Evaluating Lightweight SLMs for Cloud-Free Edge Sensing

Wian Gertenbach[1] , Herman Myburgh[2] and Allan de Freitas[2]
[1]Council for Scientific and Industrial Research, Pretoria, South Africa
[2]University of Pretoria, South Africa

wgertenbach@csir.co.za

Abstract: The integration of Large Language Models (LLMs) and Industrial Internet of Things (IIoT) is an evolving field and has shown benefits in anomaly and fault detection as well as sensor fusion. While beneficial across various applications, standard deployments commonly rely on cloud servers that incur high costs, latency, and privacy concerns. Deploying capable LLMs directly on constrained edge devices remains a key challenge, as these devices operate under strict energy and computational limitations. Current research explores the IIoT and LLM integration through model optimisation and various deployment strategies. This paper presents a selection methodology of lightweight, fine-tuned Small Language Models (SLMs) for agricultural sensor data interpretation on resource-limited hardware. Three candidate models of similar sizes, including Llama3.2 (3B), Phi4-mini (3.8B), and Qwen3.5 (4B), are evaluated on real-world agricultural sensor data. The models are benchmarked across accuracy, precision and recall, inference latency and memory footprint on a Radxa Cubie A7A using zero-shot prompting, a technique in which models perform tasks without prior examples or specialised training. The models are then fine-tuned using Parameter-Efficient Fine-Tuning (PEFT) via Low-Rank Adaptation (LoRA) on agricultural sensor datasets. This method freezes a model's original weights and trains only a compact adapter module, significantly reducing computational overhead during training and preserving performance. The fine-tuned models are then compared against the results from the untuned models to determine the most suitable SLM for edge deployment and further research. This work contributes to a reproducible model selection and fine-tuning methodology for adapting open-weight SLMs to agricultural sensing without the reliance on cloud infrastructure, serving as a foundational step toward a fully decentralised, collaborative framework of fine-tuned edge SLMs for IoT systems.

Keywords: Large language models, Edge computing, Agricultural internet of things, Parameter-efficient fine-tuning, Low-rank adaptation

1. Introduction

The integration of artificial intelligence (AI) with IIoT becomes more promising as the capabilities of LLMs evolve. LLMs with direct access to sensor data can provide IoT applications with better system insights, such as sensor fault detection and environmental anomaly classification (Kök, Demirci and Özdemir, 2024). An LLM that enables interaction with the system in natural language can potentially assist operators with better decision making, as they possess the ability to accurately interpret sensor data to make recommendations based on its internal knowledge (Pujolle, Azzouni and Nguyen, 2024; Lamaakal *et al.*, 2025). Various studies have shown the benefits of LLM assistance at the edge, including malware detection, predictive maintenance in smart manufacturing and real-time threat prediction and response in smart energy networks (Doğruluk and Açıkgöz, 2025; Khan, 2025; Rondanini *et al.*, 2025). Integrating LLMs into IIoT systems remains challenging due to models' high computing demands. Typical solutions are mostly dependent on a central, powerful cloud server, which can incur high costs, high latency and pose data privacy concerns.

This paper focuses on the first part of a broader research initiative to investigate the possibility of a completely decentralised architecture. In this architecture, specialised small models, deployed across multiple edge devices, communicate their findings to each other to correctly classify anomalies in agricultural IoT (AIoT) sensor data. The paper describes a suitable model selection methodology by evaluating the classification ability of open-weight SLMs on AIoT sensor data.

2. Overview of Current Literature

The literature reviewed for this study includes various LLM optimisation and edge deployment strategies. Notable optimisation methods include model compression, model specialisation through LoRA fine-tuning and improving hardware acceleration with Field-Programmable Gate Arrays (FPGAs). Deployment strategies include edge-cloud collaboration, model distribution architectures and multiple whole model deployments.

