COMPUTER SCIENCE BULLETIN

VOL: 09 ISSUE: 01 (2026)

E-ISSN: 2959-5347 P-ISSN: 2959-5339

A Lightweight Deep Learning Framework for Real-Time Cyberattack Detection in Internet of Things Networks

Marco Tulio Ribeiro

Microsoft Research, Redmond, Washington, USA

Abstract. *The rapid expansion of Internet of Things (IoT) networks introduces significant cybersecurity challenges because connected sensors, gateways, and embedded controllers operate with limited memory, processing capacity, and energy. Conventional intrusion detection solutions may impose computational costs that exceed the capabilities of these devices, while cloud-only analysis adds latency and creates privacy concerns. This narrative review develops a conceptual lightweight deep learning framework for real-time cyberattack detection at the IoT edge. The proposed architecture combines flow-based feature extraction, selective preprocessing, a compact one-dimensional convolutional network, calibrated risk scoring, and adaptive response at the gateway. It discusses feature selection, class imbalance, model compression, data drift, adversarial resilience, and deployment constraints. An illustrative design chart compares qualitative trade-offs between model classes; it does not represent measured experimental findings. The article outlines an evaluation protocol using reproducible train-test separation, device-aware latency measurement, and cross-dataset validation. The central conclusion is that practical reliability requires jointly optimizing detection quality, false-alarm burden, inference time, memory use, and operational maintainability rather than accuracy alone.*

Keywords: *Internet of Things, intrusion detection, lightweight deep learning, edge computing, cyberattack detection*

Introduction

Internet of Things systems connect physical processes to software services through heterogeneous sensors, wireless networks, and distributed gateways. Their diversity creates a broad attack surface: compromised devices can be recruited into botnets, communication channels can be abused for scanning and denial-of-service traffic, and poorly managed credentials may allow unauthorized access. Real-time detection must recognize malicious behavior while leaving sufficient computational headroom for the primary sensing and control functions. This paper presents a research-grounded conceptual design rather than claiming a newly validated experimental model.Heterogeneous deployments complicate consistent policy enforcement because industrial sensors, home cameras, medical peripherals, and agricultural

monitors expose different services and operate on different update schedules. Attackers exploit default credentials, vulnerable firmware, insecure APIs, and poorly segmented networks. The threat model should explicitly distinguish external traffic anomalies, lateral movement within a local network, and misuse by an already compromised endpoint.Practical implication: Heterogeneous deployments complicate consistent policy enforcement because industrial sensors, home cameras, medical peripherals, and agricultural monitors expose different services and operate on different update schedules. Attackers exploit default credentials, vulnerable firmware, insecure APIs, and poorly segmented networks. A threat model may explicitly distinguish external traffic anomalies, lateral movement within a local network, and misuse by an already compromised endpoint. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Operational requirements are not limited to attack classification. Detection must remain available under packet bursts, tolerate intermittent connectivity, preserve user privacy, and avoid disrupting time-critical functions. A gateway-based monitor can inspect summarized flows without collecting full application payloads, but encrypted traffic and protocol diversity reduce direct observability.Practical implication: Operational requirements are not limited to attack classification. Detection must remain available under packet bursts, tolerate intermittent connectivity, preserve user privacy, and avoid disrupting time-critical functions. A gateway-based monitor can inspect summarized flows without collecting full application payloads, but encrypted traffic and protocol diversity reduce direct observability. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

IoT Threat Landscape and Security Requirements

A robust design documents which assets are protected, which traffic is observable, the expected attacker capabilities, and the acceptable false-positive rate. These assumptions are necessary to interpret model performance and to determine whether a detection alert should trigger automated containment or merely human review.

Practical implication:

A robust design documents which assets are protected, which traffic is observable, the expected attacker capabilities, and the acceptable false-positive rate. These assumptions are necessary to interpret model performance and to determine whether a detection alert may trigger automated containment or merely human review. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Limitations of Conventional Intrusion Detection:

Signature-based systems provide understandable rules and can be efficient for known threats, yet they may fail when attacks change their payloads, ports, or timing. Anomaly-based systems can identify deviations from normal behavior but may confuse legitimate device updates, new services, or seasonal workload changes with malicious activity.Practical implication: Signature-based systems provide understandable rules and can be efficient for known threats, yet they may fail when attacks change their payloads, ports, or timing. Anomaly-based systems can identify deviations from normal behavior but may confuse legitimate device updates, new services, or seasonal workload changes with malicious activity. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Traditional machine-learning methods such as decision trees, support vector machines, and random forests offer useful baselines. Their effectiveness depends heavily on preprocessing, feature engineering, and representative training data. Comparisons with neural networks should include tuning effort and operational cost, not simply

headline accuracy.Practical implication: Traditional machine-learning methods such as decision trees, support vector machines, and random forests offer useful baselines. Their effectiveness depends heavily on preprocessing, feature engineering, and representative training data. Comparisons with neural networks may include tuning effort and operational cost, not simply headline accuracy. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Deep learning can learn discriminative patterns from sequential or flow-level observations. However, large models increase inference latency, memory consumption, and maintenance complexity. A compact architecture is therefore attractive when it delivers an acceptable detection trade-off within an explicitly defined device budget.Practical implication: Deep learning can learn discriminative patterns from sequential or flow-level observations. However, large models increase inference latency, memory consumption, and maintenance complexity. A compact architecture is therefore attractive when it delivers an acceptable detection trade-off within an explicitly defined device budget. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Data Collection and Network Traffic Representation:

An intrusion detection pipeline begins with traffic acquisition from a switch mirror port, router, or IoT gateway. Rather than retaining every packet, a flow extractor may aggregate packet counts, byte counts, connection duration, protocol identifiers, timing statistics, and directional ratios. These measurements reduce storage and may lessen exposure of sensitive payloads.Practical implication: An intrusion detection pipeline begins with traffic acquisition from a switch mirror port, router, or IoT gateway. Rather than retaining every packet, a flow extractor may aggregate packet counts, byte counts, connection duration, protocol identifiers, timing statistics, and directional ratios. These measurements reduce storage and may lessen exposure of sensitive payloads. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Feature construction must prevent information leakage. Fields such as capture-specific timestamps, dataset labels encoded in filenames, or host identifiers correlated with attacks can produce deceptively strong evaluation scores. Training and test partitions should be separated by capture session, device, or time when realistic generalization is the goal.Practical implication: Feature construction must prevent information leakage. Fields such as capture-specific timestamps, dataset labels encoded in filenames, or host identifiers correlated with attacks can produce deceptively strong evaluation scores. Training and test partitions may be separated by capture session, device, or time when realistic generalization is the goal. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Missing values, categorical encoding, outliers, and scaling should be handled using transformations fitted only on training data. For live deployment, the gateway must reproduce exactly the same feature definitions and normalization parameters; otherwise, discrepancies between offline and online pipelines can dominate model errors.Practical implication: Missing values, categorical encoding, outliers, and scaling may be handled using transformations fitted only on training data. For live deployment, the gateway must reproduce exactly the same feature definitions and normalization parameters; otherwise, discrepancies between offline and online pipelines can dominate model errors. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Lightweight Deep Learning Architecture:

The proposed detector accepts a fixed-length vector of normalized network-flow features. A small one-dimensional convolutional block captures local interactions among ordered feature groups, followed by global average pooling and a compact fully connected classification head. Depthwise separable convolution may further reduce multiply-accumulate operations when the feature representation justifies its use.Practical implication: A proposed detector accepts a fixed-length vector of normalized network-flow features. A small one-dimensional convolutional block captures local interactions among ordered feature groups, followed by global average pooling and a compact fully connected classification head. Depthwise separable convolution may further reduce multiply-accumulate operations when the feature representation justifies its use. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.The model is deliberately smaller than a generic deep classifier: a limited number of filters, narrow hidden layers, and a compact output head restrict parameter growth. Batch normalization can stabilize training, whereas dropout and weight decay may reduce overfitting. The exact layer sizes should be chosen through device-constrained validation rather than treated as universal defaults.Practical implication: A model is deliberately smaller than a generic deep classifier: a limited number of filters, narrow hidden layers, and a compact output head restrict parameter growth. Batch normalization can stabilize training, whereas dropout and weight decay may reduce overfitting. A exact layer sizes may be chosen through device-constrained validation rather than treated as universal defaults. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.The output is a risk score for benign or malicious behavior, optionally with a multi-class attack label. Probability calibration and threshold selection are essential because an apparently confident classifier can still generate excessive alerts in a network where attacks are rare. The gateway may escalate uncertain flows to a more capable upstream analyzer.Practical implication: A output is a risk score for benign or malicious behavior, optionally with a multi-class attack label. Probability calibration and threshold selection are essential because an apparently confident classifier can still generate excessive alerts in a network where attacks are rare. A gateway may escalate uncertain flows to a more capable upstream analyzer. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Model Training and Class Imbalance:

Training should use stratified inspection of attack classes while preserving realistic temporal or device-level holdouts. Weighted cross-entropy, focal loss, and carefully designed resampling can mitigate minority-class underrepresentation, but synthetic examples must never be generated using information from the evaluation split.Practical implication: Training may use stratified inspection of attack classes while preserving realistic temporal or device-level holdouts. Weighted cross-entropy, focal loss, and carefully designed resampling can mitigate minority-class underrepresentation, but synthetic examples must never be generated using information from the evaluation split. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Early stopping and hyperparameter selection should rely on a validation partition independent of the final test set. Repeated random trials with fixed seeds, reported confidence intervals, and disclosure of software versions improve reproducibility. Attack detection should be assessed using precision, recall, macro-F1, false-positive rate, and area under the precision-recall curve.Practical implication: Early stopping and hyperparameter selection may rely on a

validation partition independent of the final test set. Repeated random trials with fixed seeds, reported confidence intervals, and disclosure of software versions improve reproducibility. Attack detection may be assessed using precision, recall, macro-F1, false-positive rate, and area under the precision-recall curve. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.A classifier optimized only for overall accuracy can appear successful even if it misses uncommon but serious attacks. Thresholds should reflect the relative cost of false alarms and missed intrusions. An operationally relevant experiment can estimate how many alerts a security operator would receive per day under an assumed traffic volume.Practical implication: A classifier optimized only for overall accuracy can appear successful even if it misses uncommon but serious attacks. Thresholds may reflect the relative cost of false alarms and missed intrusions. An operationally relevant experiment can estimate how many alerts a security operator would receive per day under an assumed traffic volume. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Edge Deployment and Runtime Optimization:

Gateway deployment requires quantifying model size, peak RAM usage, processor utilization, energy demand, and per-flow inference latency. These measurements should be obtained on the intended hardware, because a fast workstation benchmark does not establish suitability for a low-power gateway.Practical implication: Gateway deployment requires quantifying model size, peak RAM usage, processor utilization, energy demand, and per-flow inference latency. These measurements may be obtained on the intended hardware, because a fast workstation benchmark does not establish suitability for a low-power gateway. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Post-training quantization can shrink model weights and accelerate compatible inference kernels. Structured pruning and knowledge distillation may also reduce runtime cost, although compression can disproportionately damage minority-attack recall. Any compressed variant should therefore be reevaluated on the same untouched test set.Practical implication: Post-training quantization can shrink model weights and accelerate compatible inference kernels. Structured pruning and knowledge distillation may also reduce runtime cost, although compression can disproportionately damage minority-attack recall. Any compressed variant may therefore be reevaluated on the same untouched test set. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.The deployment pipeline must include bounded queues, overload handling, safe model updates, and a fallback policy when the detector becomes unavailable. Monitoring only model inference time understates end-to-end delay; packet capture, feature aggregation, serialization, and response execution may contribute substantially to detection latency.Practical implication: A deployment pipeline must include bounded queues, overload handling, safe model updates, and a fallback policy when the detector becomes unavailable. Monitoring only model inference time understates end-to-end delay; packet capture, feature aggregation, serialization, and response execution may contribute substantially to detection latency. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Real-Time Detection and Response Workflow:

In operation, incoming packets are summarized into short flow windows, transformed using stored preprocessing parameters, and passed to the compact model. A decision engine compares the calibrated risk score against a policy threshold. Low-risk flows proceed normally, while

high-risk flows can trigger logging, rate limiting, isolation, or analyst notification.Practical implication: In operation, incoming packets are summarized into short flow windows, transformed using stored preprocessing parameters, and passed to the compact model. A decision engine compares the calibrated risk score against a policy threshold. Low-risk flows proceed normally, while high-risk flows can trigger logging, rate limiting, isolation, or analyst notification. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Response policies should be graduated. Automatically blocking traffic based on a single uncertain prediction may interrupt safety-critical services. Repeated suspicious windows, corroborating rules, device criticality, and recent historical behavior can provide context before a disruptive action is taken.Practical implication: Response policies may be graduated. Automatically blocking traffic based on a single uncertain prediction may interrupt safety-critical services. Repeated suspicious windows, corroborating rules, device criticality, and recent historical behavior can provide context before a disruptive action is taken. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.An auditable event record should retain the model version, feature schema version, decision threshold, relevant summary statistics, and final action. This supports incident investigation and enables rollback if a newly deployed model behaves unexpectedly. Privacy-preserving retention limits remain important even when only metadata is stored.Practical implication: An auditable event record may retain the model version, feature schema version, decision threshold, relevant summary statistics, and final action. This supports incident investigation and enables rollback if a newly deployed model behaves unexpectedly. Privacy-preserving retention limits remain important even when only metadata is stored. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Evaluation Protocol and Comparative Analysis:

A credible evaluation compares the proposed lightweight network against simple rule-based methods, a tuned classical baseline, and a larger neural model under consistent feature sets and data splits. Publicly discussed datasets such as N-BaIoT, Bot-IoT, and CICIoT2023 offer complementary traffic scenarios, but each contains collection artifacts and distributional limitations.Practical implication: A credible evaluation compares the proposed lightweight network against simple rule-based methods, a tuned classical baseline, and a larger neural model under consistent feature sets and data splits. Publicly discussed datasets such as N-BaIoT, Bot-IoT, and CICIoT2023 offer complementary traffic scenarios, but each contains collection artifacts and distributional limitations. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Report confusion matrices, per-class precision and recall, macro-averaged F1, false-positive rate, calibration error, model parameters, serialized size, and end-to-end latency. Cross-dataset or cross-device testing is especially important because excellent within-dataset performance may not transfer to unseen device families.Practical implication: Report confusion matrices, per-class precision and recall, macro-averaged F1, false-positive rate, calibration error, model parameters, serialized size, and end-to-end latency. Cross-dataset or cross-device testing is especially important because excellent within-dataset performance may not transfer to unseen device families. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.The chart in this review is illustrative, showing a hypothesized engineering trade-off rather than measured results. No accuracy, latency, or energy figures should be attributed to the proposed architecture until it has been implemented and

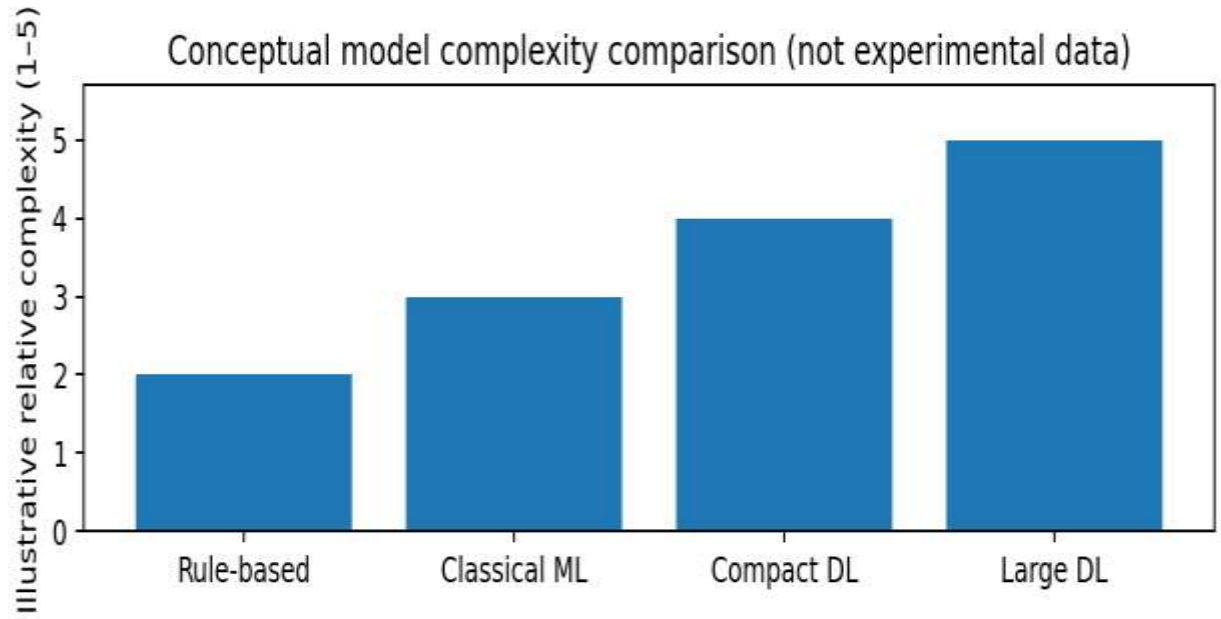
independently benchmarked. A complete experimental paper would include dataset versions, split scripts, source code, hardware specifications, and uncertainty estimates.Practical implication: A chart in this review is illustrative, showing a hypothesized engineering trade-off rather than measured results. No accuracy, latency, or energy figures may be attributed to the proposed architecture until it has been implemented and independently benchmarked. A complete experimental paper would include dataset versions, split scripts, source code, hardware specifications, and uncertainty estimates. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.

