A Lightweight Hybrid Cnn-Lstm Intrusion Detection System for Resource-Constrained Iot Networks

Dr.T.Prabakar

Assistant Professor,Department of Computer Science,SRM Arts and Science College, Kattankulathur-603203, Tamil Nadu, India

DOI: https://doi.org/10.51244/IJRSI.2026.1306000531

Received: 04 July 2026; Accepted: 09 July 2026; Published: 24 July 2026

ABSTRACT

IoT devices keep multiplying, and so do the ways attackers can get into them. Most of these devices simply don't have the CPU, RAM, or power budget to run the deep learning-based intrusion detection systems (IDS) that perform best in the research literature, which creates a real gap between what's accurate and what's actually deployable. In this paper we build a hybrid CNN-LSTM classifier and, instead of shrinking it after the fact, we design it to be small from the start: depthwise-separable convolutions handle the spatial side, a deliberately narrow LSTM layer picks up temporal patterns across flow sequences, and the whole thing goes through structured pruning and post-training quantization before it ever gets tested. We ran it against TON_IoT and IoTID20, two datasets built from different testbeds with different attack mixes, and stacked it up against Random Forest, SVM, and plain (uncompressed) CNN and LSTM models. The compressed hybrid model came out ahead on both datasets - 98.6% accuracy on TON_IoT, 97.9% on IoTID20 - while ending up about 71% smaller and 58% faster at inference than its own uncompressed version. So it seems like joint design (architecture and compression together, not compression bolted on later) is a workable path to models that are both accurate and small enough to actually run at the edge. We close with a look at the accuracy/latency/energy trade-offs we hit along the way, and where this could go next - federated learning, online updates, that kind of thing.

Keywords: Intrusion Detection System, Internet of Things, Deep Learning, CNN-LSTM, Model Compression, Edge Security

INTRODUCTION

It's easy to forget how fast IoT has grown. What started as a research curiosity is now a huge, messy web of sensors, actuators, and embedded controllers running smart homes, factory floors, hospital monitors, and traffic systems. Security hasn't kept pace. Manufacturers are working under tight cost, power, and size budgets, and security controls - real authentication, encrypted firmware updates, any kind of on-device monitoring - are often the first things cut. That's how you end up with a sprawling, uneven attack surface, one that's already been exploited plenty of times, most visibly by botnets that hijack cheap, poorly secured devices and point them at targets in massive DDoS waves.

Network IDS has been around for a long time, and deep learning has become the default tool people reach for when adapting it, mainly because it can pull useful features straight out of raw or lightly processed traffic instead of relying on someone hand-writing signatures. CNNs are good at picking up local spatial patterns in flow-based feature vectors; LSTMs handle the sequential, time-dependent side of network sessions reasonably well. The broader idea that layered models can learn representations directly from data, rather than needing them engineered by hand, goes back to foundational deep learning work and it didn't take long for that idea to show up in network security research, with early studies showing deep models beating traditional classifiers on standard intrusion benchmarks .

The problem is that most of this progress happens on hardware that looks nothing like an actual IoT device. Surveys of the field point out that high-accuracy architectures tend to get designed and tested on server-class

machines, and that getting them onto the microcontrollers and single-board computers that make up real IoT deployments takes deliberate attention to size, memory footprint, and latency - not just accuracy (Gaber et al.). Some groups have tried to solve this with hardware, using edge TPUs and similar accelerators to squeeze more throughput out of constrained devices; that works, but not every IoT device ships with (or can afford) a dedicated accelerator. Other groups have gone the algorithmic route instead - pairing lightweight feature selection with recurrent models to cut input dimensionality, or applying quantization after training to shrink a model that's already been built.

That's the gap we're trying to close here: a hybrid CNN-LSTM model built for constrained hardware from day one, not compressed as an afterthought once it's already too big. We fold architectural choices - depthwise-separable convolutions, a deliberately small LSTM - together with post-training quantization into one design pipeline. Basically, we wanted to know whether a lightweight architecture designed this way could keep most of the accuracy benefit you get from combining spatial and temporal features, while still fitting the size and latency budget of typical IoT edge hardware. We test this on two separate, realistic IoT traffic datasets and compare against both classical ML and standard deep learning baselines.

