Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

Efficient Computer Vision for Edge AI Applies Quantization Strategies that Optimize Memory, Latency, and Accuracy while Preserving Defect Sensitivity in Industrial Anomaly Detection under Limited Power Budgets

Mr. Muthumanickavel S[1], Dr. M. Sumathi[2], Dr. R. Sankarasubramanian[3]

[1]Ph.D Research Scholar, Dept. of Computer Science, Karuppannan Mariappan College, Muthur-638 105; muthumanickavel@gmail.com

[2]HoD in Information Technology,And AI &DS, Karuppannan Mariappan College, Muthur-638 105; sumathiabiaswin@gmail.com

[3]Principal, Erode Arts and Science College, Erode-638 009, Tamilnadu

Abstract: Limitations on industrial anomaly detection in computer vision is moving to edge devices, where memory, compute throughput, latency, and power budgets often make full-precision deep models impractical. This paper reviews how edge constraints shape deployment choices and motivates model-compression pipelines that preserve defect sensitivity while enabling real-time inference. It surveys neural-network quantization as a core strategy for reducing model size and accelerating inference, covering post-training quantization and quantization-aware training, along with uniform and mixed-precision approaches and recent methods designed to minimize accuracy loss at low bit-widths. It then summarizes industrial anomaly-detection paradigms (unsupervised and supervised), common datasets and evaluation practices, and representative methods that rely on feature backbones and efficient scoring/localization. Finally, it connects algorithmic choices to practical deployment workflows on embedded platforms, highlighting toolchains and runtimes (e.g., INT8 execution paths) that translate quantized models into measurable gains in throughput and energy efficiency. A defect-sensitive 8-bit quantization perspective is discussed to illustrate how task-aware compression can retain anomaly-detection performance while meeting strict on-device constraints.

Keywords: Edge AI; computer vision; industrial anomaly detection; defect detection; edge computing; neural network quantization; INT8 inference; post-training quantization (PTQ); quantization-aware training (QAT); mixed-precision quantization; model compression; resource-constrained devices

I. INTRODUCTION

Deploying deep learning models for industrial anomaly detection on resource-constrained edge devices is challenging due to the substantial memory, compute, and energy requirements of state-of-the-art vision models. Industrial anomaly detection (such as defect detection in manufacturing) often relies on high-capacity convolutional neural networks (CNNs) or vision transformers to achieve high accuracy, but these models are typically too heavy for real-time inference on embedded devices. Edge computing - performing computation on devices at or near the data source - promises low-latency, private, real-time analytics, yet is limited by device constraints (CPU speed, memory, power)[1][2]. For example, edge devices like Raspberry Pi, Orange Pi, or NVIDIA Jetson have limited memory and power budgets, making it infeasible to deploy large 32-bit floating point models without optimizations. Recent surveys on edge computing note that cloud-only processing suffers from bandwidth overload and high latency, and moving computation to the edge mitigates these issues[1][3]. However,

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

most IoT end-nodes are energy-constrained, so any on-device algorithm must be efficient[2]. This paper reviews techniques to compress and accelerate deep networks - with an emphasis on quantization - to enable real-time anomaly detection on edge hardware. We focus on three key aspects:
(1) Neural network quantization techniques for efficient inference,
(2) Edge computing constraints and their impact on deployment, and
(3) Industrial anomaly detection methods and how they can be adapted to edge devices.
We also discuss how these aspects come together in the context of EdgeSight, a defect-sensitive 8-bit quantization framework enabling high-performance anomaly detection on devices like Orange Pi and NVIDIA Jetson at 30 FPS with significantly reduced memory and power usage (as described in the EdgeSight abstract). In the following, we survey prior work in network quantization and edge deployment, outline the particular needs of industrial anomaly detection, and highlight recent advances that make it possible to preserve accuracy while compressing models for edge inference.

The remainder of this paper is organized as follows: we first outline the constraints and deployment challenges of running computer-vision anomaly detection on edge hardware. We then review neural-network quantization approaches (including PTQ (Post Training Quantization), QAT (Quantization Aware Training), and mixed-precision strategies) and summarize their trade-offs for accuracy, latency, and memory. Next, we survey industrial anomaly-detection methods and their adaptation to edge settings, and we follow with an implementation-focused discussion of practical deployment workflows on representative platforms. We conclude by summarizing key takeaways and highlighting open directions for defect-sensitive, energy-aware edge deployment.

