduces memory bandwidth, and can improve inference efficiency on hardware platforms that support integer operations.

To perform PTQ, a representative calibration dataset is typically used to estimate the dynamic ranges of activations. These ranges are then used to determine appropriate scaling factors and zero-points that map floating-point values to integer representations while minimizing quantization error [11].

Although PTQ is straightforward to apply and requires no additional training, the reduced numerical precision may introduce quantization errors that degrade model performance. The magnitude of this degradation depends on the neural network architecture, the characteristics of the dataset, and the sensitivity of different layers to quantization.

In this thesis, PTQ is applied to the trained convolutional autoencoder models after training. The quantized models are evaluated using NMSE, classifier accuracy, model size, and estimated energy consumption to assess their suitability for deployment on the

Ambiq Apollo3 microcontroller.

2.3.2 QAT

QAT is a model optimization technique in which quantization effects are incorporated during the training process. Unlike PTQ, where a trained floating-point model is quantized after training, QAT simulates low-precision arithmetic throughout training. This enables the neural network to adapt to quantization-induced errors, thereby improving the performance of the final quantized model [11, 16].

During QAT, the model is trained using floating-point parameters while fake quantization operations simulate the numerical behaviour of low precision arithmetic during both the forward and backward passes. The gradients are still computed in floating-point precision, allowing the optimization process to compensate for the errors introduced by quantization. As a result, the final model is generally more robust to reduced numerical precision than a model produced using PTQ.

Compared with PTQ, QAT typically achieves higher inference accuracy after quantization, particularly for deep neural networks and applications that are sensitive to numerical precision. However, this improved performance comes at the cost of additional computational effort because the model must be retrained with quantization simulated throughout the optimization process.

In this thesis, QAT is applied to the baseline convolutional autoencoder models to generate deployment-efficient INT8 models. The resulting models are evaluated using NMSE, classifier accuracy, model size, and estimated energy consumption. Their performance is then compared with both the baseline floating-point models and the PTQ models to assess the effectiveness of quantization-aware training for embedded image compression on the Ambiq Apollo3 microcontroller.

2.3.3 Magnitude-Based Pruning

Pruning is a model optimization technique that removes parameters or filters that contribute little to the predictive capability of a neural network. The primary objective of pruning is to reduce model complexity, memory requirements, and computational cost while maintaining acceptable model performance. By eliminating redundant parameters, pruning enables more efficient deployment of deep learning models on resource-constrained embedded devices [8, 3].

Among the various pruning approaches proposed in the literature, magnitude-based pruning is one of the most widely adopted because of its simplicity and effectiveness. The underlying assumption is that parameters with small absolute values contribute less to the network output than parameters with larger magnitudes. During pruning, weights or filters with magnitudes below a predefined threshold are removed or set to zero, producing a sparse neural network.

Pruning can be performed either before or after model training. In practice, iterative pruning followed by fine-tuning is commonly employed to recover part of the performance loss caused by removing network parameters. The overall effectiveness of pruning depends on the pruning strategy, pruning ratio, neural network architecture, and the deployment hardware.

In this thesis, a two-stage magnitude-based pruning strategy is applied to the convolutional autoencoder models. The pruned models are subsequently evaluated using NMSE, classifier accuracy, model size, and estimated energy consumption. Their performance is also compared with PTQ, QAT, and the combined optimization strategies to identify deployment-efficient models for the Ambiq Apollo3 microcontroller.

2.4 Energy Consumption in Embedded AI

Energy efficiency is one of the primary challenges in deploying artificial intelligence on resource-constrained embedded devices. Unlike cloud computing platforms, embedded systems typically operate with limited battery capacity or harvested energy, making energy consumption a critical design constraint. Consequently, reducing computational cost and communication overhead is essential for extending device lifetime and enabling long-term autonomous operation [18, 2].

The total energy consumption of an embedded AI application generally consists of two main components: computation energy and communication energy. Computation energy is consumed while executing neural network inference on the microcontroller and depends on factors such as processor architecture, clock frequency, supply voltage, and the number of arithmetic operations. Communication energy is associated with transmitting data between the embedded device and an external receiver. The required energy depends primarily on the size of the transmitted payload and the characteristics of the communication technology employed [10].

For image-based sensing applications, communication energy often represents a signifi-

cant portion of the overall energy budget because transmitting raw images requires substantially more data than transmitting compact feature representations. Learned image compression using convolutional autoencoders addresses this challenge by generating a compact latent representation on the embedded device, thereby reducing the amount of transmitted information while maintaining acceptable reconstruction quality.

Several studies have shown that jointly considering computation and communication is essential when evaluating deployment-efficient TinyML systems. Optimizing only the neural network complexity may reduce computation energy while leaving communication costs largely unchanged, whereas focusing solely on communication may increase computational overhead. Therefore, an effective deployment strategy should balance both aspects according to the application requirements and hardware constraints [2].

In this thesis, total energy consumption is evaluated using an analytical model that jointly considers the energy required for encoder computation and transmission of the compressed latent representation. The energy estimates are derived under explicitly stated assumptions and are used to compare different convolutional autoencoder architectures and model optimization techniques. Hardware validation is limited to successful encoder deployment and inference-time measurement on the Ambiq Apollo3 microcontroller. Direct measurement of consumed energy using an external power measurement device is outside the scope of this work and is left for future work.

2.5 Pareto Optimization

Many engineering design problems involve multiple objectives that conflict with one another. In TinyML-based image compression, improving one performance metric often degrades another. For example, increasing the compression ratio may reduce communication energy but can also decrease image reconstruction quality. Similarly, larger neural network models may improve reconstruction performance while requiring more memory, computation, and energy. Consequently, selecting a single optimal solution requires balancing multiple competing objectives.

Pareto optimization is a widely used multi-objective optimization approach for identifying solutions that represent the best trade-offs among conflicting objectives. A solution is considered Pareto optimal if no other solution can improve one objective without simultaneously degrading at least one other objective. The collection of all Pareto-optimal solutions is known as the Pareto front, where each solution represents a different com-

promise between competing performance metrics [5, 14].

In embedded machine learning, Pareto optimization is frequently used to evaluate the trade-offs between model accuracy, computational complexity, memory footprint, inference latency, and energy consumption. Instead of selecting a model based on a single metric, Pareto analysis enables designers to choose a deployment candidate that best satisfies the resource constraints and performance requirements of a particular application.

In this thesis, Pareto-front analysis is employed to compare different convolutional autoencoder architectures and model optimization techniques. The models are evaluated using reconstruction quality and estimated total energy consumption. The resulting Pareto-optimal solutions represent deployment-efficient operating points that provide different trade-offs between image quality and energy efficiency for resource-constrained embedded platforms.

2.6 Chapter Summary

This chapter presented the theoretical background that forms the foundation of this thesis. It introduced the concepts of TinyML and Edge AI, highlighting the challenges of deploying deep learning models on resource-constrained embedded devices. The principles of convolutional autoencoders were then discussed, emphasizing their suitability for learned image compression by generating compact latent representations for efficient data transmission. The chapter also reviewed the model optimization techniques investigated in this work, including PTQ, QAT, and magnitude-based pruning. Their roles in reducing model size, computational complexity, and memory requirements were explained together with their relevance to embedded deployment. Furthermore, the importance of jointly considering computation and communication energy was discussed, followed by an introduction to Pareto optimization as a multi-objective approach for identifying deployment-efficient models that balance reconstruction quality and energy consumption. The concepts presented in this chapter provide the theoretical basis for the methodology described in the next chapter, where the proposed TinyML image compression framework, experimental setup, model optimization procedures, analytical energy model, and hardware deployment methodology are presented in detail.

3 Methodology

3.1 Overview of the Proposed Framework

This chapter describes the methodology adopted to develop and evaluate the proposed TinyML image compression framework for resource-constrained embedded systems. The methodology consists of model development, systematic design-space exploration, model optimization, analytical energy evaluation, and hardware deployment. Figure 4 presents an overview of the complete workflow adopted in this study.

The framework begins with the MNIST dataset, which is used to train lightweight convolutional autoencoder models. A systematic design-space exploration is then performed by varying the encoder filter configurations, bottleneck dimensions, and compression ratios, resulting in 70 baseline architectures. These baseline models are evaluated using NMSE and classifier accuracy to characterize the trade-offs between reconstruction quality and compression performance.

The baseline models are subsequently optimized using PTQ, QAT, and a two-stage magnitude-based pruning strategy. Different optimization sequences are also investigated. The optimized models are compared with the baseline models in terms of NMSE, classifier accuracy, model size, and estimated energy consumption.

An analytical energy model is employed to estimate the combined energy required for encoder computation and transmission of the compressed latent representation under explicitly stated assumptions. Pareto-front analysis is then applied to identify non-dominated operating points that provide suitable trade-offs between reconstruction quality and estimated total energy consumption.

Finally, a selected INT8 convolutional autoencoder encoder is deployed on an Ambiq Apollo3 microcontroller using TensorFlow Lite for Microcontrollers (TFLM) and the NeuralSPOT software stack. Hardware experiments validate successful deployment and measure inference time, demonstrating the feasibility of the proposed framework for resource-constrained embedded AI applications.

3.2 Dataset and Preprocessing

The MNIST handwritten digit dataset was used throughout this study to train and evaluate the proposed convolutional autoencoder models. MNIST is one of the most