Related Work

You can roughly sort IoT intrusion detection work into three buckets: classical statistical/shallow learning, deep learning, and, lately, distributed or federated learning aimed at the data-locality and privacy issues that come with IoT deployments.

Random Forests, SVMs, k-NN - these classical methods show up a lot in IoT IDS work, mostly because they're cheap to run and easy to interpret. The catch is they lean on hand-engineered features and tend to hit a ceiling once the dataset has a wide enough mix of attack types. Deep learning was the obvious next step, and early work showed that automatically learned representations could beat manually engineered ones on standard benchmarks. From there, people started tailoring recurrent and hybrid designs specifically to IoT traffic. One example: an algorithm for pulling out the important features from IoTID20, paired with an LSTM classifier, that did better at identifying attack types than plain feed-forward models (Jeyanthi & Indrani). In a similar vein, an LSTM autoencoder has been used to squeeze spatio-temporal traffic patterns into a compact latent representation before handing it off to a classifier - cutting dimensionality without throwing away the useful signal (Popoola et al.).

Separately, there's been a steady push toward dealing with IoT's hardware limits directly. One study looked at running deep IDS models on Google Edge TPUs and found that dedicated accelerators can meaningfully boost throughput over general-purpose IoT chips - useful, but it only helps if your device actually has that hardware, which most legacy or budget devices don't. Others have skipped the hardware angle entirely and gone after algorithmic efficiency instead, combining dynamic quantization with compact network designs so the model can run on an ordinary microcontroller with no extra accelerator needed. Surveys covering this space keep coming back to the same point: lightweight deployment lags well behind raw accuracy as a research priority, and what's really needed is architectures that treat accuracy, size, and latency as joint goals from the start, not compression tacked on at the end (Gaber et al.).

The third direction, and one that's picked up a lot of momentum recently, is federated learning for intrusion detection. The logic here is straightforward: IoT traffic is naturally spread across a lot of devices and gateways, and shipping all that raw traffic to a central server for training costs bandwidth and raises privacy questions. Federated IDS frameworks train one shared model across distributed clients without moving raw data off-device, and get detection performance close to what centralized training would give you (Li et al.). Other work has dug into problems specific to this setting - poisoning attacks aimed at personalized federated IDS, and defenses against them, plus approaches that pair federated aggregation with anomaly detectors built to handle the statistical differences between client devices (Idrissi et al.). There's also been work on cluster-based federated setups, grouping devices by traffic profile before aggregating, which helps convergence when the data across clients isn't independent or identically distributed.

Where this paper differs from all of that: first, we don't train a full-sized hybrid model and then compress it as a separate step afterward - the architecture and the quantization pipeline get designed together, with size and latency treated as real objectives from day one rather than an afterthought. Second, we test the result on two datasets from genuinely different environments and attack taxonomies, which lets us check whether the efficiency gains actually hold up beyond a single benchmark rather than being a quirk of one dataset.

Proposed Methodology

The system breaks down into four stages: traffic pre-processing and feature extraction, the hybrid CNN-LSTM classifier itself, model compression, and an inference pipeline meant for edge deployment.

Feature Extraction:

We aggregate raw packet captures into bidirectional flows and compute a fixed-length feature vector for each one - packet and byte counts, inter-arrival time stats, protocol and flag distributions, flow duration, basically the standard flow-based feature set used across these datasets. Categorical fields get one-hot encoded; continuous fields get normalized (zero mean, unit variance) using stats from the training split only, so nothing about the test set leaks in early. Each flow also gets tagged with a short window of the k flows before it from the same source-destination pair, which gives us a sequence input for the temporal side of the model.

Hybrid CNN-LSTM Architecture:

The classifier starts with two 1D convolutional blocks over the feature dimension, each one a depthwise-separable convolution followed by batch norm and ReLU. Depthwise-separable convolutions split a standard convolution into a per-channel spatial pass and a pointwise pass, which cuts down parameters and multiply-accumulate ops a lot without giving up much representational power. That output feeds into a single LSTM layer - and we kept the hidden size smaller than what you typically see in the literature, since the goal was to summarize temporal dependencies without dragging in the parameter overhead that stacked or bidirectional LSTMs bring. The LSTM's final hidden state goes through a fully connected layer with dropout, then a softmax layer that outputs a probability over benign vs. attack classes.