2.1 Model Compression

Model compression is achieved through quantization, pruning or knowledge distillation. These techniques are used to shrink test models to a more manageable size, but still efficient enough for real-time use on edge devices (Agrawal, Kumar and Lnu, 2025). With quantization and pruning, the model's memory and computational costs can be reduced to increase inference time, although at the cost of accuracy. Pruning is generally not suitable for edge LLMs, and quantization is recommended as the better option. These models often perform similar to their

unquantized originals, given that they are fine-tuned on domain-specific data (Qin *et al.*, 2025). Open-weight models on Huggingface contain various quantized versions, ranging from 8-bit to 4-bit compression. Qin et al. developed a framework for model compression based on its application in edge computing and recommend the best fine-tuning methods for different task complexities (Qin *et al.*, 2025).

2.2 Edge-cloud Collaboration

In edge-cloud collaboration, locally deployed models are paired with a cloud model. A central coordinator algorithmically determines whether to offload heavier computational workloads to the cloud or leave simpler inference tasks for the edge LLM to execute (Hu *et al.*, 2025). This configuration optimises latency, energy and quality of service. By strategically using the edge model, latency is improved because tasks are executed locally. Remote computing is only invoked when task complexity crosses a specific threshold. Other architectures involve the use of an AI router to evaluate a user's input and predict whether an edge or cloud model will produce the best result (Kim *et al.*, 2025). Compared to running a model in the cloud permanently, these deployments demonstrate a significant cost and latency reduction, with a minimal negative impact on answer quality. The CoSense-LLM framework also demonstrates an efficient edge-cloud collaboration system (Akgul *et al.*, 2025). In this configuration, specialised agents are deployed on the edge and aim to provide semantic outputs of collected sensor data based on its internal knowledge base. The framework also relies on a central coordinator to determine if the locally deployed agents are suitable for decision-making or whether it should be offloaded to a more powerful cloud model. Edge-cloud collaboration, though effective, still relies primarily on the cloud server to perform the most demanding tasks. This raises the question whether locally deployed models can still be effective without relying on a more powerful cloud counterpart.

2.3 Model Distribution Architectures

This deployment strategy relies on partitioning a high-parameter LLM and distributing it across several devices. The devices work together to collectively run the model, as demonstrated by the EdgeShard framework (Zhang *et al.*, 2025). The collaborative computing in this framework relies on geographically distributed hardware which includes edge devices and cloud servers. Other research on this deployment strategy mentions that splitting one model across hardware only works effectively when the devices are connected physically by high-bandwidth cables (Ye *et al.*, 2025). This complicates deploying the model over a geographically distributed set of hardware, as physical connections are not always feasible.

2.4 Multiple Whole Model Deployments

Multiple low-parameter models can be deployed on separate edge devices. This configuration aims to completely remove a cloud counterpart or coordinator and rely solely on the outputs and collaboration of the individual edge models. In an LLM-assisted malware detection application, the authors propose an architecture in which each edge device contains its own small model, fine-tuned on the type of attack that the device is most exposed to (Rondanini *et al.*, 2025). However, no experiment in their paper records multi-device collaboration or even that the proposed architecture improves malware classification. This shows that, while deploying LLMs on the edge is possible, a collaboration system is needed to improve classification of domain-specific sensor data. In another paper, the CASIT framework demonstrates collaboration between agents deployed on the edge with a coordinator that orchestrates operations and synthesises reports from all agents into a concise summary (Zhong *et al.*, 2024). The authors specifically mention that this framework can work with lightweight LLMs that act as agents on edge devices, however their experiments used an application programming interface (API) for ChatGLM, which is cloud-based.

2.5 Hardware Acceleration

Hardware acceleration is achieved through the custom development of FPGA accelerators (Haris *et al.*, 2024). They propose a framework, SECDA-LLM, which allows a hardware designer to write an accelerator once, in a hardware description language, and have llama.cpp call it during inference. This offloads computation from the CPU to a GGML operator, driven by the FPGA.