II. EDGE COMPUTING CONSTRAINTS AND DEPLOYMENT CHALLENGES

Edge computing refers to processing data on or near the devices (sensors, cameras, gateways) that generate the data, rather than in a distant cloud server [4]. The motivation for edge computing in industrial settings includes reducing communication bandwidth, improving response times, and preserving data privacy [5][6]. Latency is a critical concern: many industrial inspection systems require real-time or near-real-time response (e.g. analyzing video frames on an assembly line at 30+ FPS). Offloading inference to the cloud can introduce unacceptable latency and reliability issues (e.g. due to network delays) [7]. By running inference on the edge, latency can be dramatically reduced and made more predictable [8]. Additionally, transmitting high-resolution images or video to the cloud for analysis would consume enormous bandwidth; edge processing alleviates bandwidth bottlenecks by only sending high-level results or alerts upstream [9][10].

Despite these advantages, edge devices have limited resources. Embedded CPUs and GPUs have far fewer cores and lower clock speeds than servers, and often no active cooling (which limits sustained throughput). Memory (both RAM and storage) is constrained, making it difficult to load large models. Importantly, energy is scarce on many edge devices (battery-powered or power-budgeted); as Shi et al [27]. note, “Battery is the most precious resource for things at the edge”. Thus, algorithms must be optimized for low power consumption. In practice, this means reducing the number of computations (operations) and using lower-power arithmetic where possible. For deep neural networks, 32-bit floating point operations are costly on embedded hardware, whereas 8-bit integer operations can often be executed by specialized low-power DSP units or integer accelerators with much higher efficiency [12]. Many modern edge AI chips (e.g. NVIDIA Jetson’s GPU, Google Edge TPU, ARM Ethos, etc.) provide optimized INT8 arithmetic support to accelerate quantized neural networks.

To illustrate, NVIDIA’s TensorRT library demonstrates that converting CNN models from FP32 to INT8 can increase throughput and reduce memory usage without significant accuracy loss[12][13]. Migacz (2017) [21] showed an INT8 post-training calibration approach in TensorRT that achieved nearly the same accuracy as FP32 on CNNs while greatly improving inference speed. The challenge, however, is that naive quantization from 32-bit to 8-bit can degrade model accuracy if not done carefully[14]. Edge deployment therefore requires model compression techniques that maintain accuracy while shrinking computation and memory footprint. Next, we review quantization - one of the most effective compression methods - and how it addresses edge constraints.

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

III. NEURAL NETWORK QUANTIZATION TECHNIQUES

Quantization compresses neural networks by representing weights and activations with lower-bit precision numbers (e.g. 8-bit integers) instead of the standard 32-bit floats[15][16]. This reduces memory usage by up to 4× for 8-bit (and 16× for 4-bit) and allows use of efficient integer arithmetic units[15]. For edge devices, quantization is particularly attractive because it directly cuts both storage and computation costs, often with minimal impact on model accuracy[17]. Gholami et al. (2022) [11] observe that moving from float to ≤4-bit integers "holds the potential to reduce memory and latency by a factor of 16×; and reductions of 4× to 8× are often realized in practice". Quantization has therefore become an active research area in efficient deep learning, yielding a rich set of methods. Broadly, we categorize quantization techniques into post-training quantization (PTQ) and quantization-aware training (QAT), and further into uniform vs. mixed-precision strategies, as well as recent innovations for preserving accuracy at low bit-widths.

- Post-Training Quantization (PTQ): PTQ methods convert a pre-trained model to lower precision without further training. Typically, a small calibration dataset (a few hundred unlabeled samples) is used to determine quantization parameters (scales, zero-points) layer by layer[19][20]. For example, one may collect the distribution of activations on calibration images to determine the optimal dynamic range to map into 8-bit. This approach is fast and doesn't require full retraining, making it ideal for rapid deployment on edge devices[21][22]. Integer-only inference as described by Jacob et al. (2018) [15] was an early successful PTQ approach: they quantized models like MobileNets to 8-bit and showed they could run end-to-end using integer arithmetic with only minor accuracy loss by carefully choosing quantization scales[23]. Another example, NVIDIA's calibration method in TensorRT [21], finds optimal scaling factors such that the FP32-to-INT8 conversion minimizes information loss[24]. Empirically, many CNNs can be quantized to 8-bit with <1% accuracy drop using PTQ techniques[25]. However, PTQ becomes more challenging for lower bit-widths (such as 4-bit or 2-bit): simply minimizing layer-wise MSE (Mean squared Error) of activations is often insufficient to preserve accuracy[26][20]. Research in the last few years has produced advanced PTQ algorithms to push bit-widths lower without retraining, which we discuss shortly.