Model Compression:

Once the hybrid model has converged on the training data, we run post-training dynamic quantization - 32-bit weights in the FC and conv layers become 8-bit integers, while activations stay in mixed precision. Alongside that, we do structured pruning: any convolutional filter whose contribution to the output (measured by L1 norm of its weights) falls below a threshold set on a held-out validation split gets removed. After pruning and quantizing, the model gets a few extra fine-tuning epochs to claw back whatever accuracy the compression cost it.

Inference Pipeline:

During inference, incoming traffic gets grouped into flows and feature vectors on the fly, buffered into fixed-length sequences, then run through the compressed model. The whole thing runs on-device - no raw traffic gets shipped to a central server for classification, which helps both latency and privacy. Start to finish: capture raw traffic, aggregate into flows and normalize, buffer into sequences, run the compressed CNN-LSTM, then threshold the output into benign or attack and raise an alert if it's the latter.

Experimental Setup

Datasets:

We used two public IoT intrusion detection datasets. TON_IoT comes from a realistic testbed mixing IoT and industrial IoT sensors, OS-level data, and network traffic, and covers a range of attacks - DoS, DDoS,

ransomware, backdoor, reconnaissance (Alsaedi et al.). IoTID20 comes from a smart home testbed and includes normal traffic plus Mirai-style botnet activity, man-in-the-middle attacks, and scanning. Picking two datasets from genuinely different testbeds and attack taxonomies gives us a decent check on whether the gains we see are specific to one setup or actually generalize.

Pre-processing:

For both datasets we dropped duplicate and corrupted records, one-hot encoded categorical features, and normalized continuous features using training-set stats only. Intrusion datasets are almost always imbalanced - benign traffic dwarfs most attack categories - so we balanced the training split with a mix of random undersampling on the majority class and synthetic oversampling on minority attack classes. Each dataset got split 70/15/15 into train/validation/test, stratified by label.

Baselines:

We compared the proposed hybrid CNN-LSTM against four baselines: Random Forest (200 trees), an SVM with an RBF kernel, a plain CNN (same feature extraction, no LSTM, no compression), and a plain LSTM (same sequence inputs, no conv front-end, no compression). All the deep learning models were trained with Adam, learning rate 1e-3, batch size 256, early stopping on validation loss, capped at 100 epochs.

Evaluation Metrics:

We report accuracy, precision, recall, and F1 (macro-averaged across attack classes, given the imbalance). For deployability we also track model size in MB, average per-flow inference latency on a Raspberry Pi 4 Model B (a reasonable stand-in for mid-range IoT gateway hardware), and peak memory usage during inference.

RESULTS AND DISCUSSION

Table 1 lays out detection performance on both test sets. The compressed hybrid model gets the best F1 on both datasets, edging out the uncompressed plain CNN and plain LSTM baselines by a small margin and beating the classical baselines by a much bigger one. That classical-vs-deep gap is widest on IoTID20, which has some attack categories with subtler statistical fingerprints than the coarser ones in TON_IoT - not too surprising, honestly.

Table 1. Detection performance on TON_IoT and IoTID20 test sets.

Model	TON_IoT Acc.	TON_IoT F1	IoTID20 Acc.	IoTID20 F1
Random Forest	94.2%	0.931	92.8%	0.918
SVM (RBF)	92.7%	0.914	91.3%	0.902
Plain CNN	97.8%	0.973	96.9%	0.964
Plain LSTM	97.5%	0.969	96.5%	0.960
Proposed CNN-LSTM (compressed)	98.6%	0.982	97.9%	0.974

Table 2 has the deployability numbers. Compression knocks about 71% off model size and 58% off average inference latency compared to the uncompressed hybrid model, and the accuracy cost of that compression is under 0.4 percentage points on both datasets. So most of what pruning and quantization removed wasn't actually load-bearing for the decision boundary the model had learned.

Table 2. Model size, inference latency, and peak memory on Raspberry Pi 4.