2.6 Model Specialisation

Specialisation involves fine-tuning models on domain-specific data. In the same paper by Rondanini et al., the authors describe how specialisation through LoRA fine-tuning improves individual models' ability to classify malware (Rondanini *et al.*, 2025). A specialised language modes of roughly 100 million parameters, deployed on a single-board computer, has also been demonstrated to accurately classify serialised sensor readings (Kandala, Medaranga and Varshney, 2024). Qin et al. also showed the benefits of using LoRA fine-tuned, quantised LLMs

for downstream classification tasks (Qin *et al.*, 2025). They claim that LoRA is the best PEFT method for fine-tuning small models to execute easy classification tasks.

3. Research Methodology

Considering the different deployment strategies, this study focuses on the selection and evaluation of an appropriate SLM to use for an AIoT application. The research methodology is split into four phases. Phase 1A involves the identification of candidate SLMs. Three open-weight models are chosen based on criteria such as parameter count, reasoning ability, its performance relative to its size, inference engine support and quantization formats. A public dataset of real sensor readings is chosen in Phase 1B, and the data is processed to produce a merged, labelled dataset where anomalies are tagged. The labelled dataset is then split using a stratified random split into a test dataset and training dataset. In Phase 2, the three candidate models are evaluated using the test dataset. The Radxa Cubie A7A is used as the hardware module to run the evaluation on, and it represents the edge node. The models are evaluated sequentially, and in each case the model's answer and performance metrics, including precision and recall (weighted F1 score), latency, memory (RAM) usage and tokens per second are recorded as a baseline. Phase 3 is the fine-tuning process using LoRA. All models are fine-tuned using the same training dataset, consisting of prompt-response groups, and the training conditions are kept identical for all three training sessions. The fine-tuned models are then evaluated in Phase 4 using the same test dataset and hardware as in Phase 2. The same metrics are recorded and compared to the zero-shot baseline.

3.1 Phase 1: Preparation

3.1.1 Model identification

The challenge, as mentioned previously, is that edge hardware is often resource constrained. LLMs require heavy compute power, and traditional edge hardware is not sufficient. However, LLM families include lower-parameter versions, requiring less powerful hardware and can run solely on the CPU. The trade-off is that the model is "less clever" than its high-parameter sibling. Three candidate models in the 3 to 4 billion parameter range were selected for comparison, including Phi4-mini (3.8B, Microsoft), Llama3.2 (3B, Meta) and Qwen3.5 (4B, Alibaba). For simpler to medium-complexity tasks, 2 to 4 billion parameter models are ideal, as larger models waste resources, while even smaller models risk insufficient reasoning capacity (Qin *et al.*, 2025). All three models are instruction-tuned, support efficient quantised inference via llama.cpp, and represent three architecturally distinct model families. All the models all support quantization using the K-quant format, which provides the best balance of inference speed and output quality on ARM CPUs.

3.1.2 Dataset selection

AIoT has been selected as the target domain for this study. It is a well-established domain and contains data from different sensors and operating environments. Each environment contains multiple sensor types, such as temperature, pressure and humidity, which is ideal for testing the LLM's ability to interpret multi-sensor readings.

The selected dataset comes from a study on tomato cultivation (Belli *et al.*, 2024). The dataset was produced using a LoRaWAN-connected sensor network on a farm in Parma, Italy, during the 2023 summer months from 29 June to 13 September. Soil measurements from three lines of tomato plantations were recorded, and the atmospheric conditions, which apply uniformly across the three lines, were also recorded. A third set of data was collected, containing measurements of the total water through irrigation of each line over time. In each case, the data was sampled roughly every ten minutes. The dataset is publicly available, and measurements are distributed across three comma-separated value (CSV) files corresponding to the three distinct categories mentioned:

1. The soil data includes moisture, electrical conductivity, temperature and the edge device's battery readings.
2. The environmental data includes air humidity, carbon dioxide levels, pressure, temperature, and the edge device's battery readings.
3. The water meter data includes the total water volume measured over the period.

The water meter data was ignored for the LLM testing and fine-tuning, as it merely reflects the total volume of water received by the crops through irrigation and does not contain any anomalies. The edge devices' battery levels were also ignored as these were reported as NULL across all measurements, indicating that they might have been powered from an external source.

