Indian Journal of Computer Science and Technology
https://www.doi.org/10.59256/indjcst.20260503084
Volume 5, Issue3 (September-December 2026), PP: 691-699.
www.indjcst.com
ISSN No: 2583-5300

Lightweight Machine Learning Models for Mobile-Oriented SMS Spam Detection

Anas Saidu Ibrahim*[1], Mahmud Umar[2], Auwalu Zubairu Abdullahi[3], Abdullahi Isah Abdussalam[4], Kalpesh R. Rakholia[5]

[1, 2, 5]Swarrnim School of Computing & IT, Swarrnim Startup & Innovation University, Gandhinagar, Gujarat, India.
[3,4]Department of Biotechnology, Swarrnim Startup & Innovation University, Gandhinagar, Gujarat, India.
[*] Corresponding Author.

To Cite this Article: *Anas Saidu Ibrahim*[1], Mahmud Umar[2], Auwalu Zubairu Abdullahi[3], Abdullahi Isah Abdussalam[4], Kalpesh R. Rakholia[5], " Lightweight Machine Learning Models for Mobile-Oriented SMS Spam Detection", Indian Journal of Computer Science and Technology, Volume 05, Issue 03 (September-December 2026), PP: 691-699.*

Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the Creative Commons Attribution License; Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: *Unsolicited and scam text messages are a compelling case for on-device filtering, but many recent detectors are based on deep or transformer models that may be large or slow enough to be challenging to run on phones. We evaluate eight classical classifiers—Multinomial and Complement Naive Bayes, Logistic Regression, a linear Support Vector Machine (SVM), a Decision Tree, a Random Forest, a small multi-layer perceptron and XGBoost—on the de-duplicated Short Message Service (SMS) Spam Collection (5,171 messages) from the University of California Irvine (UCI) repository using term frequency-inverse document frequency (TF-IDF) unigram and bigram features, tuned inside a nested ten-fold cross validation, comparing predictive quality, serialized model size and single-message inference latency. The linear SVM, Logistic Regression and Multinomial Naive Bayes form a top tier, with cross validated spam-class F1 scores of 0.951, 0.951 and 0.950 respectively—with no pairwise statistically significant differences among them—occupying 223–340 KiB and classifying a message in about 1 ms via a Python pipeline running on a desktop central processing unit (CPU). The Random Forest and the neural network made no accuracy improvements, while being roughly 13 and 18 times larger than the linear models. A study of vocabulary size shows that constraining the linear SVM to 2,000 terms reduces the model size to 88 KiB at a negligible sacrifice of 0.007 F1. A library-free export of Multinomial Naive Bayes, Logistic Regression and the linear SVM matched the scikit-learn predictions on all 1,035 held-out messages for possible Android implementation; on-device timings remain future work. Error analysis reveals that remaining errors tend to cluster in very short, conversational or announcement-like messages.*

Key Word: *SMS spam detection; lightweight machine learning; TF-IDF; Naive Bayes; support vector machine; mobile computing*

I. INTRODUCTION

Short Message Service (SMS) remains one of the most widely used channels for personal, commercial and security-critical communication, and it is therefore a persistent target for unsolicited bulk messages and fraud, including SMS phishing ("smishing"). Reviews of the field describe how filtering has moved from blacklists and hand-written rules towards statistical learning on message content [1,2].

Recent work has reported accuracies between roughly 98% and 99.8% on the standard benchmark with convolutional and recurrent networks [3], transformer encoders [4,5,6] and DistilBERT-based pipelines [7]. These gains come with costs that matter on phones: even a distilled transformer such as DistilBERT has roughly 66 million parameters [8], and server-side inference would require sending private message content off the device.

A smaller body of work targets the phone directly. A compact on-device smishing pipeline with a 3.46 MB detection network has been proposed [9], an on-device classifier of about 127 kB reached an accuracy of 0.99 on a large Korean smishing corpus [10], and a recent evaluation of Naive Bayes and SVM on hybrid feature spaces reported inference in under 10 ms on a standard CPU [11].

