Journal of Informatics and Web Engineering
Vol. 3 No. 3 (October 2024) eISSN: 2821-370X

Implementation of Lightweight Machine Learning Models for Real-time Text Classification on Resource-Constrained Devices

Marwah Zaid Mohammed Al-Helali[1], Naveen Palanichamy[2,*], K. Revathi[3*]
[1,2] Faculty of Computing & Informatics, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia.
[3] Department of Information Technology, SRM Valliammai Engineering College, Tamil Nadu, India.
**corresponding author: (p.naveen@mmu.edu.my; ORCiD: 0000-0003-4601-9770)*
**corresponding author: (neyadharshini@gmail.com; ORCiD: 0000-0002-8824-5285)*

Abstract - This paper addresses the growing need for implementing intelligent Natural Language Processing (NLP) systems on low-power, memory-limited devices such as Raspberry Pi, mobile phones, and IoT edge hardware. As edge computing and smart devices proliferate, there is an urgent need for more advanced NLP technology that does not require constant cloud access and is efficient in computing and provides results in real time. While deep learning and cloud-based models typically offer high text-classification accuracy and have demonstrated exceptional performance across a range of NLP tasks, they are often too resource-intensive for real-time deployment in constrained environments. To overcome these limitations, we explore a set of lightweight machine learning (ML) models—Multinomial Naive Bayes, Logistic Regression, and Decision Tree—to perform sentiment classification on a subset of the Amazon Reviews Polarity dataset. Following thorough data preprocessing and Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, two optimization techniques are employed: feature selection via Chi-Squared tests and simulated post-training quantization. Our experimental results show that resource consumption can be substantially reduced, with minimal accuracy loss, thereby demonstrating feasibility for edge-based text analytics and offline functionality. We provide a detailed comparative analysis that highlights how classical ML models remain viable in scenarios where modern deep learning architectures cannot be efficiently deployed.

Keywords—Lightweight, Machine Learning, Text Classification, Resource-Constrained, Sentiment Analysis

Received: 11 February 2025; Accepted: 25 May 2025; Published: 16 October 2025

This is an open access article under the CC BY-NC-ND 4.0 license.

1. INTRODUCTION

Modern computing ecosystems have evolved to rely more on resource-constrained devices, including mobile phones, low-power microcomputers, and different Internet of Things (IoT) edge nodes [1]. However, stringent constraints on

CPU power, memory, storage, and networking make it difficult to install standard ML models—often created for cloud-scale infrastructure—on such devices [2]. In NLP activities, where models such as transformers require substantial computational resources that are much above the capabilities of edge devices, these constraints are particularly important.

Lightweight ML models are crucial in this situation because they can operate locally with less resource usage and still produce accurate predictions. Applications that demand low latency, offline functionality, energy efficiency, and data privacy are the ones driving this need. For example, relying on cloud-based computation is impractical or perhaps impossible in disaster areas, rural locations with poor connectivity, or healthcare monitoring. By avoiding the delays or data breaches that come with online communication, lightweight models allow real-time inference right on-device.

While large neural architectures generally offer higher accuracy, their deployment on constrained hardware is not feasible [3]. As a result, the field of on-device text classification has gained momentum, with some researchers using model pruning, quantization, and knowledge distillation to compress deep neural networks without significantly sacrificing accuracy [4]. As a result, many applications require immediate feedback and secure processing of textual data [5]. Others have investigated using classical models with carefully designed features to preserve lightweight performance, such as SVM and Naive Bayes [6]. Furthermore, hybrid approaches that combine quick classifiers and rule-based filtering have demonstrated potential in time-sensitive applications [7]. Therefore, as potential substitutes, this work investigates the application of lightweight classical ML models, including Multinomial Naive Bayes, Logistic Regression, and Decision Tree. Predictive performance and computational economy are balanced in these models when they are optimized using methods like feature selection and quantization [8].

The structure of this paper is as follows section 1 provides an introduction and highlights the need for lightweight models on resource-constrained devices. Section 2 presents a review of related work in the field of text classification and model optimization. Section 3 details the methodology, including dataset preparation, model selection, and optimization techniques. Section 4 discusses the experimental results and evaluates the performance of the models. Finally, Section 5 concludes the paper and outlines potential directions for future research.

2. LITERATURE REVIEW

This section reviews prior research on deploying ML models in resource-limited environments, focusing on the nature of edge constraints and the strategies researchers have applied to reduce computational overhead. We also delve into text classification efforts in sentiment analysis, outlining key methods and data sources.

2.1 Resource Constraints in Edge Computing

Edge computing aims to process data as close to the source as possible, which improves latency and reduces bandwidth usage but also imposes severe restrictions on available hardware resources [6]. Studies underscore that while deep neural networks can yield excellent results, their memory footprint and compute demands are often incompatible with edge deployment [7]. Various lightweight techniques such as pruning and quantization have been explored, yet many revolve around neural architectures rather than classical ML [9]. Researchers have also explored model distillation [[10], hardware-aware Neural Architecture Search (NAS) [11], and dynamic computation graphs [12] to tailor networks to specific resource budgets. Additionally, federated learning frameworks have emerged to distribute model training across edge devices, reducing central processing but raising concerns around model complexity and data heterogeneity [13]. For real-time NLP tasks, latency constraints further limit the feasibility of large models, making compact architectures or classical approaches more attractive [14].