The source files were loaded into a custom Python script to calculate the maximum, minimum, mean, standard deviation for each sensor reading, as well as missing values in each column. Based on the data distribution observed in the datasets and general guidelines on growing conditions for tomato cultivation, normal operating ranges were defined (Shamshiri *et al.*, 2018). Classifying anomalies are based on the thresholds defined in Table 1.

Table 1: Defined normal operating ranges for tomato cultivation

Sensor	Unit	Lower bound	Upper bound
CO_2	ppm	400	800
Air humidity	%RH	60	90
Pressure	hPa	995	1020
Air temperature	°C	16	35
Soil moisture	%RH	15	55
Soil temperature	°C	19	32
Electrical conductivity	µS/cm	50	1200

3.1.3 Data processing and merging

The environmental and soil datasets were merged into one unified set. The merge firstly required a timestamp conversion to human-readable datetime values, as the original values were stored as Unix milliseconds. Each file contained data obtained from different sensor nodes operating on different sampling schedules, and thus the timestamps were not synchronised. A nearest-time merge was performed using a 30-minute tolerance window, and the soil readings were matched with its closest environmental readings. The merge integrity was verified by randomly selecting five rows in the merged set against their original source rows. All the environmental readings matched the original exactly, except for a small discrepancy of less than 0.2 units in some rows.

Anomaly labels were added to the merged dataset. A custom Python script evaluated each row and assigned the following labels, based on the defined thresholds in Table 1:

- Normal: All sensor readings fall within the normal defined range.
- Single sensor: One sensor reading falls outside the normal defined range.
- Multi-sensor: Two or more sensor readings fall outside the normal defined range.

In each case, the anomaly count was added to the row, as well as the appropriate explanation for labelling the readings as anomalous.

The resultant anomaly distribution was 71% normal, 17% single sensor, 12% multi-sensor. The majority of the readings are normal, which means the evaluated LLM will not be exposed to a large number of single and multi-sensor anomalies. However, the natural distribution was preserved instead of resampled artificially to provide a more honest assessment of its performance on real-world data.

3.1.4 Evaluation and fine-tuning dataset preparation

The labelled dataset of 32586 rows was split into two sets: one for the evaluation, and another for fine-tuning. A stratified 80-20 random split was applied, resulting in an evaluation set of 6518 rows and a fine-tuning set of 26068 rows. The stratification ensures that the label proportions are preserved in both splits, preventing the concentration of anomalous or normal readings in one or the other.

For the evaluation set, a 100-row stratified sample was created. This is a 98.47% reduction from its original size, but necessary in order for the evaluation to execute on the Radxa Cubie A7A within a reasonable time, since its CPU becomes a bottleneck for inference speed.

The fine-tuning set was used to create a JSON file with 1000 prompt-response groups. Each group consists of a fixed instruction, a variable input of formatted sensor data, a variable expected output, as well as the anomaly label associated with the input data. This is the standard scheme for instruction fine-tuning of pre-trained LLMs (Zamarin *et al.*, 2024). The system instruction used across all groups is as follows:

> *You are an agricultural IoT monitoring assistant deployed on an edge device. Analyse the following sensor readings from a tomato farm and provide a concise assessment. State whether conditions are*

normal or anomalous, identify which sensors are affected, state the normal range for each affected sensor, and suggest a likely cause or recommended action.

The responses were generated programmatically using templates based on the normal defined operating ranges for tomato cultivation. This strict templating method ensures that the response matches the associated label exactly, eliminating the risk of response inconsistency. During fine-tuning, the model learns the reasoning pattern demonstrated by the prompt-response groups.

3.2 Phase 2: Zero-shot Evaluation

In zero-shot inference, an LLM is tasked with recognising and categorising concepts without prior knowledge of those concepts beforehand. The three selected models were each subjected to a zero-shot evaluation to determine their ability to recognise anomalies in the tomato cultivation data. The results set the baseline of what the model is capable of and will be compared with the results from the fine-tuned models to measure model improvement and to select the best model for further study.