- Quantization-Aware Training (QAT): In QAT, the model is trained (or fine-tuned) with quantization in the loop[19]. Simulated quantization of weights and activations is applied during training so that the network learns to compensate for quantization errors. This generally yields higher accuracy than PTQ, especially at ultra-low precision, at the cost of extra training time and requiring the training dataset[19][27]. For anomaly detection tasks, obtaining labeled data for full retraining may be difficult, so PTQ is attractive; but when some data is available, QAT can recover performance lost in PTQ. Liu et al. (2024) [19] compared PTQ vs QAT for anomaly detection models and found that QAT can close the accuracy gap, achieving performance "at par with the original 32-bit floating point" model in two of their cases by fine-tuning with quantization aware training [28]. Thus, a practical deployment might use PTQ first, and if accuracy is not satisfactory, employ QAT with some training data or synthetic data to further improve it. In summary, PTQ offers quick deployment and is easier when training data is scarce, whereas QAT offers optimal accuracy at the cost of retraining.

Uniform vs. Mixed Precision: Early quantization methods often used a single uniform bit-width (e.g. 8-bit for all layers). However, not all parts of a network tolerate quantization equally. Some layers are more "sensitive" - quantizing them causes a larger accuracy drop - while others can be quantized more aggressively (to fewer bits) with little impact [29][30]. This gave rise to mixed-precision quantization, where different layers use different bit-widths to balance accuracy and efficiency. Mixed precision aims to allocate higher precision to the more sensitive layers and lower precision to others, to reach a target model size or speed while minimizing accuracy loss[30][31]. The challenge is how to automatically select the precision for each layer from an exponential search space[32]. One breakthrough was Hessian-based sensitivity analysis. HAWQ (Hessian AWare Quantization) by Dong et al. (2019) [9] introduced a method to use the second-order derivatives (Hessian) of the loss w.r.t. layer parameters to estimate each layer's impact on the overall accuracy[33][31]. Layers with a small Hessian spectrum can be quantized more roughly (fewer bits) because they contribute less to the loss, whereas layers with a large spectrum (high curvature)

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

need more precision[33]. HAWQ automatically assigns mixed precision such that more sensitive layers (high Hessian eigenvalues) get higher bit-width. This allowed quantizing ResNet and Inception networks to mostly 4-bit weights with some 8-bit outliers, achieving similar or better accuracy than prior uniform quantization, and even outperformed reinforcement-learning-based bit allocation (HAQ) in some cases[34][35]. Follow-ups HAWQ-V2 [10] and HAWQ-V3 [33] further refined this approach (using Hessian trace weighting and dyadic precision constraints respectively) to improve robustness and meet hardware-specific constraints. Generally, Hessian-based methods have proven effective to determine layer importance without exhaustive search. An alternative approach, HAQ (Hardware-Aware Quantization) by Wang et al. (2019) [31], used reinforcement learning to search for the best bit allocation for each layer under a hardware efficiency constraint. HAQ's reward function combined model accuracy with a penalty for estimated latency on target hardware, thus directly optimizing for edge deployment efficiency[34]. HAQ demonstrated mixed-precision solutions that met FPGA (Field Programmable Gate Array) latency constraints better than uniform 8-bit. Other techniques include differentiable NAS for quantization: Wu et al. (2019) [32] treated the bit-width as a continuous parameter and used gradient-based search to find per-layer precisions that optimize a proxy loss (akin to neural architecture search). These methods show that significant memory and compute reduction is possible without sacrificing accuracy, by intelligent assignment of precision across layers.

Low-Bit and Adaptive Rounding Techniques: As quantization goes to very low bits (e.g. 4-bit, 2-bit), the primary difficulty becomes the rounding error of weights and the limited dynamic range of activations. Recent research has focused on novel PTQ methods to minimize quantization error. Adaptive rounding was introduced by Nagel et al. (2020) [22] in a method called AdaRound . Instead of naive rounding of weights (to nearest representable value), AdaRound optimizes the rounding decisions via a small optimization problem per layer: it adjusts each weight's rounding (up or down) to minimize the layer's output distortion, with a regularization to keep weights near their original value. This effectively learns whether to round a particular weight up or down, yielding better accuracy especially at 4-bit or less. Li et al. (2021) extended this idea in BRECQ (Bit-Resilient Quantization) [17], which performs block reconstruction: layers are grouped into blocks, and the quantization parameters (scales and rounding of weights) are optimized such that the block's output feature maps match the float model's output as closely as possible[36][18]. BRECQ demonstrated post-training 4-bit quantization of ResNet-50 on ImageNet with <1% top-1 accuracy drop, approaching the performance of full QAT, all without any labels by just using unlabeled data for calibration. This was a remarkable result - previously, such low precision typically required fine-tuning with the training set. The key insight is to use a few calibration images and reconstruct intermediate feature maps via optimization to preserve the model's predictions. Similarly, PD-Quant (Prediction Difference Quantization) by Liu et al. (2023) [18] introduced a metric that uses the global prediction difference between the original and quantized model to guide quantization parameter selection. Rather than optimizing layer-wise errors alone, PD-Quant tries to directly minimize the discrepancy in final outputs (network predictions) caused by quantization[26]. It also incorporates techniques to mitigate overfitting to the small calibration set by smoothing the activation distributions[18]. As a result, PD-Quant achieved state-of-the-art accuracy for ultra-low precision (e.g. 2-bit) on classification models, significantly closing the gap to full precision in those extreme cases[18].