2.1.1 Classical ML Models vs. Deep Learning

Though deep learning has dominated tasks like computer vision and NLP, it may be over-engineered for simpler classification problems, especially when the target environment requires extremely efficient inference [15]. Classical algorithms—Multinomial Naive Bayes, Logistic Regression, and Decision Tree—offer smaller model sizes and faster training/inference times [8]. As a result, they serve as solid baselines for resource-constrained contexts, including embedded systems and IoT sensors [16]. Moreover, studies such as [17] demonstrate that classical models can

approach deep learning accuracy when paired with effective preprocessing and feature engineering. For instance, bag-of-words and TF-IDF representations combined with Logistic Regression have yielded competitive results in text classification tasks, particularly in low-data or constrained environments [18]. Unlike deep models, classical algorithms also require less hyperparameter tuning and are easier to deploy on microcontrollers or FPGAs [19].

2.1.2 Compression and Optimization Techniques

Beyond advanced neural compression methods, classical algorithms also benefit from feature selection, dimensionality reduction, and quantization [20]. Feature selection can greatly reduce the dimensionality in text-based tasks, where large vocabularies can inflate processing demands and memory usage [14]. Techniques such as Chi-square selection, mutual information, and L1 regularization have been shown to maintain classification performance while significantly reducing model size [21]. In parallel, quantization of input features and model weights—even in classical ML—has proven valuable in reducing runtime complexity [22]. PCA and truncated SVD have also been explored to reduce dimensionality while preserving key discriminative patterns in text data [23]. Additionally, hybrid strategies combining feature selection with lightweight ensemble models have proven effective for real-time applications with minimal trade-off in accuracy [24].

Edge environments impose strict hardware limits, yet most work focuses on neural-network compression rather than classical ML. Table 1 lists representative studies on resource constraints and classical approaches, highlighting this gap.

Table 1. Representative Studies on Resource Constraints and Classical Approaches

Reference	Focus / Domain	Main Approach	Key Findings
[6]	Fog/Edge computing paradigm	Data processing near the source	Reduced latency but tight hardware constraints
[7]	Efficient neural network pruning (XNOR-Net)	Binary convolutional layers	Large memory savings but mainly for CNN-based models
[8]	Naive Bayes in Information Retrieval (IR)	Simple probabilistic text modelling	Classical approaches can still be highly competitive
[22]	Pruning filters for efficient ConvNets	Structural compression of CNNs	Focused on deep models; less on classical ML
[25]	CNNs for sentence classification	Demonstrated deep model capacity	High accuracy but large resource footprint

2.2. Text Classification in Sentiment Analysis

Text classification covers a broad range of tasks, from document categorization to spam detection and sentiment analysis[18], [19], [26]. Sentiment analysis typically targets polarity detection—positive, negative, or neutral—of user-generated text such as product reviews, social media posts, and news articles [14].

For large-scale datasets like the Amazon Reviews Polarity corpus, deep learning has dominated top performance metrics [14]. Nonetheless, those approaches require considerable resources, making them less suitable for edge deployment. Classical ML methods can still deliver practical accuracy, particularly when the target device or environment is memory-bound, lacks consistent network connectivity, or must operate offline [21]. Recent comparative studies on sentiment analysis techniques further support this claim, demonstrating that traditional classifiers like Support Vector Machine and Random Forest can achieve robust performance for text classification tasks without the computational overhead of neural approaches [18].

Text classification in sentiment analysis covers a wide range of methods with varying trade-offs between accuracy and resource usage. Table 2 shows the major text classification methods is sentiment analysis.

Table 2. Major Text Classification Methods in Sentiment Analysis

Reference	Method	Dataset	Key Insights	Limitations
[14]	Lexicon-based sentiment analysis	Twitter data	No need for large training sets; interpretable results	Lower accuracy on domain-specific texts
[18]	Traditional classifier ensembles (SVM, NB)	Movie reviews	Ensemble methods can outperform single classifiers in sentiment	High memory usage with large ensembles
[22]	Lightweight text CNN	Multiple short text corpora	Simple convolution layers can be effective	
[27]	BERT-based deep learning	Amazon Reviews Polarity	Near state-of-the-art accuracy	Very large model size; requires GPU/TPU

2.3. Research Gap

While significant progress has been made in optimizing neural architectures for cloud-scale resources, there is comparatively limited research focusing on systematic strategies to implement classical ML models for text classification under severe hardware limitations. Existing literature tends to emphasize either very large deep learning frameworks or highly specialized compression techniques tailored to neural networks [2], [7]. Moreover, many studies overlook the practical constraints of IoT scenarios, where models must provide near-real-time inference despite minimal CPU/GPU availability and limited storage [10]. This gap suggests a need to investigate simpler, less parameter-intensive algorithms such as Naive Bayes, Logistic Regression, and Decision Trees enhanced by feature selection and quantization techniques. Such approaches could benefit various domains requiring efficient inference on constrained hardware, from text classification to environmental monitoring applications [8]. By rigorously evaluating these methods on a standard sentiment classification task, we can offer a blueprint for resource-efficient NLP deployments that preserve acceptable performance while operating in heavily constrained environments.