What is still missing is a controlled, deployment-oriented comparison of the classical models that practitioners would actually consider shipping: one that tunes every model fairly, tests whether apparent differences are statistically meaningful, and reports what each model costs in storage and latency rather than only how accurate it is. This paper provides such a **comparison on the SMS Spam Collection [12]. Our contributions are:**

1. a nested cross-validation benchmark of eight classifiers (two Naive Bayes variants, Logistic Regression, a linear SVM, a Decision Tree, a Random Forest, a small neural network and XGBoost) with per-fold significance tests;
2. deployment-oriented measurements—serialized model size, single-message inference latency and training time—reported alongside accuracy, precision, recall, F1 and ROC-AUC;
3. a vocabulary-size study that quantifies how far a linear model can be shrunk before spam-detection quality drops; a library-free export of three trained linear models (Multinomial Naive Bayes, Logistic Regression and the linear SVM),

verified to reproduce scikit-learn's predictions on all held-out messages, as a specification for an Android implementation; and an error analysis of the messages that the best models still miss.

4. The remainder of the paper is organized as follows. Section II reviews related work, Section III describes the data and methods, Section IV presents and discusses the results, and Section V concludes.

II. RELATED WORK

A. Classical Machine Learning for SMS Spam

Almeida et al. created the SMS Spam Collection and compared several classifiers on it, support-vector machines being one of the strongest [13]. Delany et al. reviewed SMS spam filtering methods and data [1], and Abdulhamid et al. reviewed mobile SMS filtering [2]. The Naive Bayes "tradition" in spam filtering, including the choice between event models [14,15], is firmly established, and machine learning for spam filtering in general has been reviewed [16]. Later work has evaluated TF-IDF and bag-of-words features with supervised classifiers, often with resampling to balance classes, and a Random Forest achieved around 99% accuracy in [17], while papers published in 2025 continue to evaluate classical learners and extreme learning machines for mobile SMS spam detection [18,19].

B. Deep and Transformer-Based Models

Roy et al. applied convolutional and recurrent networks to SMS spam [3], Liu et al. adapted a transformer for the task [4], and Oswald et al. combined BERT embeddings with intention analysis in SpotSpam [5]. Uddin et al. fine-tuned DistilBERT and RoBERTa, obtained 99.84% accuracy after balancing by back-translation, and performed an explainability analysis [6]. Ilyasa and Khadidos combined an SVM, DistilBERT embeddings and a voting classifier with auto-tuned hyperparameters, and reached 99.6% accuracy [7]. Their TF-IDF SVM baselines had F1 of around 0.92–0.95, showing the speed/size trade-off of a lightweight deployment. For smishing, Mishra and Soni proposed a two-phase model of URL checking and content classification [20].

C. Lightweight and On-Device Detection

Wei and Nguyen proposed a lightweight deep neural classifier for SMS spam [21]. On-device smishing detection has been investigated with a compact variational-autoencoder pipeline [9], and with a 127 kB deep-learning classifier trained with real victim-reported messages and measured against text-evasion attacks [10]. Verma and Sadana experimented with Naive Bayes and SVM classifiers on text, metadata and hybrid features, and also investigated cross-dataset robustness, adversarial robustness and CPU load [11].

Relative to these works, we focus on the fairly well understood, typically portable, part of the space and make the comparison systematic: all models are tuned within the validation loop; all variations are statistically tested; and size/latency is given for every model, rather than just the proposed one.

III. MATERIALS AND METHODS

A. Dataset

We used the SMS Spam Collection from the UCI Machine Learning Repository [12,13], which contains 5,574 English messages labelled as spam or legitimate ("ham") and was assembled from several sources, including a UK spam-reporting forum and a Singapore university message corpus [13,7]. Because the collection contains repeated texts, we removed exact duplicates: messages with identical text (case-sensitive) were identified and only the first occurrence was kept. This step was applied before any data partitioning, so identical messages cannot appear on both sides of a split, which would inflate measured accuracy. It removed 403 messages and left 5,171 unique messages (653 spam and 4,518 ham; 12.6% spam); no duplicated text carried conflicting labels, and near-duplicates were not removed. For the hold-out experiment the data were split once into a stratified 80% training set (4,136 messages) and a 20% test set (1,035 messages: 904 ham and 131 spam).

B. Preprocessing and Feature Extraction

Messages are lower-cased, URLs are replaced by one placeholder token and digit runs by another. No stopword removal or stemming is applied, since function words, spelling variants and abbreviations are themselves informative in SMS. Features are TF-IDF weights [22,23] over word unigrams and bigrams, keeping terms that occur in at least two messages and at most 5,000 terms overall (the limit is varied in Section IV-D). The weight of term t in message d is

$$w_{t,d} = \left(1 + \ln f_{t,d}\right) \cdot \mathrm{idf}(t) \qquad (1)$$

With the smoothed inverse document frequency

$$\mathrm{idf}(t) = \ln\left(\frac{1+n}{1+\mathrm{df}(t)}\right) + 1 \qquad (2)$$

where f is the raw count, n the number of training messages and df(t) the number containing t. Each vector is L2-normalized, and the vectorizer is fitted on training data only within every split (Fig. 1).