Table 1 summarizes and compares several prominent quantization approaches in the literature, highlighting their technique, type (PTQ vs QAT), and notable results:

Method (Year)	Type	Technique
Banner et al. (2019) [1]	PTQ	Bias correction, smooth scaling for 4-bit weights & activations[30]
HAWQ (2019) [9]	PTQ (mixed)	Hessian-based layer sensitivity[33]
HAQ (2019) [31]	PTQ	RL-based bit allocation (hardware-aware)

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

	(mixed)	
AdaRound (2020) [22]	PTQ	Optimized weight rounding (layer-wise)
BRECQ (2021) [17]	PTQ	Block reconstruction, optimized scales & rounds
PD-Quant (2023) [18]	PTQ	Prediction difference metric + dist. adjustment[26]
Polino et al. (2018) [25]	QAT + Distill.	Knowledge distillation with quantization
Once-for-All (2020) [5]	Training (NAS)	Train supernet and then specialize subnetwork for device
FastFlow (2021) [34]	N/A (model)	*Lightweight normalizing flow model for anomaly detection*
Liu et al. (2024) [19]	PTQ vs QAT study	Anomaly detection models, calibration study on edge device

Table 1: Overview of representative methods relevant to network quantization and efficient anomaly detection. (FastFlow is an example anomaly detection model optimized for speed, included for context.)

How to visualize these: The above comparisons can be expanded into a diagram or flowchart in a full paper. For example, one could create a Figure 1 as a flow diagram of a generic PTQ algorithm versus QAT: illustrating how a pre-trained model and calibration data go through a quantization algorithm (with steps like "Hessian analysis" for HAWQ or "layer reconstruction optimization" for BRECQ) to produce a quantized model, while QAT involves a training loop with fake-quantization. Figure 2 could graphically compare the sensitivity of different layers, showing how mixed precision allocates high bits to sensitive layers (perhaps via a bar graph of Hessian values and chosen bit-widths, as done in HAWQ). Additionally, a table or bar chart figure could highlight the accuracy vs model size trade-off of these methods (e.g. float32 baseline vs 8-bit vs 4-bit accuracy on a benchmark). These visual summaries help readers quickly grasp the differences in approach (e.g. analytical Hessian-based vs learning-based vs knowledge-distillation approaches)..

IV. INDUSTRIAL ANOMALY DETECTION METHODS

Industrial anomaly detection refers to identifying defects or abnormal conditions in manufactured products using computer vision. This is often cast as an unsupervised learning problem, since only normal (defect-free) samples may be available for training - defects can be rare and varied. A prominent benchmark is the MVTec Anomaly Detection (MVTec AD) dataset introduced by Bergmann et al. (2019) [3]. MVTec AD contains 5354 images across 15 object and texture categories, with over 70 types of defects (e.g. scratches, dents, contaminants) and pixel-precise ground truth for anomaly regions[3]. It has become the standard for evaluating unsupervised defect detection algorithms. Traditional approaches included training autoencoders or GANs on normal images and detecting anomalies by reconstruction error. For example, early deep learning methods used convolutional autoencoders to reproduce normal images, flagging deviations in the output as anomalies[3]. Recent state-of-the-art (SOTA) methods leverage feature embeddings from pre-trained networks and statistical models. PatchCore (Roth et al., 2022) [26] and PaDiM (Defard et al., 2021) [7] are methods that build a distribution of normal feature patches (using pre-trained CNN features) and detect anomalies by measuring distances of new image patches to this distribution. PatchCore in particular achieved an image-level anomaly detection AUROC of 99.6% on MVTec AD, halving the error of previous SOTA[26]. Another approach, FastFlow [34], uses normalizing flows to learn the distribution of