3. RESEARCH METHODOLOGY

The overarching aim is to demonstrate how lightweight ML models can be efficiently deployed for real-time text classification under hardware limitations. This section describes data sourcing, preprocessing steps, modelling approaches, and optimization strategies.

The overall experimental workflow—starting with the 10 000-sample Amazon Reviews subset, then data cleaning, TF-IDF vectorization, model training (Naive Bayes, Logistic Regression, Decision Tree), Chi-Squared feature selection, simulated quantization, and final evaluation—is illustrated in Figure 1.

3.1. Dataset obtained: Amazon Reviews Polarity Dataset Selection

The Amazon Reviews Polarity dataset comprises a vast collection of product reviews labelled as __label__1 (negative, 1–2 stars) or __label__2 (positive, 4–5 stars) [5]. For this study, a pre-existing subset of 10,000 reviews (5,000 positive and 5,000 negative) was obtained directly without performing any additional preprocessing or filtering. We used a balanced 10 000-sample subset of the Amazon Reviews Polarity dataset; its key statistics are listed in Table 3.

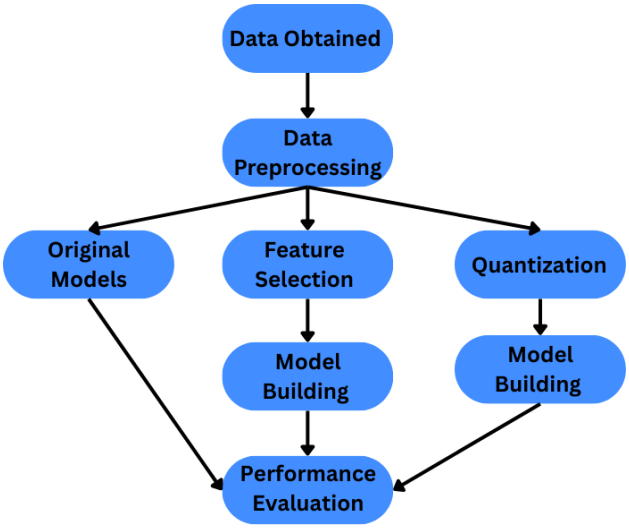


Figure 1. Overall Experimental Workflow Diagram

Table 3. Dataset Statistics for 10 000-Sample Subset

Statistic	Value
Statistic Value	10,000
Positive Reviews	5,000
Negative Reviews	5,000
Vocabulary Size (TF-IDF)	10,000

3.2. Model Selection and Implementation

Three lightweight ML algorithms were selected for implementation and evaluation based on their proven efficiency, interpretability, and low computational requirements making them ideal for resource-constrained environments targeted in this study:

3.2.1. Multinomial Naive Bayes

This probabilistic classifier applies Bayes' theorem under the assumption of feature independence. It is particularly effective for text classification due to its simplicity, high speed, and robustness in handling high-dimensional, sparse data such as bag-of-words or TF-IDF representations. Its low memory footprint makes it well-suited for deployment on edge devices.

3.2.2. Logistic Regression

A linear model that estimates the probability of binary outcomes, Logistic Regression was chosen for its strong baseline performance in text classification and ease of implementation. We use LogisticRegression(max_iter=1000), which includes L2 regularization by default to prevent overfitting while maintaining generalization. Its predictable behaviour and minimal tuning requirements further justify its selection.

3.2.3. Decision Tree

This non-parametric method builds a flowchart-like structure of decisions, offering high interpretability and fast inference. A maximum depth of 20 (max_depth=20) is set to control complexity and reduce the risk of overfitting. Decision Trees are especially useful in environments where transparent decision-making and quick predictions are crucial.

3.3. Optimization Techniques

The goal of these optimizations is to strike a balance between strong predictive performance and minimal resource usage. Below are the two main approaches applied in this study:

3.3.1. Feature Selection (Chi-Squared)

A Chi-Squared (χ^2) statistical test was used to identify features most strongly correlated with the class labels [21]. Specifically, we ranked each TF-IDF term by its χ^2 score, retaining only the top 2,000 or 5,000 features out of the initial 10,000. This selective pruning of the feature space helps reduce model complexity and training overhead, while preserving the key discriminative signals necessary for accurate classification. Similar approaches have proven effective in phishing detection applications [19].

The Chi-Squared feature-selection workflow—from the full TF-IDF matrix to the top-k ranked features—is shown in Figure 2.

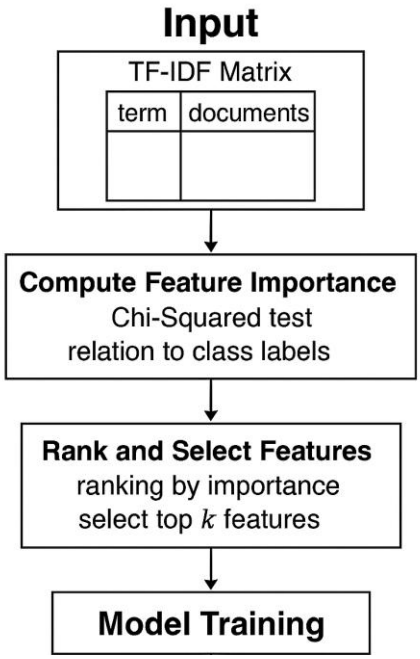


Figure 2. Chi-Squared Feature Selection Workflow

3.4. Evaluation Metrics

To comprehensively assess both classification performance and resource efficiency, we employed a dual-metric evaluation framework. This approach allows us to balance the trade-off between predictive accuracy and computational constraints that are critical in resource-limited edge deployments.