The models were downloaded from the Bartowski repository on Huggingface in GGUF format. All the models were 4-bit quantized (Q4) versions of the original. 4-bit quantization was chosen as it provides the best balance between lower video memory (VRAM) usage and accuracy loss (Demper, 2026). Inference was served via a llama.cpp server running in a Docker container on the Radxa Cubie A7A. The board's CPU consists of a combination of two Cortex-A76 @ 2.0GHz performance cores and six Cortex-A55 @ 1.8GHz efficiency cores. Furthermore, it has 16 GB LPDDR5 RAM running at up to 4800 MT/s.

The lightweight llama.cpp inference engine was chosen due to its high-performance (Demper, 2026). The CPU backend Docker image was used for the local inference server, with the model mounted to the container and the temperature set to 0.1. This value forces the model to be deterministic but with a small amount of flexibility in its creativity. A custom Python script executed the evaluation, starting with three warm-up inferences for the model to load data structures into CPU cache, allocate memory and initialise internal states. The model was then evaluated on the 100-row stratified sample drawn from the test set. Each inference was capped at 120 tokens, preventing the model from generating extremely long responses that would inflate latency measurements and make future comparisons inaccurate.

3.2.1 Zero-shot results

The results from the zero-shot evaluation are presented in Table 2. Llama3.2 achieved the highest weighted F1 score, as well as the lowest latency and RAM usage, making it the best performing model of the three. Qwen3.5 scored the worst in all categories, with the lowest F1 score and much higher latency and RAM consumption than the other models.

Table 2: Zero-Shot Results

Model	F1 (weighted)	Precision	Recall	Latency (s)	Tokens/s	RAM (MB)
Llama3.2	0.6686	0.7298	0.720	15.998	3.2	3156.1
Phi4 mini	0.5903	0.5607	0.680	39.016	2.6	3823.7
Qwen3.5	0.5896	0.5041	0.710	72.600	N/A*	5778.9

*Qwen3.5's zero-shot tokens-per-second readings were discarded from the results due to a server-side token count reporting anomaly in two data rows.

3.3 Phase 3: Model Fine Tuning

The models were fine-tuned using LoRA. In this case, the model is not trained from scratch, and rather than updating all model weights, LoRA freezes the original weights and introduces a trainable adapter module into the attention layers. This reduces computational overhead during fine-tuning, preserves model performance, and is thus a suitable method for fine-tuning in less powerful environments.

The models were trained under the exact same conditions and LoRA configuration, although Qwen3.5's configuration differed slightly due to its architecture. The same fine-tuning dataset with 1000 prompt-response groups was used in all three cases, with LoRA rank set to 16, which is the widely used middle-ground for fine-tuning 3 to 4B parameter models. Alpha was set to 32, two times the rank, and the models were trained over three epochs, as recommended in the literature (Qin *et al.*, 2025).

Fine-tuning was conducted on a Kaggle Notebook using an Nvidia T4 GPU with 4-bit medium quantization (QLoRA). This reduces the overall VRAM usage by quantizing the base model weights using training. The training loss was computed over the response tokens, and instruction and input tokens were masked to ensure that the model did not memorise the prompt structure.

The LoRA adapter weights were applied to the attention layers of each model. Table 3 shows the weight matrices (vector modules) of each model based on its architecture.

Table 3: SLM Attention Layer Modules

Model	Target Vector Modules
Llama3.2	Query, key, value, output projections
Phi4-mini	Query, key, value, output projections
Qwen3.5	Value, output, gate, up, down projections

Llama3.2 and Phi4-mini uses the standard Query, Key, Value (QKV) vectors, but Qwen3.5 uses QK-Norm (Query-Key Normalization) and applies it to the query and key vectors to prevent numerical overflow. Because these vectors were already normalized, the LoRA configuration avoided them and targeted gate, up and down projections instead.