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

normal images in feature space; FastFlow is designed to be light-weight by utilizing 2D CNN flows, and it produces anomaly segmentations in real-time. According to Yu et al., FastFlow reached 94.1% AUROC on MVTec AD with significantly faster inference than GAN-based methods[34]. These unsupervised methods typically operate as one-class models: a separate model (or at least separate feature distribution) is trained for each object category, since the appearance of normal vs anomalous can differ widely per object. However, training many models can be memory-intensive for deployment. Interestingly, Jena et al. (2024) [19] showed that a unified multi-class anomaly model can perform on par with individual one-class models on MVTec AD[19] Their multi-class model learns to detect anomalies across all object types, which could be beneficial for edge deployment as a single model can handle multiple defect types, reducing overall memory usage.

In some industrial cases, a small number of defect examples are available or defect types are known, making it a supervised defect detection problem (treated like a segmentation or classification task). For instance, the NEU-DET steel defect dataset contains images of steel surfaces labeled with six defect categories (crazing, inclusion, patches, etc.)[36]. Models can be trained to classify or localize these defects directly. Tang et al. (2023) [30] propose a fully supervised segmentation network with multi-scale features and sample selection for defect detection, demonstrating high accuracy on NEU-DET and related datasets. Another dataset, KolektorSDD2 (KSDD2), introduced by Božič et al. (2021) [4], contains over 3000 images of surfaces with defects vs. normal, designed for segmentation tasks[4].Their work explored mixed supervision: using a combination of a few fully-annotated images and many weakly-labeled images to reduce annotation cost[4][4]. In fully supervised settings, high-capacity CNNs (e.g. U-Net, DeepLab) can achieve very high defect detection accuracy when enough data is provided[4]. But these models can be even larger than unsupervised ones, making edge deployment difficult without compression.

Edge Deployment of Anomaly Detection Models: Whether unsupervised or supervised, modern anomaly detection methods often rely on deep feature extractors (e.g. ResNet-50, Wide ResNet, or EfficientNet backbones) which have millions of parameters. For example, PatchCore uses WideResNet-50 features (which is a heavy model), and FastFlow uses a DenseNet or ResNet backbone. Running these in real-time on a CPU or small GPU requires optimization. One approach is to start with a smaller, efficient architecture: models like MobileNetV2/V3 [13][14] or EfficientNet [29] were designed for mobile/edge usage, with depthwise separable convolutions and optimized layer dimensions to drastically cut down operations. A MobileNet can be 32× smaller than a ResNet-50, albeit usually at some accuracy cost[34]. In anomaly detection, some works have adopted lightweight networks as feature extractors or student models (for knowledge distillation approaches). However, even efficient architectures benefit from quantization. For instance, an EfficientNet-B0 quantized to INT8 would run significantly faster and use 4× less memory than the same EfficientNet in float16. Multiple strategies are complementary: one can use a compact architecture, prune unnecessary filters, and apply quantization together to meet strict latency/size requirements. Polino et al. (2018) [25] combined pruning and quantization with knowledge distillation: they train a smaller quantized "student" network to mimic a large "teacher" network's outputs, achieving high accuracy in a much smaller model. This strategy could be very useful for anomaly detection on edge: one could train a large model on the server (or cloud) for accuracy, then distill its knowledge into a tiny quantized model that runs on the device, as explored by some TinyML approaches.

From the EdgeSight abstract, the proposed solution emphasizes defect-sensitive quantization - effectively tailoring the compression to the anomaly detection task. This likely involves identifying which layers of the defect detection network are most crucial to detecting anomalies (e.g. layers extracting fine textures or subtle features) and assigning them higher precision. This concept aligns with the general idea of layer sensitivity we discussed (HAWQ's Hessian or other metrics), but specifically for anomaly detection, the importance might not correlate with classification loss Hessian. Instead, one might measure impact on anomaly detection performance. For example, EdgeSight might evaluate how quantizing each layer affects an anomaly detection metric (like AUROC or some anomaly score) and then allocate bit-widths accordingly. This task-specific sensitivity analysis is a novel spin that goes beyond generic sensitivity metrics. It ensures that compressing the model does not destroy the information needed to detect small defects. One could imagine an experiment where each layer is quantized and