The evaluation framework considered both performance metrics and resource utilization:

Performance Metrics:

- **Accuracy** -Proportion of correctly classified instances
- **Precision** - Ratio of true positives to all predicted positives
- **Recall** - Ratio of true positives to all actual positives
- **F1-Score** - Harmonic mean of precision and recall
- **Area Under ROC Curve (AUC)** - Model's ability to discriminate between classes

Resource Utilization Metrics:

- **Memory Usage** - RAM required during inference (MB)
- **Model Size** - Storage space required for the trained model (MB)
- **Inference Time** - Average time to classify a single text instance (ms)
- **Energy Consumption** - Estimated power usage during inference (mJ)

Resource measurements were conducted on a simulated resource-constrained environment using the Python memory_profiler and time modules. Energy consumption was estimated based on computational operations required for inference.

4. RESULTS AND DISCUSSIONS

4.1. Baseline Performance

After training each model on the processed data, we evaluated Accuracy, Precision, Recall, F1-Score, Model Size, and Training Time. Table 4 reflects results obtained from the 10 000-sample test set dataset using 10,000 TF-IDF features. Logistic Regression leads in accuracy (86.85%), while Naive Bayes boasts the quickest training (0.006 s). The Decision Tree's lower accuracy (71.95%) and longer training time reflect its complexity with higher-dimensional data.

Table 4. Baseline Results for 10 000-Sample Test Set

Model	Accuracy	Precision	Recall	F1-Score	Model Size (MB)	Training Time (s)
Naive Bayes	0.8350	0.8513	0.8231	0.8370	0.306	0.006
Logistic Regression	0.8685	0.8762	0.8669	0.8715	0.077	0.051
Decision Tree	0.7195	0.7549	0.6735	0.7119	0.073	1.806

4.2. Results After Feature Selection (Chi-Squared)

We reduced the TF-IDF feature set to 5,000 terms using a Chi-Squared test. Models were then retrained and evaluated under the same experimental conditions.

As shown in Table 5, feature selection yields a modest accuracy boost for Naive Bayes (83.90%) and reduces Logistic Regression's model size roughly by half (to 0.039 MB). Decision Tree's performance remains comparable to the baseline but trains slightly faster.

4.3. Results After Feature Selection + Quantization

Finally, we simulated int8 weight compression on the feature-selected models to further reduce file sizes. Table 6 and Table 7 compares performance before and after compression.

Combining feature selection with post-training quantization yields substantial file-size reductions especially for the Decision Tree, which shrinks to just 0.016 MB while preserving most of each model's predictive performance.

Table 5. Results after Chi-Squared Feature Selection (5 000 TF-IDF Features)

Model	Accuracy	Precision	Recall	F1-Score	Model Size (MB)	Training Time (s)
Naive Bayes (FS)	0.8390	0.8524	0.8309	0.8415	0.153	0.007
Logistic Regression (FS)	0.8640	0.8714	0.8630	0.8672	0.039	0.024
Decision Tree (FS)	0.7125	0.7441	0.6725	0.7065	0.075	1.158

Table 6. Results after Post-Training Quantization (10 000 TF-IDF Features)

Model	Accuracy	Precision	Recall	F1-Score	Model Size (MB)	Training Time (s)
Naive Bayes (Q)	0.8350	0.8513	0.8231	0.8370	0.271	0.271
Logistic Regression (Q)	0.8685	0.8762	0.8669	0.8715	0.076	0.024
Decision Tree (Q)	0.7175	0.7522	0.6725	0.7101	0.016	1.158

Table 7. Results after Feature Selection + Quantization

Model	Accuracy	Precision	Recall	F1-Score	Compressed Size (MB)	Training Time (s)
Naive Bayes (Fs+ Q)	0.8390	0.8524	0.8309	0.8415	0.127	0.005
Logistic Regression (Fs+ Q)	0.8640	0.8714	0.8630	0.8672	0.038	0.024
Decision Tree (Fs+ Q)	0.7165	0.7505	0.6725	0.7094	0.016	1.159

4.4. Summary of Findings

- *Naive Bayes* sees small performance gains with feature selection, remains highly efficient to train, and compresses well with quantization (whether standalone or combined with FS).
- *Logistic Regression* consistently achieves the highest accuracy, with its file size dropping considerably after both feature selection and compression.
- *Decision Tree* benefits from compression most dramatically in terms of file size, but further tuning could improve its classification metrics and training speed.

Figure 3 compares the accuracy of each model—Naive Bayes, Logistic Regression, and Decision Tree—across four configurations: baseline, after feature selection (FS), after quantization (Q), and after both (FS + Q). The stability in accuracy, with small gains for Naive Bayes post-FS and negligible drops due to quantization, underscores the robustness of classical models under these optimizations.