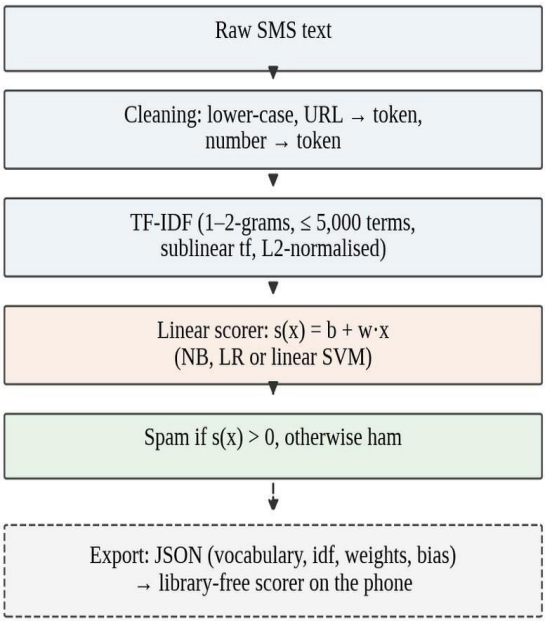


Fig. 1. Processing pipeline: text cleaning, TF-IDF features, a linear scorer and export of the trained model for on-device use.

C. Classifiers

Eight classifiers were compared (Table 1). Multinomial Naive Bayes [15] and its Complement variant [24] are probabilistic baselines; Logistic Regression and a linear support-vector machine [25] (scikit-learn's LIBLINEAR-based implementation [26]) are linear models; a Decision Tree and a Random Forest of 100 trees [27] represent tree-based learners; a multilayer perceptron with one hidden layer, trained with early stopping, represents a small neural network; and XGBoost [28] represents gradient boosting. All models other than XGBoost use scikit-learn [29]. Class weights were balanced for Logistic Regression, the SVM, the Decision Tree and the Random Forest to counter the 12.6% spam prevalence. The MLP was trained for at most 100 iterations with early stopping, XGBoost used a learning rate of 0.1, and the random seed was fixed to 42 wherever applicable.

Model	Search grid	Selected
Multinomial NB	$\alpha \in \{0.03, 0.1, 0.3, 1\}$	$\alpha = 0.03$
Complement NB	$\alpha \in \{0.03, 0.1, 0.3, 1\}$	$\alpha = 1$
Logistic Regr.	$C \in \{1, 3, 10, 30\}$	$C = 30$
Linear SVM	$C \in \{0.1, 0.3, 1, 3\}$	$C = 3$
Decision Tree	depth $\in \{20, 40, \text{none}\}$; min. leaf $\in \{1, 2\}$	40; 1
Random Forest	depth $\in \{50, \text{none}\}$; min. leaf $\in \{1, 2\}$	none; 2
MLP (64)	L2 $\alpha \in \{10^{-4}, 10^{-2}\}$; hidden $\in \{32, 64\}$	10^{-4}; 32
XGBoost	depth $\in \{4, 6\}$; trees $\in \{150, 300\}$	4; 300

Table 1. Hyperparameter grids and values selected on the hold-out training split (4,136 messages); the nested cross-validation selects its own values inside every fold.

D. Evaluation Protocol

Models were first compared with nested, stratified 10-fold cross-validation on all 5,171 de-duplicated messages. Inside each outer training fold, a 3-fold grid search (Table 1) selected hyperparameters by spam-class F1, and the refitted model was scored on the untouched outer fold, which avoids the optimistic bias of tuning and testing on the same folds. Spam is the positive class. We report accuracy, precision (P), recall (R), F1 and ROC-AUC, where

$$F1 = \frac{2PR}{P+R} \quad (3)$$

Precision matters because a false positive blocks a legitimate message, whereas recall measures how much spam is caught. To test whether differences between models are meaningful, a Wilcoxon signed-rank test [30] was applied to the paired per-fold F1 scores of the best-ranked model and each other model. Because cross-validation folds share training data,the p-values are approximate [31]; they are read as indicative, and no correction for multiple comparisons was applied. In addition, a separate single stratified hold-out evaluation (Section III-A), drawn from the same de-duplicated data, was used to produce confusion matrices, ROC curves and deployment measurements, with hyperparameters tuned on the training split only and the test messages playing no role in model selection. The two experiments are separate evaluations but they are not statistically independent: the 1,035 hold-out test messages are also part of the 5,171 messages used in the nested cross-validation, so the two sets of results overlap and should not be read as independent confirmations of each

other. The values in Table 1 belong to the hold-out experiment, whereas the nested cross-validation selects its own configuration in every fold (Section IV-A).