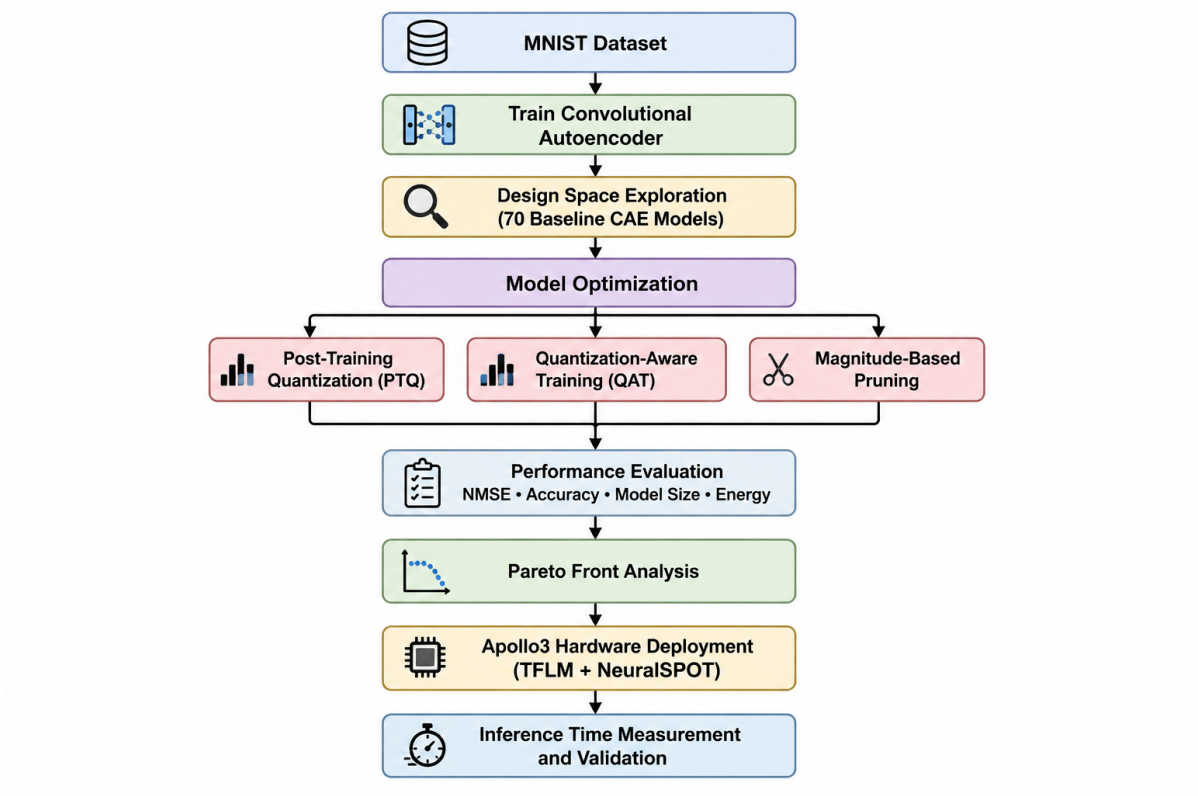

Figure 4: Overview of the methodology adopted in this study. The workflow begins with the MNIST dataset and training of convolutional autoencoder models, followed by design-space exploration of 70 baseline architectures. The models are then optimized using PTQ, QAT, and magnitude-based pruning. After performance evaluation and Pareto-front analysis, a selected INT8 encoder is deployed on the Ambiq Apollo3 microcontroller for inference-time measurement and hardware validation. (Illustration created by the author)

widely used benchmark datasets for image classification and image reconstruction tasks. It consists of 70,000 grayscale images of handwritten digits (0–9), of which 60,000 images are provided for training and 10,000 images are reserved for testing. Each image has a resolution of 28×28 pixels with a single grayscale channel [12].

Before training, the pixel values of all images were normalized to the range of 0 to 1 by dividing each pixel value by 255. Image normalization improves numerical stability during training and facilitates faster convergence of the optimization algorithm.

The same normalized images were used as both the input and target output during autoencoder training because the objective of the network is to reconstruct the original image after compressing it into a lower-dimensional latent representation. No data augmentation techniques were applied, since the objective of this study was to evaluate the image compression capability of lightweight convolutional autoencoders rather than improve classification performance.

The trained encoder models were subsequently converted into deployment ready formats for hardware implementation. For embedded deployment, representative datasets were used during post-training quantization to calibrate the activation ranges, while quantization-aware training incorporated quantization effects during the training process. All baseline and optimized models were evaluated using the same training and testing datasets to ensure a fair comparison.

3.3 Convolutional Autoencoder Architecture

The proposed image compression framework is based on a lightweight CAE designed for deployment on resource-constrained embedded devices. The network consists of two main components: an encoder and a decoder. The encoder compresses the input image into a compact latent representation, while the decoder reconstructs the original image from this latent representation.

The encoder comprises two convolutional layers followed by batch normalization and Rectified Linear Unit (ReLU) activation functions. The extracted feature maps are flattened and mapped to a fully connected bottleneck layer, which produces the latent representation. Different bottleneck dimensions were investigated to study the trade-off between compression ratio, reconstruction quality, and deployment efficiency.

The decoder reconstructs the original image by applying a fully connected layer followed by transposed convolutional layers. During training, the encoder and decoder are jointly optimized to minimize the reconstruction loss between the input and reconstructed images. After training, only the encoder is deployed on the Ambiq Apollo3 microcontroller, while the decoder is executed on the receiver. This partitioning reduces the communication payload while keeping the computational requirements of the embedded device low.

To investigate the influence of model complexity on deployment performance, a systematic design-space exploration was performed by varying the encoder filter configurations, bottleneck dimensions, and compression ratios. In total, 70 baseline convolutional autoencoder architectures were generated and evaluated. The resulting models were compared using NMSE, classifier accuracy, model size, and estimated energy consumption before applying model optimization techniques.

Figure 5 shows an example of the encoder architecture used in the baseline experiments. The encoder compresses the input MNIST image into a low dimensional latent repre-

Encoder architecture

Model: "encoder_f12_f21_bn2_lc7"

Layer (type)	Output Shape	Param #
input_layer_6 (InputLayer)	(None, 28, 28, 1)	0
conv2d_15 (Conv2D)	(None, 28, 28, 2)	20
max_pooling2d_6 (MaxPooling2D)	(None, 14, 14, 2)	0
conv2d_16 (Conv2D)	(None, 14, 14, 1)	19
max_pooling2d_7 (MaxPooling2D)	(None, 7, 7, 1)	0
bottleneck_conv (Conv2D)	(None, 7, 7, 2)	4
flatten_3 (Flatten)	(None, 98)	0
latent (Dense)	(None, 7)	693

Total params: 736 (2.88 KB)

Trainable params: 736 (2.88 KB)

Non-trainable params: 0 (0.00 B)

Figure 5: Example encoder architecture of a baseline convolutional autoencoder model. The red box highlights the bottleneck part of the encoder. (Figure created by the author)

sentation. The red box highlights the bottleneck part of the encoder. In this stage, the feature maps are further compressed, flattened, and mapped to the final latent vector. This latent vector represents the compressed image information that is transmitted from the embedded device instead of the full input image.

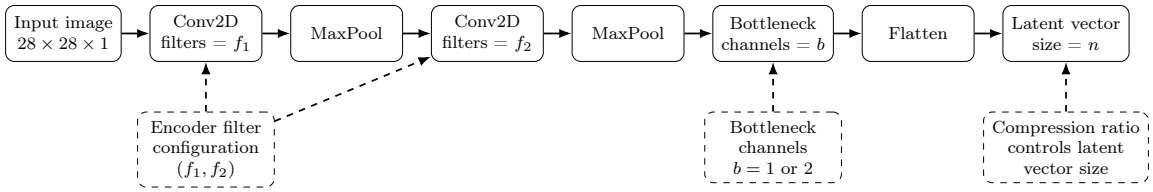

Figure 6: Relationship between the design-space parameters and the encoder architecture. The encoder filter configuration (f_1, f_2) controls the first and second convolutional layers, the bottleneck channel parameter controls the bottleneck layer, and the compression ratio controls the final latent-vector size.

Figure 6 explains how the design-space parameters are connected to the encoder architecture shown in Figure 5. The encoder filter configuration (f_1, f_2) defines the number of filters in the first and second convolutional layers. The bottleneck channel parameter controls the depth of the bottleneck representation. The compression ratio controls the size of the final latent vector, which represents the compressed data transmitted from the embedded device.

Decoder architecture

Model: "fixed_heavy_decoder_128"

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, 7)	0
dense_3 (Dense)	(None, 3136)	25,088
reshape_3 (Reshape)	(None, 7, 7, 64)	0
up_sampling2d_6 (UpSampling2D)	(None, 14, 14, 64)	0
conv2d_17 (Conv2D)	(None, 14, 14, 64)	36,928
up_sampling2d_7 (UpSampling2D)	(None, 28, 28, 64)	0
conv2d_18 (Conv2D)	(None, 28, 28, 32)	18,464
conv2d_19 (Conv2D)	(None, 28, 28, 1)	289

Total params: 80,769 (315.50 KB)

Trainable params: 80,769 (315.50 KB)

Non-trainable params: 0 (0.00 B)

Figure 7: Example decoder architecture used to reconstruct the image from the latent vector.(Figure created by the author)

The decoder, shown in Figure 7, reconstructs the image from the latent representation. In the considered deployment scenario, the encoder is assumed to run on the resource constrained IoT node, while the decoder can be executed either on a gateway, edge server, or more powerful receiver side device. This makes encoder complexity particularly important for TinyML deployment.

3.4 Design-Space Exploration

A systematic design-space exploration was conducted to investigate how the convolutional autoencoder architecture influences reconstruction quality, compression performance, model complexity, and estimated energy consumption. Instead of evaluating a single architecture, multiple lightweight baseline models were generated by varying the encoder filter configuration, bottleneck channel configuration, and compression ratio. As illustrated in Figure 6, the design-space parameters are directly connected to specific parts of the encoder architecture: the encoder filter configuration controls the two convolutional layers, the bottleneck channel configuration controls the bottleneck layer, and the compression ratio controls the final latent-vector size. The encoder filter configurations considered in this study were $(2, 1)$, $(4, 2)$, $(8, 4)$, $(16, 8)$, and $(32, 16)$, where each pair represents the number of filters used in the first and second convolutional layers,

respectively. Bottleneck channel configurations of one and two channels were evaluated. Compression ratios 0.01, 0.02, 0.10, 0.20, 0.30, 0.40 and 0.50 were considered to control the size of the latent representation. The valid combinations of these design parameters resulted in a total of 70 baseline convolutional autoencoder models.

All baseline models were trained and evaluated using the same dataset, preprocessing procedure, training configuration, and evaluation metrics to ensure a fair comparison. Reconstruction quality was measured using NMSE, while downstream classifier accuracy was used to assess whether the reconstructed images retained task-relevant information. Model size and estimated energy consumption were also recorded to characterize the deployment cost of each architecture.

The design-space exploration was used to identify the relationships among encoder complexity, latent representation size, reconstruction quality, and deployment efficiency. The resulting baseline models formed the reference set for the subsequent evaluation of PTQ, QAT, magnitude-based pruning, and combined optimization strategies.

Design parameter	**Values considered**
Encoder filter configuration	$(2, 1)$, $(4, 2)$, $(8, 4)$, $(16, 8)$, $(32, 16)$
Bottleneck channels	1 and 2
Compression ratio	0.01, 0.10, 0.20, 0.30, 0.40, and 0.50
Total baseline models	70
Evaluation metrics	NMSE, classifier accuracy, model size, and estimated energy consumption

Table 1: Design parameters used to generate the baseline convolutional autoencoder models.

3.5 Model Optimization

The theoretical background of PTQ, QAT, and magnitude-based pruning was introduced in Section 2.3. This section describes how these optimization techniques were applied experimentally to the 70 baseline convolutional autoencoder models obtained from the design-space exploration.

The purpose of this stage was to evaluate whether model optimization could improve deployment efficiency while preserving acceptable reconstruction quality. Each optimized model was compared with its corresponding FP32 baseline model using the same evaluation metrics: NMSE, classifier accuracy, model size, and estimated total energy consumption.

Three individual optimization techniques were first evaluated: PTQ, QAT, and encoder weight pruning. In addition, combined optimization strategies were investigated to study whether applying quantization and pruning together could provide better energy vs quality trade-offs than using a single optimization technique alone.