Precision levels under different optimization scenarios are compared in Figure 4. Precision is highest for Logistic Regression across all variants, with only minor decreases following feature selection (FS) or quantization (Q). Naive Bayes shows a slight improvement after FS, while Decision Tree remains relatively lower.

Recall trends across optimization variants are illustrated in Figure 5. Naive Bayes shows a modest recall gain after feature selection (FS), while Logistic Regression's recall remains nearly constant across baseline, FS, Q, and FS + Q. Decision Tree recall stays lower but consistent through all stages.

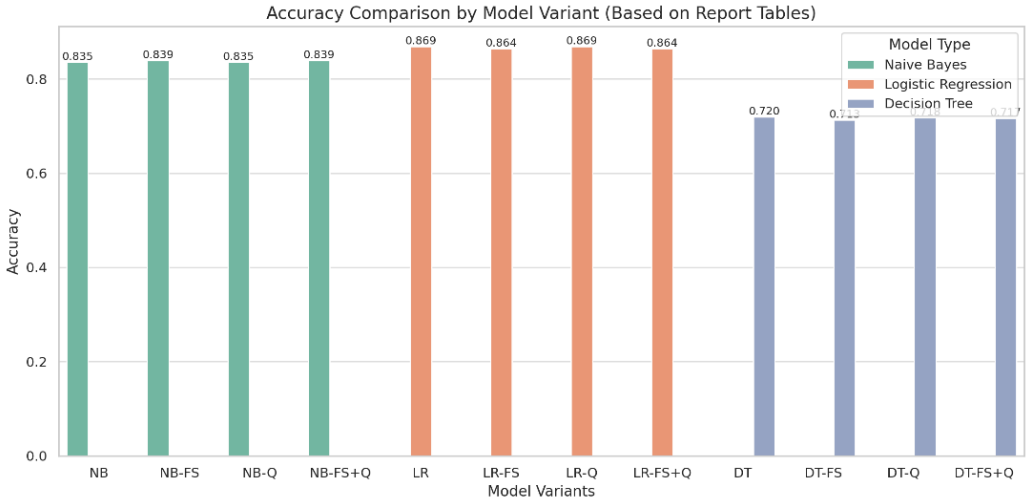


Figure 3. Accuracy Comparison Across Model Variants

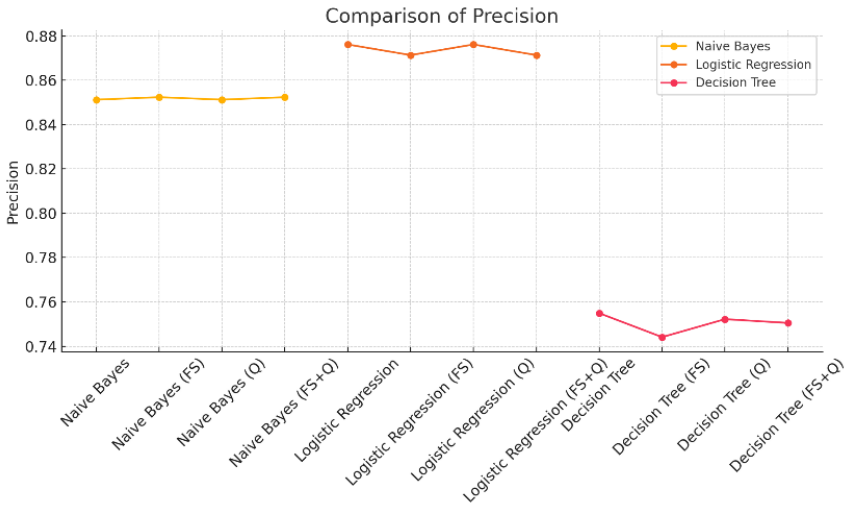


Figure 4. Precision Comparison Across Model Variants

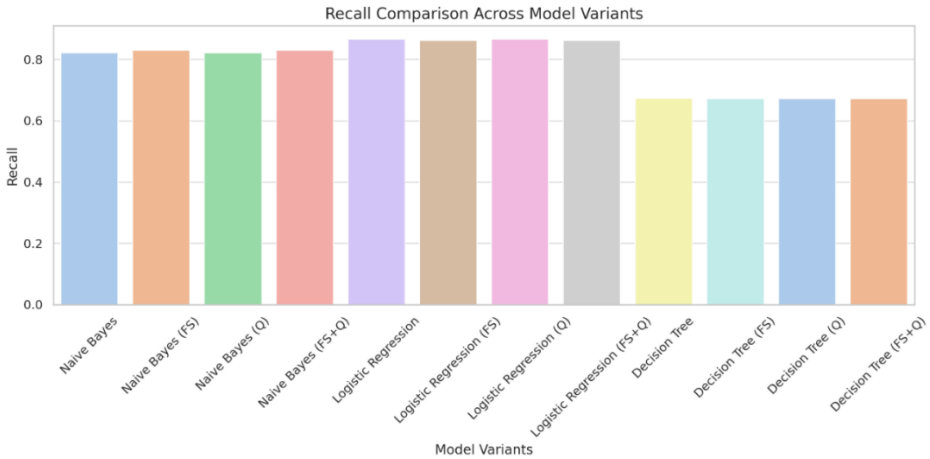


Figure 5. Recall Comparison Across Model Variants

File size comparisons across optimization variants are shown in Figure 6. Quantization significantly reduces model size—especially for the Decision Tree, which shrinks to 0.016 MB—while feature selection also meaningfully reduces size for Logistic Regression.