Model	Size (MB)	Latency (ms/flow)	Peak Memory (MB)
Uncompressed Hybrid CNN-LSTM	4.8	12.4	61
Proposed Compressed CNN-LSTM	1.4	5.2	24

That's basically the core argument of this paper: design the architecture and the compression together, and a hybrid spatial-temporal model can hang on to most of its accuracy edge while getting genuinely deployable on-device. Structured pruning did more of the heavy lifting on latency than quantization did - pulling out whole filters cuts the actual operation count, where quantization mostly just shrinks precision - while quantization was the bigger factor in shrinking the file size on disk. We needed both; neither one alone got us gains on both fronts.

One thing that didn't fully go away: rare attack categories - some of the reconnaissance sub-types in TON_IoT, for instance - still scored lower than the common ones like DoS, even after balancing the training set. That tracks with what other people have found - resampling alone doesn't really solve class imbalance in intrusion data. Cost-sensitive loss functions or few-shot approaches seem like the more promising fix, and that's probably where we'd look next if we kept pushing on this.

Conclusion and Future Work

What we set out to build was a lightweight hybrid CNN-LSTM IDS that's actually deployable on constrained IoT hardware, not just accurate on paper. Putting depthwise-separable convolutions, a compact LSTM, structured pruning, and post-training quantization into one co-designed pipeline got us there: accuracy that's competitive with (and in our tests, slightly ahead of) uncompressed hybrid and single-branch baselines, plus a model that's about 71% smaller and 58% faster at inference. And this held up across two datasets from different environments with different attack mixes, which suggests it's not just a one-dataset fluke.

There's a few directions we'd want to take this next. Federated learning is the obvious one - letting multiple IoT gateways improve a shared model together without centralizing raw traffic, building on the federated IDS work that's already out there, while keeping the efficiency wins we got here. Second, the rare-attack-category problem is still open, and cost-sensitive or few-shot learning components seem like the more promising fix than more resampling. Third, we only tested on a Raspberry Pi 4 - it'd be worth checking how this holds up on actual microcontroller-class hardware with tighter memory limits, since that's a real step down from what we used.

REFERENCES

1. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015.
2. A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in Proc. 9th EAI Int. Conf. Bio-inspired Information and Communications Technologies (BIONETICS), 2016, pp. 21-26.
3. Z. Li, W. Fang, C. Zhu, G. Song, and W. Zhang, "Toward deep learning based intrusion detection system: A survey," in Proc. 2024 6th Int. Conf. Big Data Engineering (BDE), 2024.
4. T. Gaber, J. B. Awotunde, M. Torky, S. A. Ajagbe, M. Hammoudeh, and W. Li, "Metaverse-IDS: Deep learning-based intrusion detection system for Metaverse-IoT networks," Internet of Things, vol. 24, p. 100977, 2023.
5. W. Fang, C. Zhu, and W. Zhang, "Toward secure and lightweight data transmission for cloud-edge-terminal collaboration in Artificial Intelligence of Things," IEEE Internet of Things Journal, vol. 11, no. 1, pp. 105-113, 2024.
6. S. Hosseininoorbin et al., "A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization," PeerJ Computer Science, vol. 9, p. e1569, 2023.
7. N. Jeyanthi and B. Indrani, "ACAAS: An efficient feature extraction and recurrent neural network based intrusion detection approach for IoTID20," 2023.
8. S. I. Popoola, B. Adebisi, M. Hammoudeh, G. Gui, and H. Gacanin, "Hybrid deep learning for botnet attack detection in the internet-of-things networks," IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4944-4956, 2020.

9. A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems," IEEE Access, vol. 8, pp. 165130-165150, 2020.
10. J. Li, X. Tong, J. Liu, and L. Cheng, "An efficient federated learning system for network intrusion detection," IEEE Systems Journal, vol. 17, no. 2, pp. 2455-2464, 2023.
11. M. J. Idrissi, H. Alami, A. El Mahdaouy, A. El Mekki, S. Oualil, Z. Yartaoui, and I. Berrada, "Fed-anids: Federated learning for anomaly-based network intrusion detection systems," Expert Systems with Applications, vol. 234, p. 121000, 2023.
12. T. T. Thein, Y. Shiraishi, and M. Morii, "Personalized federated learning-based intrusion detection system: Poisoning attack and defense," Future Generation Computer Systems, vol. 153, pp. 182-192, 2024.