The models were fine-tuned for 189 gradient steps across the three epochs. The training loss converged within one epoch for all models, with negligible improvements in epochs 2 and 3. This suggests that the model can learn its task from an even smaller set of domain-specific examples. After fine-tuning, the LoRA adapters were merged into the base model weights and converted to 4-bit medium quantized GGUF formats for the deployment on the A7A via llama.cpp. Qwen3.5 required a different approach, and the fine-tuned adapter was converted to GGUF format separately and loaded alongside the base model during inference time.

3.4 Phase 4: Evaluation of Fine-tuned Models

The fine-tuned models were evaluated using the exact methodology as described in Section 3.2, with the same 100-row stratified test dataset.

3.4.1 Fine-tuned model results

Fine-tuning increased performance across all models, as seen in Table 4. There was an overall 22.3% average F1 improvement relative to the zero-shot baseline. Qwen3.5 achieved the highest F1 score, followed by Llama3.2 and Phi4-mini respectively. Qwen3.5 reversed its last place in the zero-shot evaluation to first place with an overall improvement of 30.3%. Llama3.2 exhibited a precision and recall trade-off, with precision decreasing from 0.7298 to 0.6948 while recall increased from 0.720 to 0.830. The precision decrease can be attributed to the model's bias toward flagging anomalies, even in normal readings. This is not necessarily bad behaviour, as false alarms in an agricultural context are preferable to missed anomalies, which carries higher operational risk. Llama3.2's inference latency also increased quite drastically, possibly due to the larger merged GGUF file. The same applies to Phi4-mini, although the increase is less drastic than Llama3.2. Qwen3.5's latency was mostly unchanged, as the adapter was loaded in during inference time rather than merged into the base model to produce a larger file. However, Qwen3.5's tokens per second rate of 1.9 results in the longest inference time of the three models.

Table 4: Zero-Shot versus Fine-Tuned Evaluation Results

Model	Stage	F1	Precision	Recall	Latency (s)	RAM (MB)	Tokens/s
Llama3.2 (3B)	Zero-shot	0.6686	0.7298	0.720	15.998	3156.1	3.2
	Fine-tuned	0.7550	0.6948	0.830	40.032	11670.3 †	3.0
Phi4 mini (3.8B)	Zero-shot	0.5903	0.5607	0.680	39.016	3823.7	2.6
	Fine-tuned	0.7374	0.6779	0.810	55.714	4642.2	2.4
Qwen3.5 (4B)	Zero-shot	0.5896	0.5041	0.710	72.600	5778.9	N/A*
	Fine-tuned	0.7684	0.7297	0.830	72.136	5449.5	1.9

† Llama3.2's fine-tuned RAM figure may reflect a system-level measurement artefact during a high-load period on the Radxa Cubie A7A rather than true model footprint.

4. Discussion

The most significant finding is Qwen3.5's F1 score reversal from last to first place after fine-tuning. This shows that the fine-tuning process holds merit, as choosing a model based on the zero-shot evaluation could have discarded the best model before fine-tuning.

The results reveal an important accuracy-latency trade-off that must be considered when selecting a suitable model for edge deployment. Even though Qwen3.5 scored the highest accuracy, it is the slowest. Llama3.2 offers an accuracy of only 1.74% weaker than Qwen3.5 but performs better in terms of latency and tokens per second. This is an acceptable compromise for latency-sensitive systems with real-time fault response. However, for periodic monitoring in an agricultural application, where sensor readings are collected every several minutes, Qwen3.5's latency is acceptable.

Several limitations should be noticed, of which their motivations have been discussed in earlier sections. The evaluation sample of 100 rows is of modest size, and the anomaly distribution favours normal readings. The dataset originated from a tomato cultivation on a single farm in Italy and may not generalise to other crops and climates. Lastly, the A7A's power consumption was not measured during evaluation due to hardware constraints, limiting the sustainability assessment of each model.