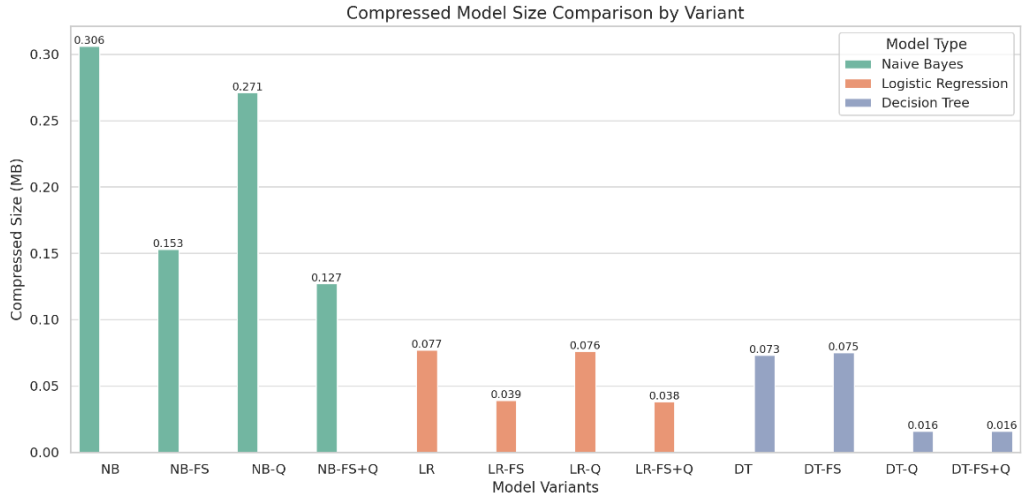


Figure 6. Compressed Model Size Comparison (MB)

Training time comparisons across optimization variants are shown in Figure 7. Naive Bayes is the fastest overall—dropping to 0.005 seconds after feature selection and quantization—while Logistic Regression balances speed and accuracy, reducing from 0.051 to 0.024 seconds. Decision Tree remains the slowest, exceeding 1 second in all variants, underscoring the trade-off between model complexity and training efficiency.

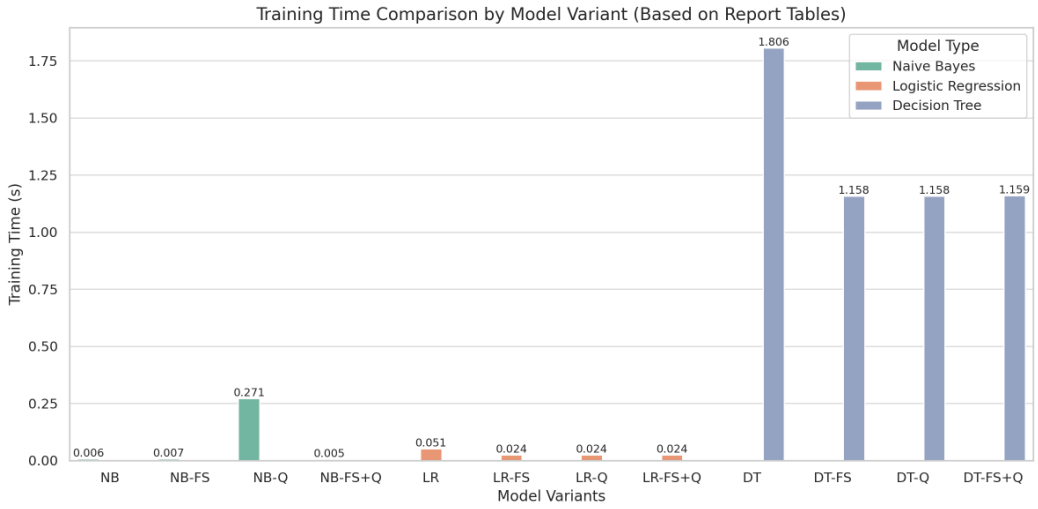


Figure 7. Training Time Comparison Across Model Variants

4.5. Limitations of Simulated vs. Real-World Quantization

Our simulated int8 weight compression approximates storage savings but omits several key aspects of production-grade quantization frameworks such as TensorFlow Lite and ONNX Runtime [16], [17]. First, we do not perform quantization-aware training, which in real pipelines adjusts model parameters during training to compensate for reduced numerical precision and minimize accuracy loss. Second, our simulation applies uniform precision across all

weights, whereas real-world tools support per-channel quantization, operator fusion, and hardware-specific optimizations that substantially affect inference latency and energy consumption on different edge processors (e.g., ARM Cortex-A vs. x86) [16]. Third, production frameworks use calibration steps to determine optimal dynamic ranges for activations and weights—a process not captured in our prototype. Finally, graph-level optimizations (e.g., integer kernel fusion) and custom hardware kernels available in TensorFlow Lite and ONNX Runtime can further shrink model size and accelerate runtime beyond what our simple compression can achieve. Future work should integrate these classical ML models into actual TFLite and ONNX quantization workflows to quantify true performance and accuracy trade-offs on resource-constrained devices.