The following subsections describe the implementation of each optimization method and the evaluation procedure used to compare the optimized models.

3.5.1 PTQ

PTQ was applied to the trained FP32 convolutional autoencoder models after training. For each baseline model, the trained TensorFlow model was converted into an INT8 TensorFlow Lite model using a representative calibration dataset. The calibration data were used to estimate the activation ranges required for integer quantization.

After conversion, each PTQ model was evaluated using the same test set and evaluation procedure as the corresponding FP32 baseline model. Reconstruction quality was measured using NMSE, while classifier accuracy was used to evaluate whether the reconstructed images preserved task-relevant information. Model size and estimated total energy consumption were also recorded to assess deployment efficiency.

This comparison was used to determine how much deployment efficiency could be improved by applying PTQ without retraining the model.

3.5.2 QAT

QAT was applied to reduce the loss in reconstruction quality caused by low-precision deployment. Unlike PTQ, where quantization is applied only after training, QAT includes simulated quantization effects during training. This allows the model to adapt to INT8 arithmetic before final conversion.

In this thesis, the baseline FP32 convolutional autoencoder models were retrained using QAT. After retraining, the QAT models were converted into INT8 TensorFlow Lite models for evaluation and hardware deployment consideration.

The QAT models were evaluated using the same metrics as the baseline and PTQ models: NMSE, classifier accuracy, model size, and estimated total energy consumption. The results were then compared with the FP32 and PTQ models to determine whether QAT provided a better trade-off between reconstruction quality and deployment efficiency.

3.5.3 Magnitude-Based Encoder Pruning

Magnitude-based pruning was applied to the encoder part of the convolutional autoencoder. The encoder was selected for pruning because it is the component deployed on the resource-constrained embedded device, while the decoder is executed on the receiver side.

A two-stage pruning procedure was used. In the first stage, sparsity was encouraged during training by adding an L_1 regularization term to the encoder weights. This encouraged less important weights to move closer to zero. In the second stage, a hard threshold was applied after training to remove weights with small magnitudes.

After pruning, the remaining model parameters were fine-tuned to recover reconstruction performance. The pruned models were then evaluated using NMSE, classifier accuracy, model size, and estimated total energy consumption.

The objective of this experiment was to determine whether encoder pruning could reduce computational complexity while maintaining acceptable reconstruction quality.

3.5.4 Combined Optimization Strategies

In addition to evaluating PTQ, QAT, and pruning individually, combined optimization strategies were also investigated. The purpose was to determine whether applying multiple optimization techniques together could provide better deployment efficiency than using a single technique alone.

The following optimization sequences were evaluated:

- **Pruning + PTQ:** encoder pruning was applied first, followed by PTQ.
- **Pruning + QAT:** encoder pruning was applied first, followed by QAT.
- **PTQ + Pruning:** PTQ was applied first, followed by pruning.
- **Pruning + Dropout:** pruning was combined with dropout-based fine-tuning.
- **Random pruning:** weights were removed randomly and used as a comparison method.

Each combined optimization strategy was evaluated using the same experimental setup as the baseline and individually optimized models. The results were compared using reconstruction quality, classifier accuracy, model size, and estimated total energy consumption.

The combined optimization experiments were included to identify whether a sequence of optimization techniques could provide improved energy vs quality trade-offs for TinyML deployment. The resulting optimized models were later included in the Pareto-front analysis to identify suitable candidates for hardware implementation.

3.6 Analytical Energy Model

To compare the deployment efficiency of different convolutional autoencoder models, an analytical energy model was developed to estimate the total energy consumption of the proposed TinyML image compression framework. The model considers the two dominant energy components of the system: the computation energy required to execute the encoder on the embedded device and the communication energy required to transmit the compressed latent representation.

Only the encoder is assumed to execute on the resource constrained IoT node, while the decoder is assumed to execute on the receiver or host computer. Consequently, the analytical model considers only the encoder computation energy and the energy associated with transferring the compressed latent payload. The decoder energy, wireless protocol overhead, and the energy consumed by an external radio transceiver are outside the scope of this model.

The total energy consumption is expressed as

$$E_{\text{total}} = E_{\text{comp}} + E_{\text{tx}}, \tag{1}$$

where E_{comp} denotes the encoder computation energy and E_{tx} denotes the transmission energy.

3.6.1 Computation Energy

The computation energy represents the energy consumed by the microcontroller during encoder inference. It depends on the computational complexity of the network, which is characterized by the number of multiply accumulate (MAC) operations executed by the encoder.

The computation energy is estimated as

$$E_{\text{comp}} = \mu_c V a C, \tag{2}$$

where

- μ_c is the hardware-dependent computation energy coefficient,
- V is the supply voltage,
- a is the number of processor cycles required per MAC operation,
- C is the total number of MAC operations executed by the encoder.

Models with more convolutional filters, larger bottleneck dimensions, or higher architectural complexity generally require a larger number of MAC operations and therefore consume more computation energy.

3.6.2 Transmission Energy

After image compression, the encoder produces a latent representation containing n latent variables. If each latent variable is represented using k bits, the total number of transmitted bits is

$$n_{\text{bits}} = nk. \tag{3}$$

Assuming that one bit is transmitted per clock cycle and the microcontroller operates at a clock frequency f, the transmission time is

$$t = \frac{n_{\text{bits}}}{f} = \frac{nk}{f}. \tag{4}$$

The active power consumption of the microcontroller is

$$P = IV, \tag{5}$$

where the active current is modeled as

$$I = \mu_t f, \tag{6}$$

with μ_t representing the hardware-dependent current coefficient. Based on the target embedded platform, the current coefficient is assumed to be $\mu_t = 6$ μA/MHz and the operating voltage is assumed to be $V = 3.3$ V.

Therefore, the transmission energy is

$$E_{\text{tx}} = Pt = (\mu_t f V) \left(\frac{nk}{f} \right), \tag{7}$$

which simplifies to

$$E_{\text{tx}} = \mu_t V k n. \tag{8}$$

The analytical model assumes that the antenna energy and communication protocol overhead are negligible. Consequently, the transmission energy is proportional to the latent vector dimension and the number of bits used to represent each latent value.

3.6.3 Total Energy Consumption

Combining the encoder computation energy and the latent-vector transmission energy gives the estimated total energy consumption of the proposed framework:

$$E_{\text{total}} = \mu_{\text{c}} V a C + \mu_{\text{t}} V k n. \tag{9}$$

In Equation 9, the first term represents the energy required to execute the encoder on the embedded device, while the second term represents the energy associated with transferring the compressed latent representation. The total energy therefore depends on both the encoder complexity and the size of the transmitted latent vector.

This equation estimates the deployment energy only. It does not directly include reconstruction quality or reconstruction error. Therefore, reconstruction quality is evaluated separately using NMSE, and the estimated energy values are later compared with NMSE through Pareto-front analysis. In this way, the energy model provides the deployment-cost estimate, while NMSE provides the reconstruction-quality measure.

With the compression-ratio definition used in this thesis, a smaller compression ratio produces a smaller latent vector and therefore reduces the number of transmitted values. This lowers transmission energy, but it can increase reconstruction error because

less information is preserved. Conversely, a larger compression ratio produces a larger latent vector, which can improve reconstruction quality but increases the communication payload and estimated total energy consumption.

The analytical energy model was applied consistently to all baseline and optimized convolutional autoencoder models to compare their deployment efficiency under the same assumptions.

3.7 Hardware Deployment

To validate the practical feasibility of the proposed TinyML image compression framework, a selected INT8 convolutional autoencoder encoder was deployed on an Ambiq Apollo3 Blue Plus microcontroller. The deployment focused exclusively on the encoder because image compression is performed on the resource-constrained embedded device, while image reconstruction is carried out by the decoder on the receiver side. This deployment strategy significantly reduces the computational and memory requirements of the embedded platform.

After training, the selected convolutional autoencoder model was converted from the TensorFlow format to the TFLite format. QAT was used to generate an INT8 model suitable for deployment on the target hardware. The TFLite model was subsequently converted into a C header file and embedded into the firmware. TensorFlow Lite for Microcontrollers (TFLM) was used as the inference engine, while the NeuralSPOT software stack and AmbiqSuite SDK provided the embedded software environment for executing the model on the Apollo3 Blue Plus evaluation board [4].

During initialization, the embedded application loads the neural network model, allocates the tensor arena required by TFLM, initializes the Micro Interpreter, and prepares the input and output tensors for inference. Once an input image is supplied, the encoder performs forward inference and produces a compressed latent representation. The latent vector can then be transmitted to an external receiver, where the decoder reconstructs the original image.

Figure 8 illustrates the workflow used to convert, integrate, and deploy the selected INT8 encoder on the Ambiq Apollo3 Blue Plus microcontroller.

The primary objective of the hardware deployment was to verify that the selected INT8 encoder could execute successfully within the memory limitations of the Apollo3 micro-

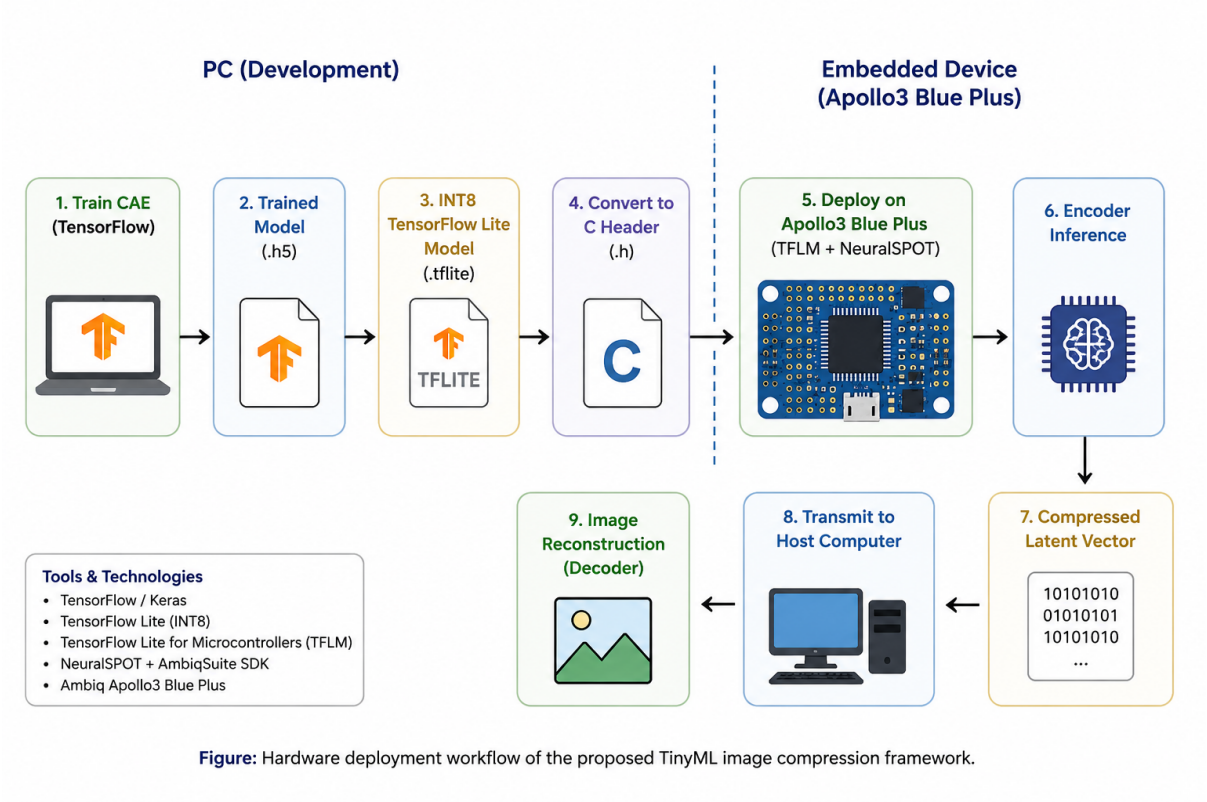

Figure 8: Hardware deployment workflow of the proposed TinyML image compression framework. The convolutional autoencoder is trained in TensorFlow and exported as an INT8 TensorFlow Lite model. The model is converted into a C header file and integrated into the embedded firmware using TFLM, NeuralSPOT, and the AmbiqSuite SDK. The encoder is executed on the Ambiq Apollo3 Blue Plus microcontroller to generate a compressed latent representation, which is transferred to the host computer for image reconstruction by the decoder. Illustration created by the author.