5. Future Work

Several directions for future investigation are identified. Sub-3B parameter models, including Llama3.2 (1B), SmolLM2 (1.7B), and Qwen3.5 (0.8B), represent promising candidates for even more constrained edge deployments and will be evaluated in future work. These models may offer acceptable accuracy at significantly lower latency and memory consumption.

This study serves as the foundation for a broader initiative, where a fully decentralised, peer-to-peer collaboration framework for SLMs on AIoT systems is researched. The framework aims to enable multiple intelligent edge nodes, like multiple Radxa A7As with Llama3.2, to collaborate through a threshold-based query-response mechanism to achieve collective intelligence without relying on a central cloud server. A simulation and physical hardware validation on a multi-node testbed are planned for further research.

6. Conclusion

This paper investigated a methodology for the selection of fine-tuned SLMs for the interpretation of AIoT sensor data on a resource-constrained edge device. This is towards a fully collaborative decentralised system of intelligent edge nodes, which does not rely on a central cloud server. Three candidate SLMs were identified for the study based on their parameter count, inference engine support, performance and reasoning abilities. The models were evaluated across zero-shot and LoRA fine-tuned configurations on the Radxa A7A Cubie using a real-world tomato cultivation dataset.

The fine-tuning improved classification performance of all models on average by 22.3% compared to the zero-shot baseline. Qwen3.5 achieved the highest F1 score after fine-tuning, although it is also the slowest of the three. All three models converged within one training epoch, suggesting that fine-tuning can be achieved effectively with a small number of samples and lower compute power.

The results demonstrate a practical method to achieve cloud-free, on-device agricultural monitoring using fine-tuned SLMs, with Llama3.2 offering the best accuracy versus latency trade-off for latency-sensitive deployments and Qwen3.5 providing the highest classification accuracy for periodic monitoring applications.

Ethics Declaration: No ethical clearance was required for the completion of research activities mentioned in this paper.

AI Declaration: Generative artificial intelligence (AI) was solely used for synthesizing and summarising literature and not to fix grammatical errors or generate text for the manuscript. No tools were used to write original content. AI tools used to aid the literature synthesis process include Claude.

References

Agrawal, R., Kumar, H. and Lnu, S.R. (2025) "Efficient LLMs for Edge Devices: Pruning, Quantization, and Distillation Techniques," *2025 International Conference on Machine Learning and Autonomous Systems (ICMLAS). 2025 International Conference on Machine Learning and Autonomous Systems (ICMLAS)*, Prawet, Thailand: IEEE, pp. 1413–1418. Available at: https://doi.org/10.1109/ICMLAS64557.2025.10968787.

Akgul, H. *et al.* (2025) “CoSense-LLM: Semantics at the Edge with Cost- and Uncertainty-Aware Cloud-Edge Cooperation.” arXiv. Available at: https://doi.org/10.48550/ARXIV.2510.19670.

Belli, L. *et al.* (2024) “IoT-based Dataset of a Tomato Cultivation Under Different Irrigation Regimes.” Mendeley Data. Available at: https://doi.org/10.17632/35wh56287y.2.

Demper, M. (2026) “Run AI Models Locally with llama.cpp,” *GitHub*, 7 May. Available at: https://github.com/marcel-dempers/docker-development-youtube-series/tree/master/ai/models/llama-cpp (Accessed: January 6, 2026).

Doğruluk, E. and Açıkgöz, H. (2025) “Edge-Centric Federated Learning for LLMs in Smart Manufacturing: Architectures, Challenges, and Opportunities,” *2025 4th International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. *2025 4th International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, Tirupur, India: IEEE, pp. 1250–1256. Available at: https://doi.org/10.1109/ICIMIA67127.2025.11200757.

Haris, J. *et al.* (2024) “Designing Efficient LLM Accelerators for Edge Devices.” arXiv. Available at: https://doi.org/10.48550/ARXIV.2408.00462.

Hu, Y. *et al.* (2025) “A Cloud–Edge Collaborative Architecture for Multimodal LLM-Based Advanced Driver Assistance Systems in IoT Networks,” *IEEE Internet of Things Journal*, 12(10), pp. 13208–13221. Available at: https://doi.org/10.1109/JIOT.2024.3509628.