E. Deployment-Cost Measurement

Model size measures the pickled (serialized) size of the complete pipeline (the fitted TF-IDF vectorizer with its vocabulary and idf, plus the classifier parameters) in KiB (1 KiB = 1024 bytes). Pickled sizes depend on library versions. Latency measures the wall-clock time, reported in milliseconds per message, to classify one message at a time through the complete pipeline (cleaning, vectorization and prediction). Ensemble models were artificially constrained to single-threaded execution so that thread start-up time is not a factor in benchmarking; each model was provided 100 warm-up calls, and the median of five iterations of 300 test messages was used in reporting. Training time measures the mean wall-clock time in seconds per outer fold, and includes the inner grid search (parallel on all logical cores); iteration times can be magnified by model complexity and the number of inner cross-validation folds. All experiments were executed on a Windows 11 personal computer (Intel processor, Family 6 Model 142, 8 logical cores, 8 gigabytes RAM) using the CPU only with Python 3.11.9, scikit-learn 1.9.1, XGBoost 3.2.0, NumPy 2.4.6, pandas 3.1.0 and SciPy 1.17.1. These are desktop CPU values: they support comparison across models but are not on-device measurements.

F. Portable Linear Scorer

A fitted linear model, and Multinomial Naive Bayes on TF-IDF features, reduce to a linear score

$$s(x) = b + \sum_j w_j x_j \qquad (4)$$

where x is the message's TF-IDF vector; the message is flagged as spam when s(x) > 0. For Logistic Regression and the SVM, w and b are the learned coefficients and intercept; for Naive Bayes, $w_j = \ln P(t_j \mid \text{spam}) - \ln P(t_j \mid \text{ham})$ and $b = \ln P(\text{spam}) - \ln P(\text{ham})$. For Multinomial Naive Bayes, Logistic Regression and the linear SVM we export the vocabulary, idf values, weights and bias to a JSON file together with a textual description of the preprocessing. To check that this description is a sufficient specification, an independent scorer that reads only the JSON file was compared with the scikit-learn pipeline on the hold-out messages (Section IV-F).

IV. RESULTS AND DISCUSSION

A. Cross-Validated Performance

Table 2 and Fig. 2 summarize the nested cross-validation results. Three models form a top tier: the linear SVM (F1 0.951 ± 0.024), Logistic Regression (0.951 ± 0.023) and Multinomial Naive Bayes (0.950 ± 0.022), all with an accuracy of about 0.988. The multilayer perceptron (0.942) and the Random Forest (0.941) follow about 0.01 F1 lower, and Complement Naive Bayes (0.929), XGBoost (0.927) and the Decision Tree (0.838) trail. The models differ in how they err: the MLP and Multinomial NB have the highest precision (0.987 for both) but lower recall (0.902 and 0.917), whereas Logistic Regression and Complement NB catch more spam (recall 0.948 and 0.947) and the SVM lies in between (precision 0.963, recall 0.940).

Model	Acc.	Prec.	Rec.	F1 (± SD)
Linear SVM	0.988	0.963	0.940	0.951 ± 0.024
Logistic Regr.	0.988	0.954	0.948	0.951 ± 0.023
Multinomial NB	0.988	0.987	0.917	0.950 ± 0.022
MLP (64)	0.986	0.987	0.902	0.942 ± 0.024
Random Forest	0.985	0.949	0.934	0.941 ± 0.014
Complement NB	0.982	0.912	0.947	0.929 ± 0.027
XGBoost	0.982	0.958	0.899	0.927 ± 0.022
Decision Tree	0.958	0.821	0.858	0.838 ± 0.030

Table 2. Nested 10-fold cross-validation results (spam class; mean over folds).