controller. Inference time was measured directly on the embedded platform using the built-in timing functionality available in the NeuralSPOT framework. The measured inference time was used to validate the practical deployment feasibility of the proposed TinyML framework. Hardware evaluation in this thesis is limited to successful model execution and inference time measurement, while energy consumption is estimated separately using the analytical energy model presented in Chapter 3.

3.8 Performance Evaluation Metrics

The proposed TinyML image compression framework is evaluated using multiple performance metrics that assess reconstruction quality, model effectiveness, deployment efficiency, and energy consumption. Since no single metric can fully characterize the performance of a compression model, several complementary metrics are considered throughout this thesis. These metrics are consistently used to compare the baseline convolutional au-

toencoder models and the optimized models obtained through quantization and pruning.

3.8.1 NMSE

NMSE is used to evaluate the reconstruction quality of the convolutional autoencoder. It measures the difference between the original image and the reconstructed image while normalizing the error with respect to the signal energy. Lower NMSE values indicate better reconstruction quality and more accurate preservation of image information. NMSE is widely used in image reconstruction and compression tasks because it enables fair comparison across different models and datasets [6, 19].

3.8.2 Classifier Accuracy

Although the autoencoder is optimized for image reconstruction rather than image classification, preserving semantic information is important for many embedded vision applications. Therefore, a pretrained image classifier is used to classify the reconstructed images, and the resulting classification accuracy is reported. Higher classifier accuracy indicates that the compressed latent representation preserves the important visual features required for downstream tasks.

3.8.3 Model Size

Model size represents the amount of memory required to store the trained neural network parameters. This metric is particularly important for TinyML applications because embedded microcontrollers provide limited flash memory. Model size is reported for both the baseline and optimized models to evaluate the effectiveness of quantization and pruning in reducing storage requirements [2, 11].

3.8.4 Inference Time

Inference time represents the execution time required by the embedded encoder to process a single input image. During hardware deployment, inference time is measured directly on the Ambiq Apollo3 Blue Plus microcontroller using the built-in timing functionality available in the NeuralSPOT framework. Faster inference indicates more efficient execution on the target embedded platform and is an important requirement for real-time TinyML applications [4].

3.8.5 Estimated Total Energy Consumption

The analytical energy model developed in Chapter 3 is used to estimate the total energy consumption of each convolutional autoencoder. The estimated total energy consists of the encoder computation energy and the transmission energy associated with the compressed latent representation. This metric enables a systematic comparison of different network architectures and optimization strategies under the same hardware assumptions. Lower estimated energy indicates a more deployment-efficient model for resource-constrained embedded systems.

The combination of these metrics provides a comprehensive evaluation of the proposed TinyML framework by considering reconstruction quality, preservation of semantic information, memory footprint, computational efficiency, and estimated energy consumption. These metrics form the basis for the experimental comparison presented in Chapter 4.

4 Results and Discussion

4.1 Baseline Model Performance

This section presents the experimental results of the FP32 baseline convolutional autoencoder models. These models were obtained from the design-space exploration described in Chapter 3 and are used as reference models for evaluating the effect of quantization, pruning, and combined optimization techniques.

The baseline models were evaluated using NMSE, downstream classifier accuracy, model size, and estimated total energy consumption. The purpose of this analysis is to identify how encoder complexity and latent-vector size affect reconstruction quality and deployment efficiency.

The results show a clear trade-off between reconstruction quality and energy consumption. Larger encoder architectures and larger latent representations generally reduce NMSE and improve classifier accuracy, but they also increase computation cost and transmission energy. In contrast, stronger compression reduces the communication payload but usually increases reconstruction error.

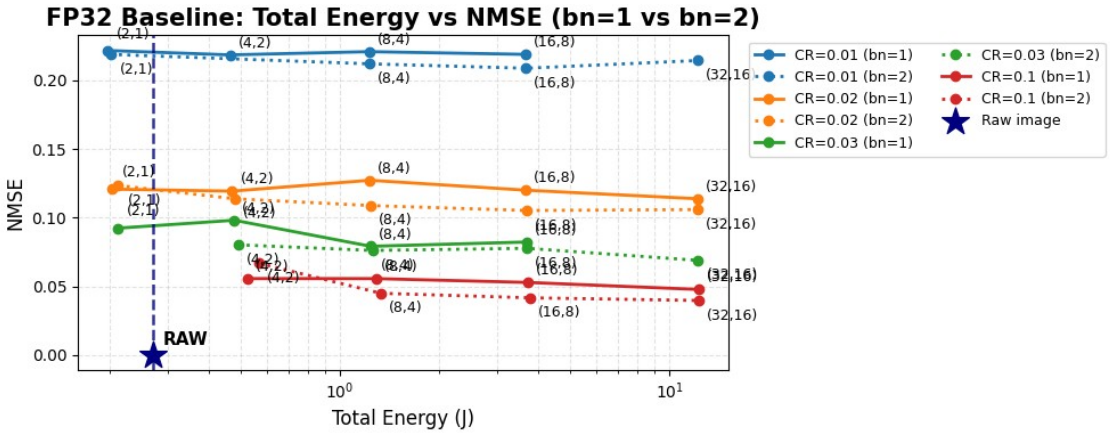

Figure 9: Total energy consumption versus NMSE for FP32 baseline autoencoder models. CR denotes the compression ratio, and bn denotes the number of bottleneck channels. The raw-image transmission case is shown as a reference point. Source: author's experiments.

In Figure 9, CR denotes the compression ratio and bn denotes the number of bottleneck channels used in the bottleneck layer. The compression ratio controls the size of the final latent vector. For an MNIST input image of size 28×28, the raw input contains 784 pixel values. Therefore, a smaller CR produces a smaller latent vector and reduces the transmitted payload, while a larger CR produces a larger latent vector and preserves

more image information. For example, CR values of 0.01, 0.02, 0.10, 0.20, and 0.30 correspond approximately to latent-vector sizes of 7, 78, 156, and 235 values, respectively. The parameter bn controls the bottleneck depth before the final latent representation is generated.

Table 2: Relationship between compression ratio and latent-vector size for MNIST images.

Compression ratio (CR)	**Approximate latent-vector size**
0.01	7
0.02	15
0.10	78
0.20	156
0.30	235
0.40	313
0.50	392

Thus, CR mainly controls the amount of information transmitted, while bn controls the internal bottleneck capacity of the encoder. This explains why models with larger CR values usually achieve lower NMSE but consume more transmission energy. However, a smaller latent vector preserves less image information and usually increases NMSE. Conversely, a larger compression ratio improves reconstruction quality but increases the communication payload and total energy consumption.

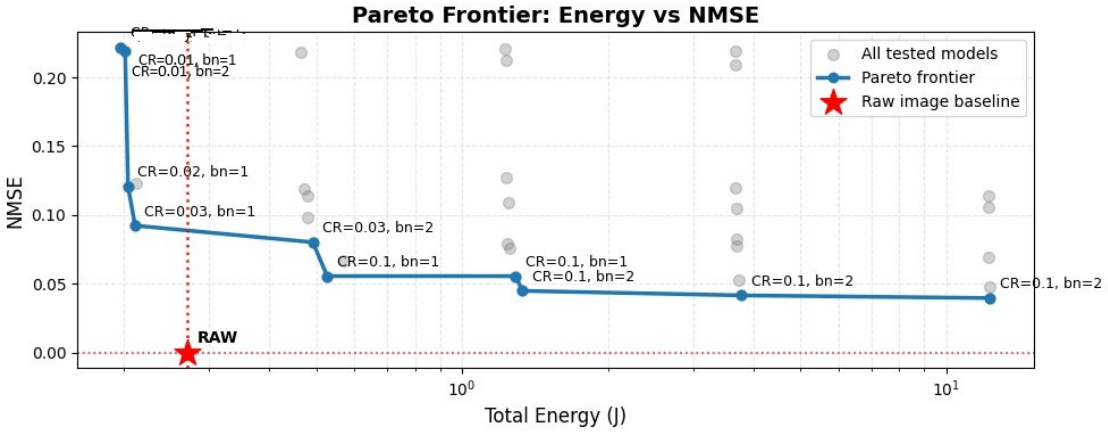

Figure 10: Pareto frontier of FP32 baseline models based on estimated total energy consumption and NMSE. Source: author's experiments.

Figure 10 presents the Pareto frontier of the FP32 baseline models. A model is considered Pareto-optimal if no other model achieves both lower estimated total energy consumption and lower NMSE at the same time. Therefore, the Pareto frontier identifies the most efficient baseline models for different energy budgets.

Classifier accuracy was also evaluated to determine whether the reconstructed images preserved sufficient information for downstream recognition tasks. This is important because a low reconstruction error does not always guarantee that the compressed image remains useful for machine-learning-based sensing applications.