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

the increase in anomaly detection error is measured, yielding a ranking of layer importance for the task - then the mixed-precision quantization algorithm uses this ranking to decide bit allocations under hardware constraints (e.g. a total model size budget). This approach echoes recent ideas in the quantization literature where activation sensitivity scores are used to find which layers are hardest to quantize[37]. Indeed, a very recent study on LLMs by Xia et al. (2025) proposed identifying "sensitive layers" via outlier detection on activations and giving them more bits[37],which improved 4-bit accuracy notably. EdgeSight's defect-sensitivity analysis can be seen as an analogous approach in the vision domain: layers that produce features critical for anomaly discrimination (for example, final embedding layers in a feature-model-based detector) would be quantized at 8-bit or higher, whereas early layers or less relevant layers might go to 4-bit without issue. The outcome reported - preserving accuracy within one percentage point of the FP32 baseline while reducing model size by 73% and latency by 70% - demonstrates the effectiveness of such a tailored compression. It means the quantized model (likely INT8 for most layers, and perhaps some layers in higher precision or with careful calibration) detects defects nearly as well as the original, but is almost one-quarter the size and much faster.

V. DEPLOYMENT ON EDGE HARDWARE (JETSON, ORANGE PI) WITH PYTORCH

Implementing these quantization techniques for deployment involves both software framework support and hardware acceleration. PyTorch [24] has built-in support for quantization: as of 2019, PyTorch introduced a quantization toolkit for PTQ and QAT, including modules to simulate quantized ops and backend support for CPU (via FBGEMM or QNNPACK)[19]. This allows a developer to take a model (e.g. a ResNet-based anomaly detector), insert quantization observer modules to collect stats, and then convert the model to a quantized version. PyTorch's QAT can then fine-tune the quantized model. Notably, PyTorch QAT was used by Liu et al. (2024) [19] in their edge anomaly detection study: they compared PTQ vs QAT using PyTorch's tools, and found QAT-improved models matched FP32 accuracy in two of three cases[19]. For actual inference on devices like NVIDIA Jetson, one common workflow is to export the PyTorch model (which may be quantized or just a regular model) to ONNX format and then use NVIDIA TensorRT to optimize and run it. TensorRT will take a quantized model or even a full precision model and perform its own PTQ using calibration data to generate a fast INT8 engine, as described by Migacz [21]. In EdgeSight, the authors mention deploying on NVIDIA Jetson (which has a GPU and supports TensorRT INT8) and Orange Pi. The Orange Pi is typically an ARM-based single-board computer (similar to Raspberry Pi). If it lacks a specialized AI accelerator, the quantized model would run on the CPU (possibly utilizing Neon instructions for int8). In such a case, frameworks like TFLite or OpenVINO could be used as they are optimized for ARM CPU inference with int8. However, given the context, they might have run PyTorch with QNNPACK backend on the Orange Pi, achieving ~30 FPS (Frames Per Second) for their quantized model, which aligns with their claim of real-time performance.

On Jetson, the process likely involved leveraging TensorRT or the ONNX Runtime with TensorRT delegate for maximal speed. Jetson's INT8 speedups are significant - it has been reported that e.g. ResNet-50 INT8 can run 2-4× faster than FP16 on a Jetson Xavier NX (which was the model used in [19]) while also reducing power. EdgeSight notes a 72% reduction in energy consumption, which is crucial for embedded deployments. This implies that not only is the model smaller and faster, but it also draws much less power (likely because the GPU spends less time active per frame, and int8 computations are more energy-efficient). This level of improvement can make the difference between a feasible deployment on a fanless device versus one that overheats or drains battery quickly.

From an implementation perspective, one key consideration is calibration for PTQ on anomaly detectors. Since anomaly detection might involve output distributions that are not typical classification probabilities, calibrating activation ranges properly is important. As mentioned by Liu et al. [19], they explored two calibration methods for PTQ and found one performed notably better for the unsupervised task[19]. While they did not detail it in the abstract, this could refer to using either

i) only normal images vs a mix of normal+anomaly images for calibration, or
ii) different metrics such as percentile clipping vs max range.

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 10 March 2026 **Revised**: 18 May 2026 **Accepted**: 15 June 2026

Their finding underscores that techniques from classification may not directly translate - e.g. using a few anomalous samples in calibration might skew the scale ranges. Likely, using only normal data (which the model is primarily trained on) for calibration gave better results in unsupervised AD (Anomaly Detection), or vice versa. This is something an EdgeSight-like framework would account for: being task-aware includes using appropriate calibration data representative of the operating conditions.