Kandala, S.V., Medaranga, P. and Varshney, A. (2024) “TinyLLM: A Framework for Training and Deploying Language Models at the Edge Computers.” arXiv. Available at: https://doi.org/10.48550/ARXIV.2412.15304.

Khan, M.A. (2025) “LLM-Driven Real-Time Threat Prediction and Response for Internet of Energy (IoE),” *IEEE Network*, pp. 1–8. Available at: https://doi.org/10.1109/MNET.2025.3612090.

Kim, M. *et al.* (2025) “Edge vs Cloud: How Do We Balance Cost, Latency, and Quality for Large Language Models Over 5G Networks?,” *2025 IEEE Wireless Communications and Networking Conference (WCNC)*. *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, Milan, Italy: IEEE, pp. 1–6. Available at: https://doi.org/10.1109/WCNC61545.2025.10978177.

Kök, İ., Demirci, O. and Özdemir, S. (2024) “When IoT Meet LLMs: Applications and Challenges,” *2024 IEEE International Conference on Big Data (BigData)*. *2024 IEEE International Conference on Big Data (BigData)*, Washington, DC, USA: IEEE, pp. 7075–7084. Available at: https://doi.org/10.1109/BigData62323.2024.10825187.

Lamaakal, I. *et al.* (2025) “Tiny Language Models for Automation and Control: Overview, Potential Applications, and Future Research Directions,” *Sensors*, 25(5), p. 1318. Available at: https://doi.org/10.3390/s25051318.

Pujolle, G., Azzouni, A. and Nguyen, T.M.T. (2024) “An Intelligent Edge Embedded Data Center,” *2024 IEEE 13th International Conference on Cloud Networking (CloudNet)*. *2024 IEEE 13th International Conference on Cloud Networking (CloudNet)*, Rio de Janeiro, Brazil: IEEE, pp. 1–5. Available at: https://doi.org/10.1109/CloudNet62863.2024.10815907.

Qin, R. *et al.* (2025) “Empirical Guidelines for Deploying LLMs onto Resource-constrained Edge Devices,” *ACM Transactions on Design Automation of Electronic Systems*, 30(5), pp. 1–58. Available at: https://doi.org/10.1145/3736721.

Rondanini, C. *et al.* (2025) “Malware Detection at the Edge with Lightweight LLMs: A Performance Evaluation,” *ACM Transactions on Internet Technology*, p. 3769681. Available at: https://doi.org/10.1145/3769681.

Shamshiri, R.R. *et al.* (2018) “Review of optimum temperature, humidity, and vapour pressure deficit for microclimate evaluation and control in greenhouse cultivation of tomato: a review,” *International Agrophysics*, 32(2), pp. 287–302. Available at: https://doi.org/10.1515/intag-2017-0005.

Ye, S. *et al.* (2025) “Resource-Efficient Collaborative Edge Transformer Inference With Hybrid Model Parallelism,” *IEEE Transactions on Mobile Computing*, 24(10), pp. 10945–10962. Available at: https://doi.org/10.1109/TMC.2025.3574695.

Zamarin, S. *et al.* (2024) “An introduction to preparing your own dataset for LLM training,” *Machine Learning*, 19 December. Available at: https://aws.amazon.com/blogs/machine-learning/an-introduction-to-preparing-your-own-dataset-for-llm-training/ (Accessed: June 3, 2026).

Zhang, M. *et al.* (2025) “EdgeShard: Efficient LLM Inference via Collaborative Edge Computing,” *IEEE Internet of Things Journal*, 12(10), pp. 13119–13131. Available at: https://doi.org/10.1109/JIOT.2024.3524255.

Zhong, N. *et al.* (2024) “CASIT: Collective Intelligent Agent System for Internet of Things,” *IEEE Internet of Things Journal*, 11(11), pp. 19646–19656. Available at: https://doi.org/10.1109/JIOT.2024.3366906.