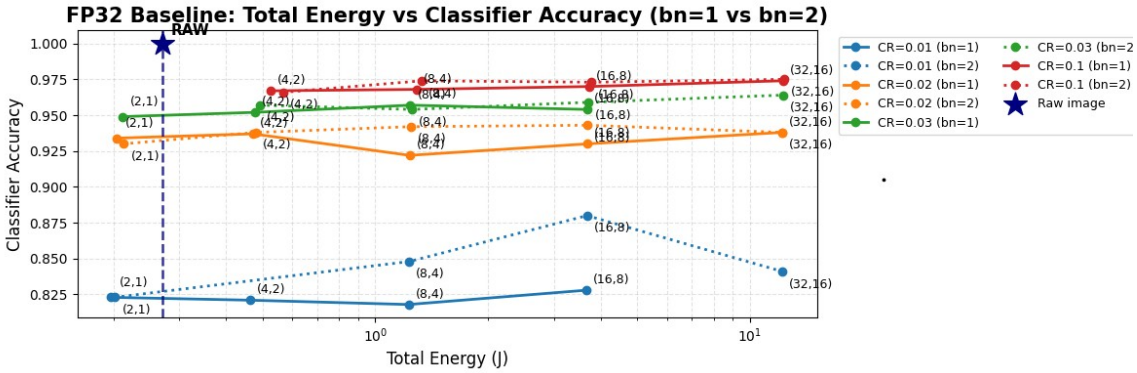

Figure 11: Total energy consumption versus classifier accuracy for FP32 baseline models. Source: author's experiments.

As shown in Figure 11, classifier accuracy generally improves as the latent representation becomes larger. This indicates that larger latent vectors preserve more discriminative features of the input digits. However, after a certain point, the improvement in classifier accuracy becomes small compared with the increase in estimated energy consumption. This confirms that the most useful operating points are not necessarily the largest models, but rather the models that provide a balanced trade-off between energy consumption and task performance.

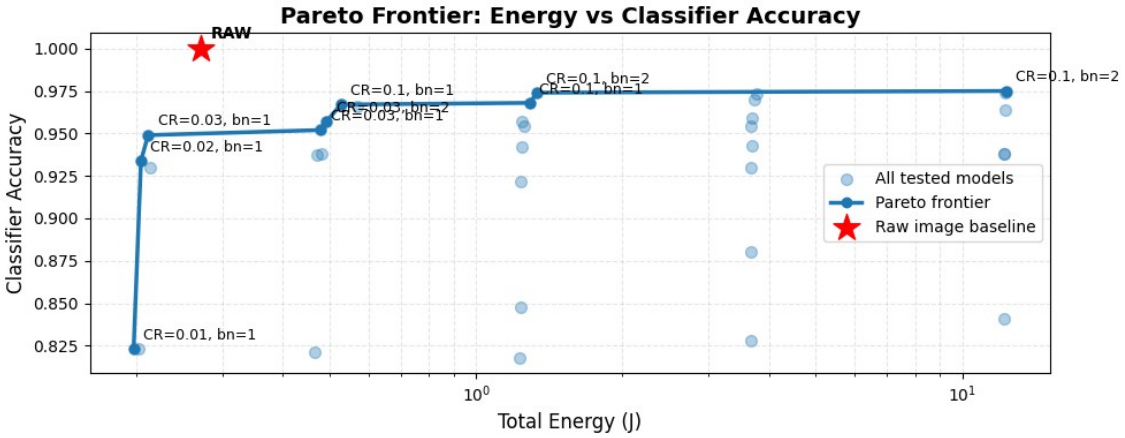

Figure 12: Pareto frontier of FP32 baseline models based on total energy and classifier accuracy. Source: author's experiments.

Figure 12 shows that many FP32 baseline models do not clearly outperform the raw-image baseline when both energy consumption and classifier accuracy are considered.

Although these models reduce the transmitted payload by sending a compressed latent representation, the additional encoder computation and reconstruction loss can reduce the overall benefit. This indicates that the baseline FP32 models alone are not always sufficient for efficient embedded deployment. Therefore, model optimization techniques such as quantization and pruning are required to improve the energy vs accuracy trade-off.

4.2 Effect of Quantization

This section evaluates the effect of applying PTQ and QAT to the baseline convolutional autoencoder models. The theoretical background of these methods was introduced in Section 2.3; therefore, this section focuses on their effect on reconstruction quality, classifier accuracy, model size, and estimated energy consumption.

The quantized models were compared with their corresponding FP32 baseline models. The objective was to determine whether INT8 deployment can reduce model size and improve hardware feasibility while preserving acceptable reconstruction quality.

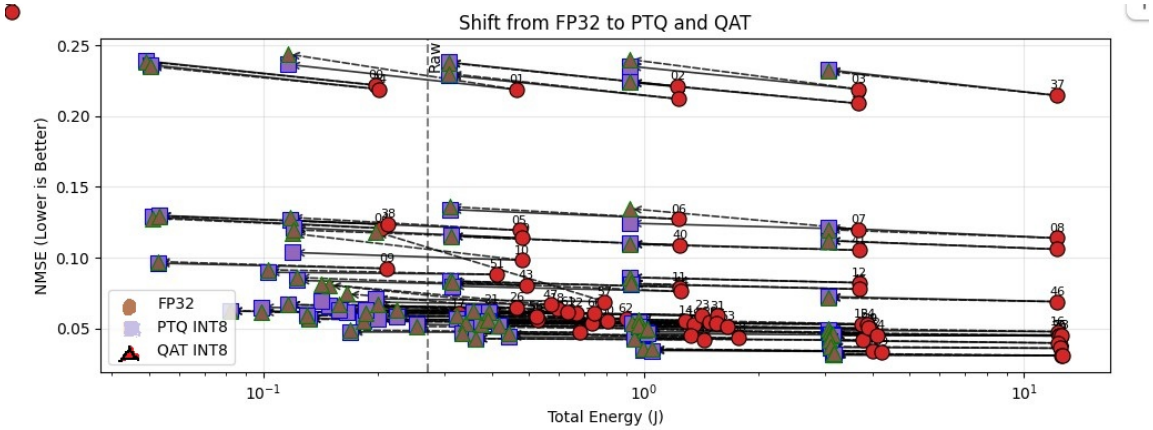

Figure 13: Shift from FP32 models to PTQ INT8 and QAT INT8 models in terms of estimated total energy consumption and NMSE. Source: author's experiments.

In Figure 13, each marker represents one evaluated model. The figure shows that the INT8 quantized models generally remain close to the FP32 baseline models in terms of NMSE, while improving deployment feasibility through reduced numerical precision. QAT models tend to preserve reconstruction quality better than PTQ models because quantization effects are included during training. The results show that both PTQ and QAT reduce deployment cost while maintaining acceptable reconstruction quality for many architectures. Most quantized models remain close to their corresponding FP32 models in terms of NMSE, indicating that the convolutional autoencoders are relatively robust to reduced numerical precision.

Compared with PTQ, QAT generally achieves slightly lower NMSE and better classifier accuracy because the model is exposed to quantization effects during training. PTQ remains useful when fast conversion is required, while QAT is more suitable when the final deployment model must preserve reconstruction quality as much as possible.

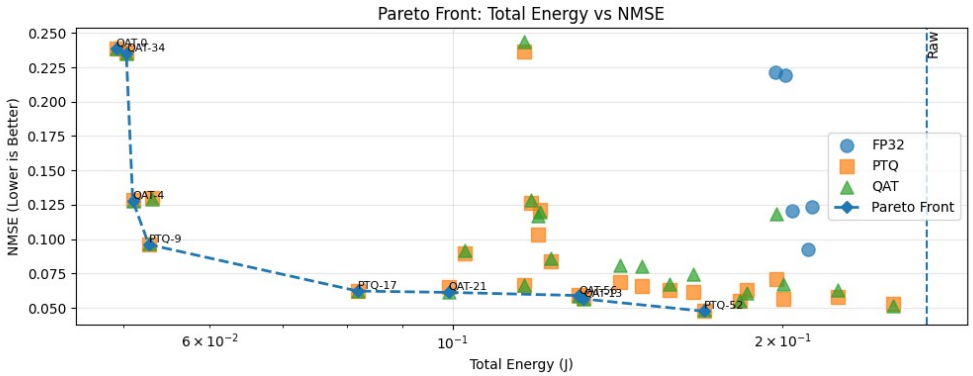

Figure 14: Pareto front comparison of FP32, PTQ INT8, and QAT INT8 models. Source: author's experiments.

Pareto-optimal models:

	type	model_no	enc_f1	enc_f2	bottleneck_c	cr	latent_c	macs_enc	E_mac_J_common	E_tx_J_common	E_total_J_common	nmse
0	QAT	0	2	1	1	0.01	7	18032	0.048686	0.000605	0.049291	0.238444
1	QAT	34	2	1	2	0.01	7	18424	0.049745	0.000605	0.050350	0.235123
2	QAT	4	2	1	1	0.02	15	18424	0.049745	0.001296	0.051041	0.127823
3	PTQ	9	2	1	1	0.03	23	18816	0.050803	0.001987	0.052790	0.096054
4	PTQ	17	2	1	1	0.20	156	25333	0.068399	0.013478	0.081878	0.062198
5	QAT	21	2	1	1	0.30	235	29204	0.078851	0.020304	0.099155	0.061221
6	QAT	56	2	1	2	0.30	235	40768	0.110074	0.020304	0.130378	0.058934
7	QAT	13	4	2	1	0.10	78	46256	0.124891	0.006739	0.131630	0.056476
8	PTQ	52	4	2	2	0.20	156	57820	0.156114	0.013478	0.169592	0.047509

Figure 15: Pareto-optimal FP32, PTQ INT8, and QAT INT8 models identified from the energy–NMSE analysis. Source: author's experiments.

Figure 14 shows that several quantized models achieve reconstruction quality comparable to FP32 models while reducing deployment cost. Among the two quantization methods, QAT provides a better trade-off between reconstruction quality and estimated total energy consumption. PTQ offers a simpler and faster optimization path, whereas QAT is more suitable for final deployment model selection.

Overall, quantization is an effective optimization technique for the proposed TinyML framework. It reduces numerical precision and memory footprint while maintaining acceptable reconstruction quality, making the optimized models more suitable for embedded deployment.

4.3 Effect of Two-Stage Encoder Weight Pruning

A two-stage encoder weight pruning method was applied to reduce the computational complexity of the autoencoder. Since the encoder is the component deployed on the resource-constrained IoT node, pruning was applied only to the encoder weights. The decoder was kept unchanged and executed on the receiver side.

In the first stage, soft sparsity was introduced during training by adding an L_1 regularization term to the reconstruction loss:

$$L = L_{\text{rec}} + \gamma \|W_{\text{enc}}\|_1, \tag{10}$$

where L_{rec} is the reconstruction loss, W_{enc} denotes the encoder weights, and γ controls the strength of sparsity regularization. The regularization term encourages less important weights to move closer to zero during training.

In the second stage, hard-threshold pruning was applied after training. For a pruning threshold τ, each encoder weight was updated as follows:

$$W_{ij}^{\text{pruned}} = \begin{cases} W_{ij}, & \text{if } |W_{ij}| \geq \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Different threshold values were evaluated to study the trade-off between sparsity, reconstruction quality, classifier accuracy, and estimated total energy consumption. The resulting pruned models were then analyzed using Pareto-front analysis to identify energy-efficient operating points.