5. CONCLUSION

A thorough methodology for putting lightweight ML models into practice that allow for real-time text classification on hardware with limited resources has been shown in this study. We trained Multinomial Naive Bayes, Logistic Regression, and Decision Tree models as baselines by selecting 10,000 reviews from the Amazon Reviews Polarity dataset, doing standard preprocessing, and applying TF-IDF vectorization. To further minimize dimensionality, memory, and storage overhead, we then implemented feature selection (using Chi-Squared) and simulated post-training quantization.more efficiently.

Our results demonstrate that these classical models are still very promising substitutes for massive neural networks in edge computing settings when they are optimized. Combining quantization with smaller feature sets resulted in particularly noticeable memory and file-size savings, frequently reaching near-baseline accuracy. By reducing dependency on cloud infrastructure, these enhancements protect data privacy, enable offline functioning, and enable on-device inference. To further balance efficiency and performance, future studies might investigate hybrid modeling techniques, such as utilizing specialized hardware accelerators or combining smaller neural networks with traditional ML.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

FUNDING STATEMENT

The authors received no funding from any party for the research and publication of this article.

AUTHOR CONTRIBUTIONS

Marwah Zaid Mohammed Al-Helali: Conceptualization, Data Curation, Methodology, Validation, Writing – Original Draft Preparation;
Naveen Palanichamy: Project Administration, Supervision, Writing – Review & Editing;
K. Revathi: Review & Editing.

CONFLICT OF INTERESTS

No conflict of interests were disclosed.

ETHICS STATEMENTS

Our publication ethics follow The Committee of Publication Ethics (COPE) guideline. https://publicationethics.org/.

REFERENCES

[1] M.G.S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine Learning at the Network Edge: A Survey," *ACM Comput Surv*, vol. 54, no. 8, pp. 1–37, Nov. 2022, doi: 10.1145/3469029.

[2] D. Mishra, A. Trotta, E. Traversi, M. Di Felice, and E. Natalizio, "Cooperative Cellular UAV-to-Everything (C-U2X) communication based on 5G sidelink for UAV swarms," *Comput Commun*, vol. 192, pp. 173–184, Aug. 2022, doi: 10.1016/j.comcom.2022.06.001.

[3] Y. Chen, "Convolutional Neural Network for Sentence Classification," 2015. Accessed: Mar. 03, 2025. [Online]. Available: https://dspacemainprd01.lib.uwaterloo.ca/server/api/core/bitstreams/2ef42d4c-aba3-4adb-bd99-2ceb6099553b/content

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Computer Vision -- ECCV 2016*, Springer International Publishing, 2016, pp. 525–542, doi: 10.1007/978-3-319-46493-0_32.

[5] X. Zhang, J. Zhao, and Y. LeCun, "Character-level Convolutional Networks for Text Classification," in *Advances in Neural Information Processing Systems*, vol. 28, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 649–657.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, New York, NY, USA: ACM, pp. 13–16, Aug. 2012, doi: 10.1145/2342509.2342513.

[7] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks," *CoRR*, 2017, Accessed: Mar. 03, 2025. [Online]. Available: http://arxiv.org/abs/1710.09282.

[8] T.T. Khoei and N. Kaabouch, "Machine Learning: Models, Challenges, and Research Directions," *Future Internet*, vol. 15, no. 10, pp. 332, Oct. 2023, doi: 10.3390/fi15100332.

[9] E. Yvinec, "Efficient Neural Networks: Post Training Pruning and Quantization," Sorbonne Université, 2023. Accessed: Jun. 03, 2025. [Online]. Available: https://theses.hal.science/tel-04496138

[10] R.I. Mukhamediev *et al.*, "Review of Artificial Intelligence and Machine Learning Technologies: Classification, Restrictions, Opportunities and Challenges," *Mathematics*, vol. 10, no. 15, p. 2552, Jul. 2022, doi: 10.3390/math10152552.

[11] S. Salmani Pour Avval, N.D. Eskue, R.M. Groves, and V. Yaghoubi, "Systematic review on neural architecture search," *Artif Intell Rev*, vol. 58, no. 3, p. 73, Jan. 2025, doi: 10.1007/s10462-024-11058-w.

[12] Y. Zheng, Z. Wei, and J. Liu, "Decoupled Graph Neural Networks for Large Dynamic Graphs," *arXiv preprint*, May 2023, [Online]. Available: https://arxiv.org/pdf/2305.08273

[13] T. Alonso *et al.*, "Elastic-DF: Scaling Performance of DNN Inference in FPGA Clouds through Automatic Partitioning," *ACM Trans Reconfigurable Technol Syst*, vol. 15, no. 2, pp. 1–34, Jun. 2022, doi: 10.1145/3470567.

[14] E. Cambria, D. Das, S. Bandyopadhyay, and A. Feraco, "Affective Computing and Sentiment Analysis," In *A practical guide to sentiment analysis*, pp. 1–10, 2017, doi: 10.1007/978-3-319-55394-8_1.