Reliability, Generalization, and Future Research:

IoT behavior changes as firmware is updated, users add devices, and attackers adapt. This concept drift can invalidate thresholds and feature distributions. A deployed system should monitor shifts in incoming features, changes in alert rates, and sampled human-confirmed outcomes while preserving a stable rollback checkpoint.Practical implication: IoT behavior changes as firmware is updated, users add devices, and attackers adapt. This concept drift can invalidate thresholds and feature distributions. A deployed system may monitor shifts in incoming features, changes in alert rates, and sampled human-confirmed outcomes while preserving a stable rollback checkpoint. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Adversarial evasion remains a major concern because attackers may manipulate packet timing, traffic volume, or protocol fields to mimic benign activity. Defenses include robust feature engineering, adversarial evaluation, complementary signatures, and layered monitoring. No single learned classifier should be assumed to provide comprehensive security.Practical implication: Adversarial evasion remains a major concern because attackers may manipulate packet timing, traffic volume, or protocol fields to mimic benign activity. Defenses include robust feature engineering, adversarial evaluation, complementary signatures, and layered monitoring. No single learned classifier may be assumed to provide comprehensive security. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.Future work should evaluate federated updates, self-supervised representations, explainable alert summaries, privacy-preserving telemetry, and energy-aware adaptive inference. The most valuable advances will demonstrate reliable performance across diverse real deployments, with transparent failure analysis and practical update mechanisms.

Practical implication:

Future work may evaluate federated updates, self-supervised representations, explainable alert summaries, privacy-preserving telemetry, and energy-aware adaptive inference. A most valuable advances will demonstrate reliable performance across diverse real deployments, with transparent failure analysis and practical update mechanisms. Accordingly, an implementation study must document its assumptions, identify failure modes, and connect every design decision to a measurable deployment requirement.



Summary:

This article has presented a conceptual lightweight deep learning framework for detecting cyberattacks in Internet of Things networks in near real time. Its core design places flow aggregation and compact inference at an edge gateway to reduce the need for continuous raw-packet transfer. The architecture combines careful feature engineering, a small convolutional classifier, calibrated decision thresholds, and graduated operational response.The review emphasizes that cybersecurity utility cannot be inferred from classification accuracy alone. Minority-class recall, false alarms, memory consumption, power use, end-to-end delay, and generalization across devices must be evaluated together. Data leakage and unrealistic random splits can make intrusion detectors look substantially stronger than they are in practice.The framework is a research proposal, not a claim of implemented or experimentally verified performance. Future validation should compare strong baselines using public and locally captured datasets, measure resource consumption on real edge hardware, and publish reproducible evaluation materials. This distinction is necessary for transparent scholarly reporting. lightweight neural inference is a promising component of layered IoT defense, provided that models are monitored, retrained carefully, audited for distribution shift, and integrated with human oversight. Sustainable deployment depends as much on sound systems engineering and threat modeling as on network architecture.

References

- Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). Kitsune: An ensemble of autoencoders for online network intrusion detection. NDSS Symposium. https://doi.org/10.14722/ndss.2018.23204
- Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., & Elovici, Y. (2018). N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Computing, 17(3), 12–22.
- Koroniotis, N., Moustafa, N., Sitnikova, E., & Turnbull, B. (2019). Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. Future Generation Computer Systems, 100, 779–796.
- Ton, T. D., & colleagues. (2023). CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment. Sensors, 23(13), 5941.

- Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. Journal of Information Security and Applications, 50, 102419.
- Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE Access, 7, 41525–41550.
- Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems. Military Communications and Information Systems Conference.
- Sarker, I. H. (2021). Deep cybersecurity: A comprehensive overview from neural network and deep learning perspective. SN Computer Science, 2, 154.
- 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.
- Howard, A. G., et al. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.
- Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv:1503.02531.
- Jacob, B., et al. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proceedings of CVPR, 2704–2713.