Figure 16 shows the energy–NMSE trade-off for different pruning levels. The results indicate that moderate pruning can reduce computational cost while maintaining reconstruction quality close to the baseline models. However, aggressive pruning can remove important weights and significantly degrade reconstruction quality.

The results also show that not all models respond equally to pruning. Larger models contain more redundant parameters and can often tolerate pruning better than very small models. In contrast, highly compressed models already have limited capacity, so removing additional weights can cause a larger degradation in performance.

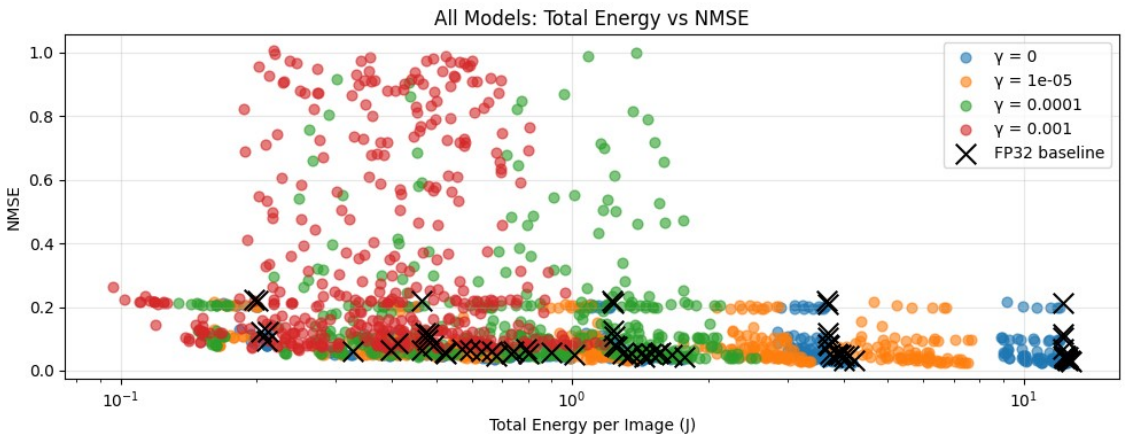

Figure 16: Effect of pruning on estimated total energy consumption and NMSE for all evaluated models. Source: author's experiments.

4.4 Pareto Front Analysis

To identify deployment-efficient operating points, Pareto-front analysis was performed using estimated total energy consumption and NMSE. Each point on the Pareto front represents a non-dominated solution, meaning that no other model simultaneously achieves lower energy consumption and lower reconstruction error.

The analysis illustrates the trade-off between reconstruction quality and deployment efficiency. Models located toward the lower-left region of the plot achieve both low estimated energy consumption and low NMSE, making them attractive candidates for resource-constrained embedded deployment.

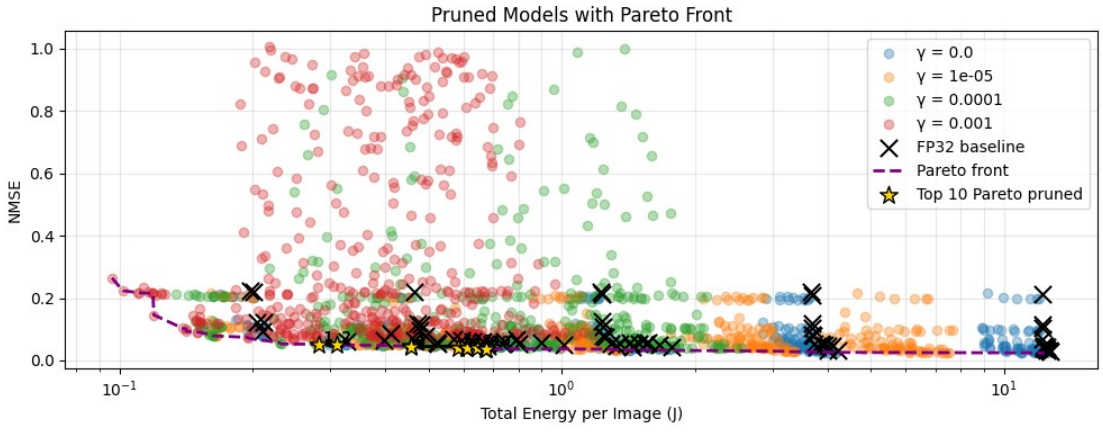

Figure 17: Pareto front of pruned models based on estimated total energy consumption and NMSE. Source: author's experiments.

Figure 17 presents the Pareto front of the pruned models. Several pruned models lie close to the baseline Pareto frontier, indicating that pruning can provide useful energy-

aware optimization when applied with suitable pruning strength. These models reduce computation while maintaining sufficient reconstruction quality.

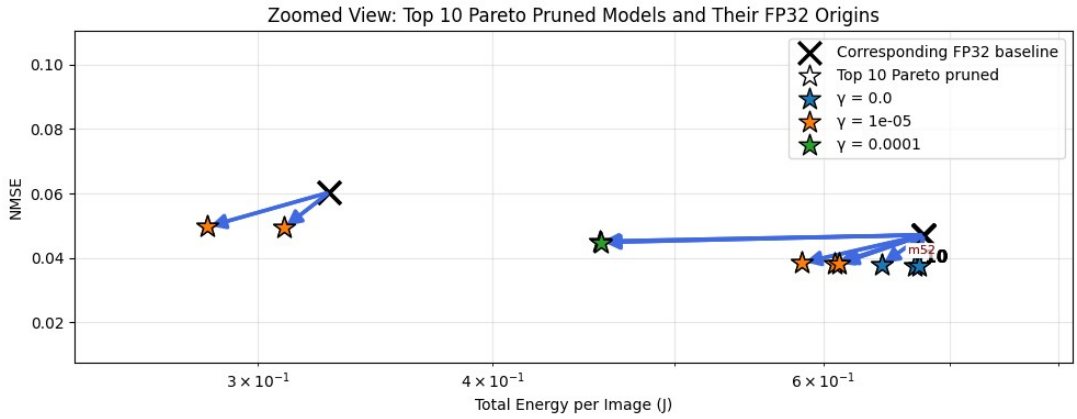

Figure 18: Pareto frontier of pruned models and their corresponding FP32 baseline models. The black cross markers indicate the original unpruned FP32 baseline models from which the pruned variants were generated. The star markers indicate pruned candidate models. Source: author's experiments.

In Figure 18, the corresponding FP32 baseline models are the original unpruned models used as reference points before pruning. In this experiment, pruning was applied to selected FP32 baseline architectures, including model indices 17 and 52. The black cross markers show these original FP32 models, while the star markers show the pruned variants obtained after applying different pruning settings. This comparison is used to evaluate whether pruning can reduce estimated energy consumption while maintaining acceptable reconstruction quality.

4.5 Evaluation of Combined Optimization Techniques on Selected FP32 Models

This section evaluates the effect of combined optimization techniques on selected FP32 baseline models. The individual optimization methods, including PTQ, QAT, and pruning, were introduced in Section 2.3 and their experimental implementation was described in Section 3.5. Therefore, this section focuses only on the comparative results.

The purpose of this experiment was to determine whether applying multiple optimization techniques together could improve the energy vs quality trade-off compared with using a single optimization method alone. Selected FP32 baseline models with different compression ratios and bottleneck sizes were used as reference models.

The following optimization cases were compared:

- Baseline FP32 model,
- Threshold-only pruning,
- Gamma-threshold pruning,
- Random pruning,
- PTQ,
- QAT,
- Pruning followed by dropout-based fine-tuning,
- Pruning followed by PTQ,
- Pruning followed by QAT,
- PTQ followed by pruning.

The notation `enc16_8_bn1_cr0.02_lat15` indicates an encoder with 16 filters in the first convolutional layer, 8 filters in the second convolutional layer, one bottleneck channel, a compression ratio of 0.02, and a latent vector size of 15.

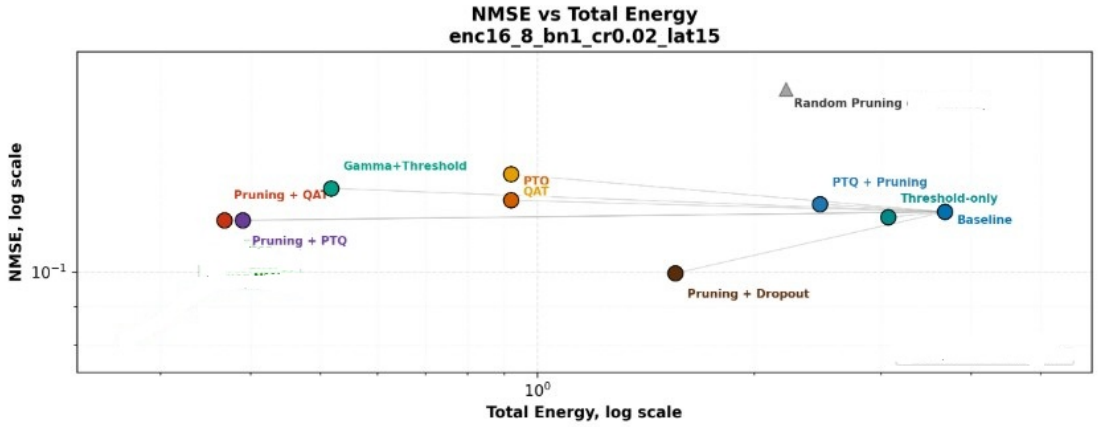

Figure 19: Total energy versus NMSE comparison of different optimization techniques applied to the selected FP32 baseline model `enc16_8_bn1_cr0.02_lat15`. The notation enc16_8_bn1_cr0.02_lat15 indicates an encoder with 16 filters in the first convolutional layer, 8 filters in the second convolutional layer, one bottleneck channel, a compression ratio of 0.02, and a latent vector size of 15.Source: author's experiments.

Figure 19 shows the effect of applying different optimization techniques to one selected FP32 baseline model. Points located further to the left indicate lower estimated energy consumption, while points located lower on the vertical axis indicate lower NMSE and therefore better reconstruction quality.

The results show that not all optimization techniques improve the energy–quality trade-off equally. Some methods reduce energy at the cost of higher reconstruction error, whereas combined approaches such as pruning followed by quantization provide a more useful balance between energy reduction and reconstruction quality. This is relevant for TinyML deployment because both computation cost and memory usage must be reduced while maintaining acceptable reconstruction quality.