[15] W. Jiang *et al.*, "Challenges and practices of deep learning model reengineering: A case study on computer vision," *Empir Softw Eng*, vol. 29, no. 6, p. 142, Nov. 2024, doi: 10.1007/s10664-024-10521-0.

[16] "Post-training quantization," TensorFlow. Accessed: Apr. 01, 2025. [Online]. Available: https://www.tensorflow.org/model_optimization/guide/quantization/post_training

[17] "Quantize ONNX models." Accessed: Apr. 01, 2025. [Online]. Available: https://onnxruntime.ai/docs/performance/model-optimizations/quantization.html

[18] T. Ahmed Khan, R. Sadiq, Z. Shahid, M.M. Alam, and M.M. Su'ud, "Sentiment Analysis using Support Vector Machine and Random Forest," *Journal of Informatics and Web Engineering*, vol. 3, no. 1, pp. 67–75, Feb. 2024, doi: 10.33093/jiwe.2024.3.1.5.

[19] M.A. Daniel, S.-C. Chong, L.-Y. Chong, and K.-K. Wee, "Optimising Phishing Detection: A Comparative Analysis of Machine Learning Methods with Feature Selection," *Journal of Informatics and Web Engineering*, vol. 4, no. 1, pp. 200–212, Feb. 2025, doi: 10.33093/jiwe.2025.4.1.15.

[20] X. Cheng, "A Comprehensive Study of Feature Selection Techniques in Machine Learning Models," *Insights in Computer, Signals and Systems*, vol. 1, no. 1, pp. 65–78, Nov. 2024, doi: 10.70088/xpf2b276.

[21] P.V. Dantas, W. Sabino da Silva, L.C. Cordeiro, and C.B. Carvalho, "A comprehensive review of model compression techniques in machine learning," *Applied Intelligence*, vol. 54, no. 22, pp. 11804–11844, Nov. 2024, doi: 10.1007/s10489-024-05747-w.

[22] B. Hawks, J. Duarte, N. J. Fraser, A. Pappalardo, N. Tran, and Y. Umuroglu, "Ps and Qs: Quantization-Aware Pruning for Efficient Low Latency Neural Network Inference," *Front Artif Intell*, vol. 4, Jul. 2021, doi: 10.3389/frai.2021.676564.

[23] B. Kim, "Dimensionality and data size reduction using singular value decomposition," *Issues in Information Systems*, vol. 25, no. 3, pp. 231–237, 2024, doi: 10.48009/3_iis_2024_118.

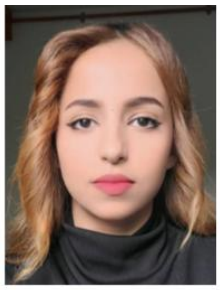
[24] S. Palaniappan, R. Logeswaran, S. Khanam, and Y. Zhang, "Machine Learning Model for Predicting Net Environmental Effects," *Journal of Informatics and Web Engineering*, vol. 4, no. 1, pp. 243–253, Feb. 2025, doi: 10.33093/jiwe.2025.4.1.18.

[25] M. Zulqarnain, R. Ghazali, Y.M.M. Hassim, and M. Rehan, "A comparative review on deep learning models for text classification," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, no. 1, p. 325, Jul. 2020, doi: 10.11591/ijeecs.v19.i1.pp325-335.

[26] M. Zulqarnain, R. Ghazali, Y.M.M. Hassim, and M. Rehan, "A comparative review on deep learning models for text classification," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, no. 1, p. 325, Jul. 2020, doi: 10.11591/ijeecs.v19.i1.pp325-335.

[27] J. Devlin, M.-W. Chang, K. Lee, K.T. Google, and A.I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2019. [Online]. Available: https://github.com/tensorflow/tensor2tensor

BIOGRAPHIES OF AUTHORS



Marwah Zaid Mohammed Al-Helali is an undergraduate student at the Faculty of Computing and Informatics, Multimedia University, Cyberjaya, Malaysia. Her research focuses on lightweight machine learning models, real-time text classification, and edge computing. She is particularly interested in optimizing natural language processing tasks for deployment on resource-constrained devices. Her current work explores efficient algorithm implementation and model compression techniques for edge intelligence applications. She can be contacted at email: 1211307415@student.mmu.edu.my.

	**Palanichamy Naveen** is a Senior Lecturer at the Faculty of Computing and Informatics, Multimedia University. She joined the faculty after completing her PhD at Curtin University, Malaysia. She earned her Bachelor of Engineering (CSE) and Master of Engineering (CSE) from Anna University, India. Her research interests include smart grids, cloud computing, machine learning, deep learning, generative AI and recommender systems. She is actively involved in several research projects funded by Multimedia University. She can be contacted at email p.naveen@mmu.edu.my.
	**K. Revathi** is currently working as an Associate Professor at SRM Valliammai Engineering College, Chennai, India. She has experience of about 15+ Years in Teaching and Research. Her research interests include Affective Computing, Artificial Intelligence, Machine Learning, Deep Learning, Internet of T, and Wireless Sensor Networks. She has published 26 articles in peer-reviewed journals and holds 4 Design Patents and 1 Utility Patent. She also serves as a reviewer for reputed journals. She can be contacted at email neyadharshini@gmail.com.