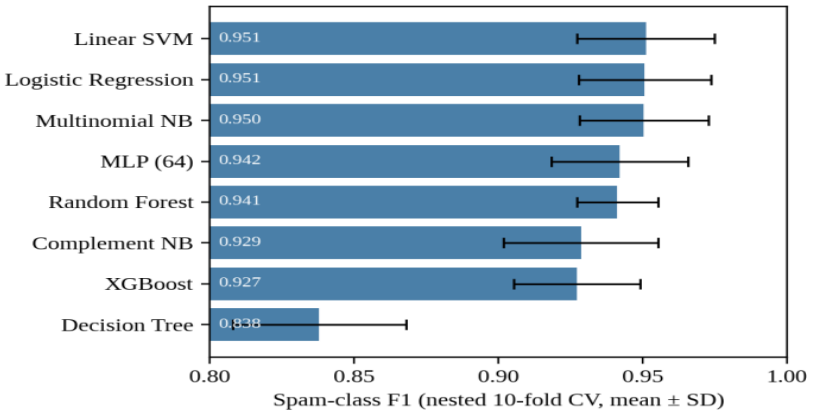


Fig. 2. Spam-class F1 of the eight classifiers in nested 10-fold cross-validation (mean ± standard deviation).

Wilcoxon tests against the linear SVM, the top-ranked model, found no significant difference from Logistic Regression ($p = 0.84$), Multinomial NB ($p = 0.85$), the Random Forest ($p = 0.10$) or the MLP ($p = 0.084$), but significant differences from Complement NB and the Decision Tree (each $p = 0.002$, the smallest value attainable with ten folds) and from XGBoost ($p = 0.004$). A non-significant test does not prove equivalence, and ten folds give limited power; no statistically significant differences were detected among the SVM, Logistic Regression and Multinomial NB under this procedure, so these models are best regarded as performing comparably in these experiments. The Random Forest and the MLP, about 0.01 F1 lower, cannot be clearly separated from them statistically either. The p-values are approximate because the folds share training data, and no correction for multiple comparisons was applied (Section III-D).

The configuration selected inside the folds was not always the same. Logistic Regression chose $C = 30$ in eight folds and $C = 10$ in two, Multinomial NB chose $\alpha = 0.03$ in seven folds and $\alpha = 0.1$ in three, the SVM chose $C = 3$ in six folds and $C = 1$ in four, and Complement NB chose $\alpha = 1$ in all ten, whereas the Decision Tree, XGBoost and the MLP varied more. Because the grids were small and several selected values lie at the edge of their grid (Table 1), larger grids might change individual scores slightly.

B. Hold-Out Results and Error Counts

Tables 3 and 4 list the hold-out results and confusion counts. Multinomial NB, Logistic Regression and the linear SVM again lead, with F1 of 0.953, 0.952 and 0.952, but they err differently: Multinomial NB made 2 false positives and missed 10 spam messages (precision 0.984, recall 0.924), Logistic Regression made 1 false positive and 11 misses (0.992, 0.916), and the SVM made no false positive and 12 misses (precision 1.000, recall 0.908). Multinomial NB and Logistic Regression disagreed on 8 of the 1,035 test messages. The MLP also raised no false alarms (precision 1.000) but missed 20 spam messages, XGBoost made 2 false positives and missed 18, and Complement NB caught the most spam (125 of 131) at the price of 10 false positives. The Decision Tree was clearly the weakest, with 25 false positives and 23 misses. ROC-AUC (Fig. 3) is 0.996 for Logistic Regression and the SVM, 0.992 for Multinomial NB, 0.991 for the Random Forest, 0.987–0.989 for the MLP and Complement NB, 0.983 for XGBoost and 0.898 for the Decision Tree. Because the test set contains only 131 spam messages, one message changes recall by about 0.8 percentage points; rankings among the top models on this split are therefore within noise, and cross-validation is used for ranking.

Model	Prec.	Rec.	F1	AUC	Size (KiB)	Lat. (ms)
Multinomial NB	0.984	0.924	0.953	0.992	340	0.98
Complement NB	0.926	0.954	0.940	0.989	379	0.97
Logistic Regr.	0.992	0.916	0.952	0.996	223	0.98
Linear SVM	1.000	0.908	0.952	0.996	223	1.00
Decision Tree	0.812	0.824	0.818	0.898	200	1.02
Random Forest	0.960	0.916	0.938	0.991	2,832	9.56
MLP (64)	1.000	0.847	0.917	0.987	3,942	2.04
XGBoost	0.983	0.863	0.919	0.983	504	2.53

Table 3. Hold-out results and deployment cost (spam class).