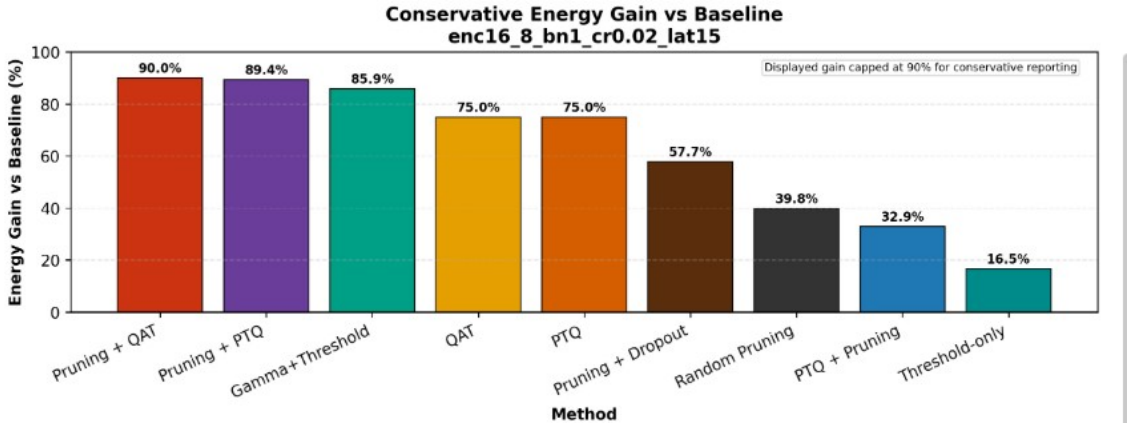

Figure 20: Estimated energy reduction of each optimization method compared with the selected FP32 baseline model. The notation enc16_8_bn1_cr0.02_lat15 indicates an encoder with 16 filters in the first convolutional layer, 8 filters in the second convolutional layer, one bottleneck channel, a compression ratio of 0.02, and a latent vector size of 15.Source: author's experiments.

Figure 20 compares the estimated energy reduction achieved by each optimization method relative to the FP32 baseline. Combined techniques such as pruning + QAT, pruning + PTQ, and PTQ + pruning achieve large energy reductions while maintaining acceptable reconstruction quality. Random pruning may also reduce energy, but it can increase NMSE because important weights may be removed without considering their contribution to reconstruction. Therefore, magnitude-based pruning is more suitable than random pruning for this application.

Figure 21 shows the same type of combined optimization analysis for a second selected model. The results again indicate that combined techniques such as pruning + QAT and pruning + PTQ can reduce estimated energy consumption while maintaining acceptable reconstruction quality. In contrast, random pruning reduces energy less reliably and can degrade reconstruction quality.

Figure 22 shows the energy–NMSE gain curve for the selected model. The knee region of the curve is particularly important because it represents a practical trade-off between energy saving and reconstruction quality. Operating near this region allows energy con-

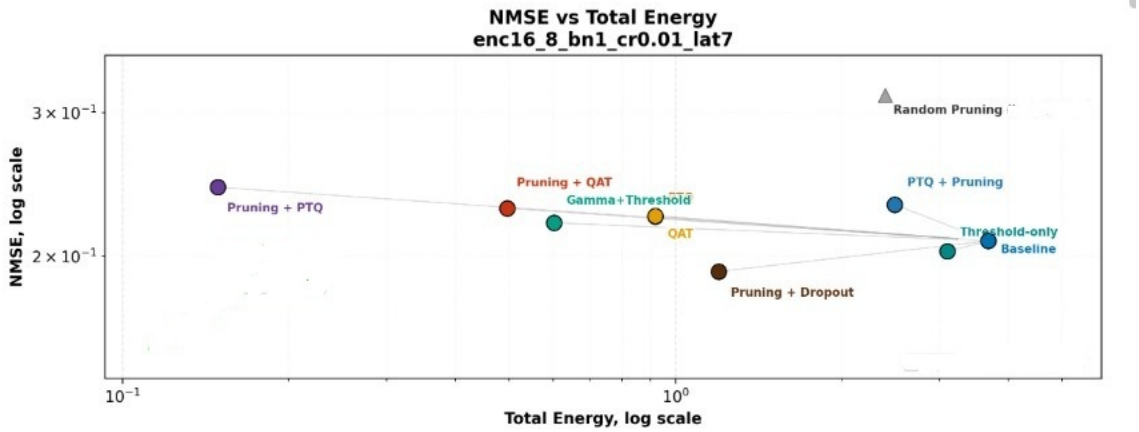

Figure 21: Total energy versus NMSE comparison for a second selected FP32 baseline model with a different compression ratio. The notation `enc16_8_bn1_cr0.01_lat7` indicates an encoder with 16 filters in the first convolutional layer, 8 filters in the second convolutional layer, one bottleneck channel, a compression ratio of 0.01, and a latent vector size of 7. Source: author's experiments.

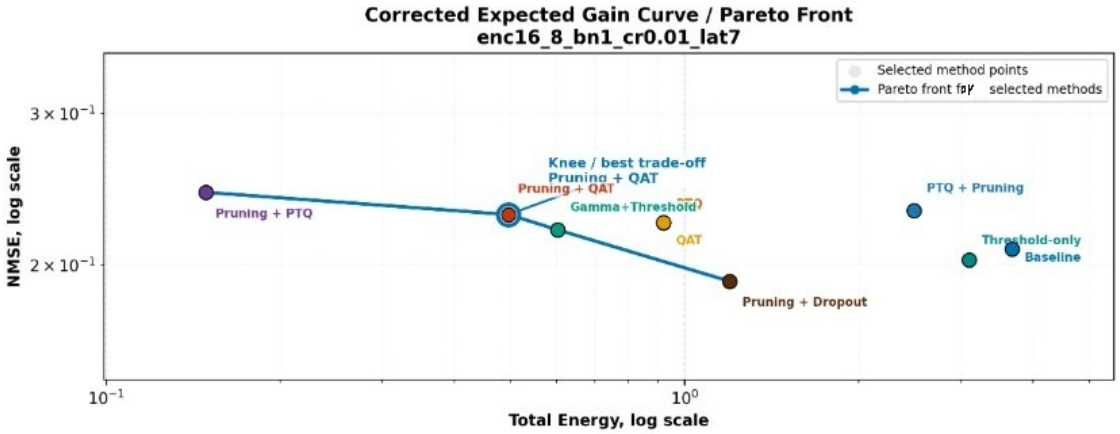

Figure 22: Total energy versus NMSE gain curve and Pareto front for the selected optimized model. The notation `enc16_8_bn1_cr0.01_lat7` indicates an encoder with 16 filters in the first convolutional layer, 8 filters in the second convolutional layer, one bottleneck channel, a compression ratio of 0.01, and a latent vector size of 7.Source: author's experiments.

sumption to be reduced without causing excessive degradation in image reconstruction.

Overall, the combined optimization results show that practical TinyML deployment requires more than one optimization criterion. FP32 baseline models and pruned models provide useful reference points, but INT8-optimized models are more suitable for embedded deployment because they reduce memory requirements and improve hardware feasibility.

4.6 Energy Consumption Analysis

The analytical energy model used in this thesis was introduced in Section 3.6. In this study, the analytical energy model assumes a hardware-dependent current coefficient of $\mu_t = 6$ μA/MHz and an operating voltage of $V = 3.3$ V.

Figure 14 illustrates the relationship between total estimated energy consumption and reconstruction quality. Models with lower reconstruction error generally require larger network architectures, increasing computation energy. Conversely, highly compressed models reduce transmission energy but may increase reconstruction error because less information is preserved in the latent representation.

The results demonstrate that efficient image compression is achieved when the reduction in transmission energy outweighs the additional computation required for encoder inference. Consequently, the most suitable deployment models are not necessarily those with the lowest NMSE, but those that provide a balanced trade-off between reconstruction quality and total energy consumption.

The impact of different optimization techniques is also evident from the experimental results. Quantization primarily improves deployment efficiency by reducing numerical precision and model memory requirements, whereas pruning reduces computational complexity by removing less significant network parameters. The combination of these optimization techniques enables the development of lightweight convolutional autoencoder models that achieve low estimated energy consumption while maintaining acceptable reconstruction quality.

4.7 Hardware Deployment Results

To validate the practical feasibility of the proposed TinyML image compression framework, the selected INT8 convolutional autoencoder encoder was deployed on an Ambiq

Apollo3 Blue Plus evaluation board using TFLM and the NeuralSPOT software framework. The hardware deployment was performed to verify that the selected model could execute successfully on a resource-constrained microcontroller while satisfying the memory and inference-time requirements of the target platform.

Based on the Pareto-front analysis, the QAT model with encoder filters (2,1), bottleneck dimension 2, compression ratio 0.30, and a latent vector size of 235 (latent c = 235) was selected for hardware deployment on the Ambiq Apollo3 microcontroller. This model was chosen because it provided a favorable trade-off between reconstruction quality and total energy consumption.

Figure 23 illustrates the experimental hardware setup used in this work.

4.7.1 Experimental Setup

The experimental setup consisted of an Ambiq Apollo3 Blue Plus evaluation board connected to a host computer through a SEGGER J-Link debugger. The trained INT8 encoder model was embedded into the firmware using TFLM and executed on the Apollo3 microcontroller. During inference, the encoder generated a compressed latent representation, which was transferred to the host computer for image reconstruction using the decoder.

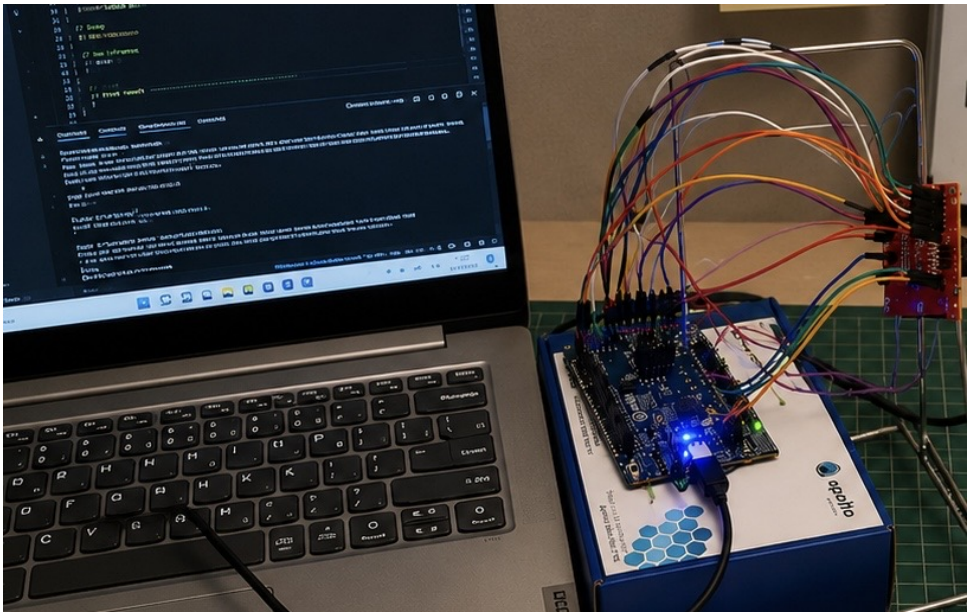

Figure 23: Hardware Deployment Set Up ((From Author's experiments)

Figure 23 shows the experimental hardware deployment setup. The setup consists of an Ambiq Apollo3 Blue Plus EVB connected to a host computer through a J-Link debugger.