To facilitate developers, frameworks now also provide mixed-precision training and export. For example, PyTorch and TensorFlow let you train in lower precision (FP16) or quantize weights, and hardware like Jetson's GPU supports FP16 and INT8 math natively. NVIDIA Jetson devices particularly shine with quantization because their GPU Tensor Cores can execute INT8 matrix operations with high throughput (and FP16 as well). NVIDIA's DeepStream SDK and TensorRT encourage using INT8 for any vision DNN (Deep Neural Network) when possible, to maximize the number of video streams/processes that can run in real-time on a single device. On the CPU side (Orange Pi), quantization is basically the only way to meet real-time speeds, since a FP32 model of any complexity would be too slow in Python on an ARM CPU. By quantizing and possibly using C++ runtime or vectorized ops, one can achieve significant speedup. For instance, an 8-bit model can leverage ARM Neon vector instructions processing 16 or more values in parallel, whereas FP32 would process far fewer per cycle.

In summary, **edge deployment of anomaly detection requires an end-to-end pipeline:** train a high-accuracy model on server, compress it via quantization (and possibly distillation or pruning), validate that accuracy is preserved on tasks like MVTec AD or KSDD2 (making use of defect-sensitive metrics for any layer-specific tweaks), then use hardware-appropriate runtimes (TensorRT on Jetson, TFLite or ONNX on ARM CPU) to achieve the desired throughput. The collective advances surveyed - from quantization algorithms (HAWQ, BRECQ, etc.) to efficient model design (MobileNets, EfficientNet) and to specialized anomaly detection models (FastFlow, PatchCore) - are enabling what the EdgeSight abstract claims: real-time, accurate defect detection on inexpensive devices with >70% reduction in memory and energy. This opens up new possibilities for deploying AI-powered quality control in distributed manufacturing environments without relying on bulky servers or cloud connectivity.

VI. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we reviewed the landscape of neural network quantization and its application to edge-based industrial anomaly detection. Quantization has proven to be a linchpin technology for compressing models, offering 4-8× reductions in memory and compute with minimal accuracy loss when done intelligently[18][17]. Techniques like mixed-precision allocation (e.g. HAWQ [9]) and advanced post-training optimization (BRECQ [17], PD-Quant [18]) push the boundaries of low-bit quantization, making even 4-bit networks feasible for complex vision tasks. These developments coincide with the growing demand for on-device inference in Industry 4.0 settings, where edge computing is essential for low-latency, secure, and reliable operation[3][27]. We also discussed how industrial anomaly detection, as a mission-critical application, benefits from task-specific considerations - such as defect-sensitive layer analysis - when compressing models. The EdgeSight framework exemplifies a holistic approach: leveraging quantization while accounting for the unique characteristics of anomaly detection data, it reportedly achieves nearly the same accuracy as full precision on datasets like MVTec AD and KSDD2, with a fraction of the resource usage.

Looking forward, there are several avenues to explore. One is automated mixed-precision tools that incorporate not just classification loss or Hessian, but custom objective functions (e.g. anomaly detection scores or even multi-task objectives) to decide quantization strategies - effectively bringing AutoML to edge compression. Another is the combination of quantization with other compression methods: for instance, pruning redundant filters from a backbone and then quantizing it can compound the gains. However, aggressively combining these can sometimes amplify errors, so research into joint optimization (perhaps distillation guiding both pruning and quantization as in [25]) is valuable. Additionally, as vision models evolve (transformers, MLP-mixers, etc.), ensuring that quantization techniques keep up - e.g. the recent PTQ4ViT [35] specifically for Vision Transformers (ViT) - will be important for anomaly detection, since ViT-based models are starting to enter this field for their

representation power. Finally, a promising trend is edge-first anomaly detection design: creating models and algorithms with quantization in mind from the outset. Rather than quantizing a large model post-hoc, one could train a model that inherently uses quantized operations or is constrained during training to be quantization-friendly (for example, avoiding operations that are hard to quantize like certain normalizations or non-linearities). This co-design of model and compression will likely yield the most efficient solutions.

In conclusion, the synergy of efficient model design, advanced quantization, and knowledge of the task domain is enabling industrial anomaly detection to break free from the cloud and thrive on the edge. This survey highlighted how far the community has come in achieving total recall of defects at the factory floor, in real-time, on tiny devices - and charted some paths ahead to continue this progress.

REFERENCES

[1] Banner, R., Nahshan, Y., & Soudry, D. (2019). Post training 4-bit quantization of convolutional networks for rapid-deployment. Advances in Neural Information Processing Systems, 32.[30][34]

[2] Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432.

[3] Bergmann, P., Fauser, M., Sattlegger, D., & Steger, C. (2019). MVTec AD: A comprehensive real-world dataset for unsupervised anomaly detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 9592-9600.[3]

[4] Božič, J., Tabernik, D., & Skočaj, D. (2021). Mixed supervision for surface-defect detection: From weakly to fully supervised learning. Computers in Industry, 129, 103459.[4]

[5] Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2020). Once-for-all: Train one network and specialize it for efficient deployment. In International Conference on Learning Representations.