The encoder model is executed on the microcontroller, while the decoder runs on the host computer. The host computer also receives the compressed latent vector through serial communication and reconstructs the transmitted image.

4.7.2 Encoder Inference Time

The inference performance of the selected INT8 encoder was evaluated on the Ambiq Apollo3 Blue Plus microcontroller. During deployment, only the encoder was executed on the embedded device, while the decoder was executed on the host computer.

Inference time was measured using the Apollo3 microcontroller's built-in microsecond timer. The timer value was recorded immediately before and after the TensorFlow Lite Micro `Invoke()` function, and the elapsed execution time was calculated from the difference between the two measurements.

The average encoder inference time was approximately 11.3 ms. This result demonstrates that the optimized INT8 convolutional autoencoder can execute efficiently on a resource-constrained microcontroller while satisfying the latency requirements of embedded TinyML applications.

4.7.3 Raw Image versus Compressed Transmission

The proposed framework transmits the compressed latent representation instead of the original image. Consequently, the communication payload is significantly reduced.

For the MNIST dataset, each input image contains 28×28 pixels, corresponding to 784 bytes of raw image data. After encoding, the latent representation contains only 235 bytes. This reduction in transmitted data decreases both communication time and transmission energy.

Assuming a packet size of 64 bytes and a transmission time of 6.4 ms per packet, transmitting the original image requires approximately 627.2 ms. In comparison, transmitting the compressed latent representation requires approximately 192.0 ms. Including the measured encoder inference time of 11.3 ms, the complete compression-and-transmission pipeline requires approximately 203.3 ms.

Compared with direct transmission of the raw image, the proposed framework reduces the overall transmission time by approximately 68.2%.

4.7.4 Transmission Energy Reduction

Reducing the communication payload also reduces the total energy required for data transmission. Assuming an average microcontroller power consumption of 6.85 mW, transmitting the original MNIST image requires approximately 4296.3 μJ. In comparison, transmitting the compressed latent representation requires approximately 1365.2 μJ.

The proposed framework therefore achieves an estimated energy saving of approximately 2931.1 μJ, corresponding to a reduction of approximately 68.2% compared with transmitting the original image.

Parameter	**Raw Transmission**	**Compressed Transmission**
Image size	784 bytes	235 bytes
Number of packets (64-byte packets)	98	30
Transmission time	627.2 ms	192.0 ms
Encoder inference time	–	11.3 ms
Total execution time	627.2 ms	203.3 ms
MCU power	6.85 mW	6.85 mW
Energy consumption	4296.3 μJ	1365.2 μJ
Energy saving	–	2931.1 μJ
Relative energy saving	–	68.2%

Table 3: Comparison of raw image transmission and the proposed compressed transmission pipeline.

These results demonstrate that executing a lightweight convolutional encoder on the embedded device and transmitting only the compressed latent representation can substantially reduce communication time and energy consumption. The proposed framework therefore represents a practical solution for energy-constrained embedded TinyML applications and provides a foundation for future deployment on energy-harvesting IoT platforms.

Table3 summarizes the measured deployment results. Although the proposed framework introduces an encoder inference time of 11.3 ms, transmitting the compressed latent vector instead of the original image reduces both communication time and total energy consumption. The complete compressed pipeline requires only 199.3 ms and 1365.2 μJ, compared with 627.2 ms and 4296.3 μJ for raw image transmission, corresponding to an energy reduction of approximately 68.2%.

4.8 Discussion

This study investigated the deployment of lightweight convolutional autoencoders for image compression on resource-constrained embedded platforms. The experimental results demonstrate that the proposed framework successfully balances reconstruction quality, deployment efficiency, and estimated energy consumption through systematic model optimization and hardware validation.

The design-space exploration revealed that the architecture of the convolutional autoencoder has a significant influence on reconstruction quality and deployment cost. Increasing the number of convolutional filters and bottleneck dimensions generally improved reconstruction performance by reducing the NMSE. However, these improvements were accompanied by increased computational complexity, larger model sizes, and higher estimated energy consumption. Consequently, selecting an appropriate architecture requires balancing reconstruction quality against deployment efficiency.

The evaluation of model optimization techniques showed that PTQ, QAT, and magnitude based pruning each contributed differently to deployment efficiency. PTQ substantially reduced model size with only a small reduction in reconstruction quality. QAT generally achieved better reconstruction quality than PTQ while maintaining the benefits of INT8 deployment. Magnitude based pruning reduced computational complexity by removing less significant network parameters, enabling lower computation energy for appropriately selected pruning levels.

The combined optimization strategies demonstrated that applying quantization together with pruning can further improve deployment efficiency. In particular, the QAT based optimized models provided a favorable balance between reconstruction quality and estimated energy consumption. These models were therefore selected as suitable candidates for hardware deployment.

The analytical energy evaluation highlighted the importance of jointly considering computation energy and communication energy when designing TinyML image compression systems. While more complex encoder architectures require additional computation, transmitting a compact latent representation significantly reduces the communication payload. The results indicate that the reduction in transmission energy outweighs the additional computation required by the encoder for the selected deployment models.

Hardware deployment on the Ambiq Apollo3 Blue Plus microcontroller demonstrated

the practical feasibility of the proposed framework. The selected INT8 encoder executed successfully within the resource limitations of the embedded platform, achieving an average inference time of approximately 11.3 ms. Furthermore, transmitting the compressed latent representation instead of the original image substantially reduced both communication time and estimated transmission energy, demonstrating the suitability of the proposed framework for energy-constrained embedded applications.

Although the experimental evaluation is limited to the MNIST dataset and a single embedded platform, the proposed methodology provides a systematic framework for evaluating lightweight convolutional autoencoders under resource constraints. Future work should extend the framework to more complex image datasets, wireless communication experiments, real-time camera acquisition, and energy-harvesting embedded systems to further validate its applicability in practical IoT deployments.

5 Conclusion and Future Work

5.1 Conclusion

This thesis presented an energy-aware TinyML framework for learned image compression on resource-constrained embedded devices. The work investigated lightweight convolutional autoencoders and model optimization techniques to reduce communication payload while maintaining acceptable reconstruction quality.

A design-space exploration of 70 convolutional autoencoder models showed that encoder architecture, bottleneck size, and latent-vector size strongly affect the trade-off between reconstruction quality, classifier accuracy, model size, and estimated energy consumption. Smaller latent representations reduced transmission cost, while larger representations generally improved reconstruction quality.

The evaluated optimization techniques showed that quantization and pruning can improve deployment efficiency. Among the tested approaches, QAT provided a suitable balance between reconstruction quality and INT8 deployment feasibility.

A selected INT8 encoder model was successfully deployed on the Ambiq Apollo3 Blue Plus microcontroller using TFLM and the neuralSPOT framework. The measured encoder inference time was approximately 11.3 ms. The analytical comparison also showed that transmitting the compressed latent representation can reduce estimated energy consumption compared with raw-image transmission.

Overall, the results demonstrate that learned image compression combined with TinyML optimization can reduce communication payload and improve deployment efficiency for resource-constrained IoT devices.

5.2 Future Work

Although the proposed framework demonstrates promising results for deployment-efficient image compression on resource-constrained embedded systems, several directions remain for future work.

First, the experimental evaluation in this thesis is limited to the MNIST dataset. Future work should evaluate the proposed framework using real sensor images to test its generalization capability under more realistic visual sensing conditions.

Second, the analytical energy model used in this thesis is based on simplified assumptions.

These assumptions include a fixed packet payload size, fixed packet transmission time, constant MCU active power, negligible communication protocol overhead, no packet loss or retransmission, and no direct measurement of radio transceiver energy. In this thesis, a fixed packet payload size means that each transmitted packet is assumed to carry the same amount of data, for example 64 bits per packet. Therefore, the number of packets is estimated by dividing the total number of transmitted bits by this fixed packet size. In a real wireless system, however, packet size, protocol overhead, retransmissions, and radio power consumption may vary depending on the communication technology and channel conditions. Future studies should validate these analytical estimates using practical wireless communication technologies such as Bluetooth Low Energy (BLE), LoRa, Wi-Fi, or other low-power IoT communication protocols.

Third, the current hardware implementation uses preloaded MNIST images as encoder inputs. Integrating a real-time image sensor would enable end-to-end evaluation of image acquisition, compression, transmission, and reconstruction on an embedded platform.

Finally, the proposed framework could be extended by incorporating adaptive model selection, dynamic compression strategies, and energy-aware scheduling. Evaluating the framework on battery-powered or energy-harvesting embedded platforms would further demonstrate its suitability for long-term autonomous TinyML systems.

6 References

[1] Johannes Ballé et al. "Variational Image Compression with a Scale Hyperprior". In: *International Conference on Learning Representations*. 2018.

[2] Colby Banbury et al. "Benchmarking TinyML Systems: Challenges and Direction". In: *arXiv preprint arXiv:2103.14370* (2021).

[3] Davis Blalock et al. "What is the State of Neural Network Pruning?" In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 129–146.

[4] Robert David, Jared Duke, Advait Jain, et al. "TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems". In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 800–811.

[5] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[7] Jayavardhana Gubbi et al. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660.

[8] Song Han et al. "Learning Both Weights and Connections for Efficient Neural Networks". In: *Advances in Neural Information Processing Systems* (2015).

[9] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507.

[10] Mark Horowitz. "1.1 Computing's Energy Problem (and What We Can Do About It)". In: *IEEE International Solid-State Circuits Conference (ISSCC)*. 2014, pp. 10–14.

[11] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *CVPR*. 2018.

[12] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[13] Y. Lin et al. "TinyML: Current Progress and Future Opportunities". In: *Sensors* 23.4 (2023).

[14] R. T. Marler and J. S. Arora. "Survey of Multi-Objective Optimization Methods for Engineering". In: *Structural and Multidisciplinary Optimization* 26.6 (2004), pp. 369–395.

[15] Jonathan Masci et al. “Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction”. In: *International Conference on Artificial Neural Networks*. 2011, pp. 52–59.

[16] Markus Nagel et al. “A White Paper on Neural Network Quantization”. In: *arXiv preprint arXiv:2106.08295* (2021).

[17] Ramon Sanchez-Iborra and Antonio Skarmeta. “TinyML for the Internet of Things: A Survey”. In: *Future Internet* 12.9 (2020), p. 159.

[18] Weisong Shi et al. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646.

[19] Lucas Theis et al. “Lossy Image Compression with Compressive Autoencoders”. In: *International Conference on Learning Representations*. 2017.

[20] Pete Warden and Daniel Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O’Reilly Media, 2019.

[21] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. “Wireless sensor network survey”. In: *Computer Networks* 52.12 (2008), pp. 2292–2330.

[22] Zhi Zhou et al. “Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing”. In: *Proceedings of the IEEE* 107.8 (2019), pp. 1738–1762.