[6] Cao, K., Liu, Y., Meng, G., & Sun, Q. (2020). An overview on edge computing research. IEEE Access, 8, 85714-85728.[1]

[7] Defard, T., Setkov, A., Loesch, A., & Audigier, R. (2021). PaDiM: A patch distribution modeling framework for anomaly detection and localization. In International Conference on Pattern Recognition.

[8] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[9] Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., & Keutzer, K. (2019). HAWQ: Hessian aware quantization of neural networks with mixed-precision. In Proceedings of the IEEE/CVF International Conference on Computer Vision, 293-302.[33][34]

[10] Dong, Z., Yao, Z., Arfeen, D., Gholami, A., Mahoney, M. W., & Keutzer, K. (2020). HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. Advances in Neural Information Processing Systems, 33, 18518-18529.

[11] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2022). A survey of quantization methods for efficient neural network inference. In Low-Power Computer Vision (pp. 291-341). CRC Press.[15][18]

[12] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[13] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., … & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[14] Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., … & Le, Q. V. (2019). Searching for MobileNetV3. In Proceedings of the IEEE/CVF International Conference on Computer Vision.

[15] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., … & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2704-2713.[23]

[16] Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342.

[17] Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., … & Gu, S. (2021). BRECQ: Pushing the limit of post-training quantization by block reconstruction. In International Conference on Learning Representations.[36]

[18] Liu, Y., Zhang, H., & Wang, L. (2023). PD-Quant: Post-training quantization based on prediction difference metric. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.[26][18]

[19] Liu, J., Chen, S., & Wu, X. (2024). Unified anomaly detection methods on edge devices using knowledge distillation and quantization. arXiv preprint arXiv:2407.02968.[28]

[20] Lou, Q., Guo, F., Kim, M. H., Liu, L., & Jiang, L. (2020). AutoQ: Automated kernel-wise neural network quantization. In International Conference on Learning Representations.

[21] Migacz, S. (2017). 8-bit inference with TensorRT. NVIDIA GPU Technology Conference.[12]

[22] Nagel, M., van Baalen, M., Blankevoort, T., & Welling, M. (2020). Up or down? Adaptive rounding for post-training quantization. In International Conference on Machine Learning. PMLR.

[23] Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., van Baalen, M., & Blankevoort, T. (2021). A white paper on neural network quantization. arXiv preprint arXiv:2106.08295.

[24] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32.

[25] Polino, A., Pascanu, R., & Alistarh, D. (2018). Model compression via distillation and quantization. In International Conference on Learning Representations.

[26] Roth, K., Pemula, L., Zepeda, J., Schölkopf, B., Brox, T., & Gehler, P. (2022). Towards total recall in industrial anomaly detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[27] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637-646.[2]

[28] Stock, P., Joulin, A., Gribonval, R., Graham, B., & Jégou, H. (2020). And the bit goes down: Revisiting the quantization of neural networks. In International Conference on Learning Representations.

[29] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In Proceedings of the 36th International Conference on Machine Learning. PMLR.

[30] Tang, X., Qiu, T., Gao, F., Yu, Z., & Zhao, M. (2023). A joint multi-scale network and sample selection framework for automatic defect segmentation. IEEE Transactions on Instrumentation and Measurement, 72, 1-13.

[31] Wang, K., Liu, Z., Lin, Y., Lin, J., & Han, S. (2019). HAQ: Hardware-aware automated quantization with mixed precision. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[32] Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., & Keutzer, K. (2019). Mixed precision quantization of ConvNets via differentiable neural architecture search. arXiv preprint arXiv:1812.00090.

[33] Yao, Z., Dong, Z, Zheng, Z., Gholami, A., Yu, J., Tan, E., … & Keutzer, K. (2021). HAWQ-V3: Dyadic neural network quantization. In International Conference on Machine Learning. PMLR.

[34] Yu, J., Zheng, Y., Wang, X., Li, W., Wu, Y., Zhao, R., & Wu, L. (2021). FastFlow: Unsupervised anomaly detection and localization via 2D normalizing flows. arXiv preprint arXiv:2111.07677.

[35] Yuan, Z., Xue, C, Chen, Y., Wu, Q., & Sun, G. (2022). PTQ4ViT: Post-training quantization for vision transformers with twin uniform quantization. In European Conference on Computer Vision. Springer.

[36] Lv et al., Sensors 2020

[37] Zhang et sal., arXiv:2503.06518 https://arxiv.org/html/2503.06518v1s