Model	TN	FP	FN	TP
Multinomial NB	902	2	10	121
Complement NB	894	10	6	125
Logistic Regr.	903	1	11	120
Linear SVM	904	0	12	119
Decision Tree	879	25	23	108
Random Forest	899	5	11	120
MLP (64)	904	0	20	111
XGBoost	902	2	18	113

Table 4. Hold-out confusion counts (1,035 messages: 904 ham, 131 spam).

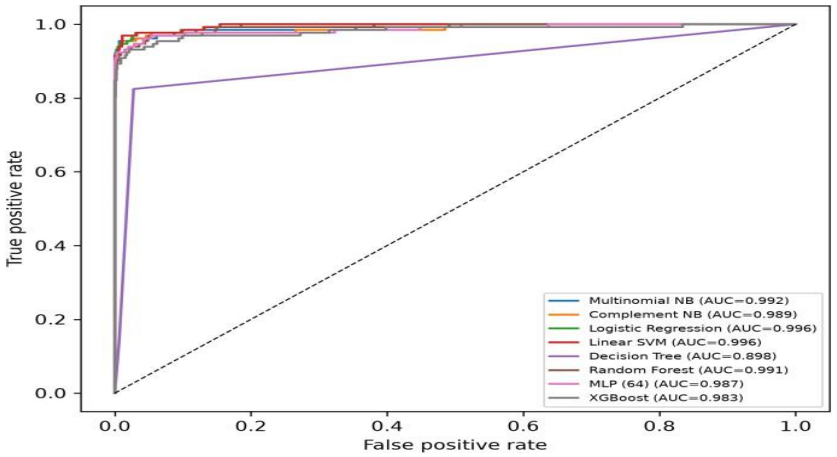


Fig. 3. ROC curves of the eight classifiers on the hold-out set

C. Deployment Cost

Fig. 4 shows model size and latency (desktop CPU, Python). The Decision Tree (200 KiB), Logistic Regression and the SVM (223 KiB each), Multinomial NB (340 KiB) and Complement NB (379 KiB) are all below 400 KiB, most of which is the 5,000-term vocabulary rather than the classifier; XGBoost needs 504 KiB. The Random Forest needs 2,832 KiB, about 13 times the size of the linear models, and the MLP 3,942 KiB, about 18 times. Latency tells a similar story: five of the eight models classify a message in roughly 1 ms through the complete Python pipeline, whereas the MLP needs 2.0 ms, XGBoost 2.5 ms and the Random Forest 9.6 ms, about ten times slower than the linear models. Training is also cheaper for the simple models: including the parallel inner grid search, a cross-validation fold took about 1.3 s for the SVM and 2.0 s for Logistic Regression, compared with 18 s for XGBoost and 52 s for the MLP. Neither the Random Forest nor the MLP bought any accuracy for these costs (Table 2). The roughly 1 ms figure includes Python library overhead and was measured on a desktop CPU; a purpose-built scorer (Section IV-F) performs only dictionary look-ups and one dot product, so its on-device cost is expected to be small, but this was not measured.

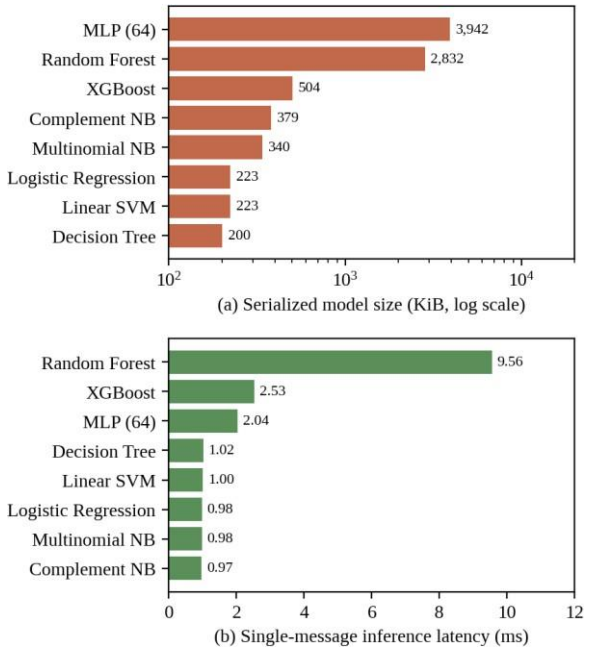


Fig. 4. (a) Serialized model size and (b) single-message inference latency of the eight classifiers (desktop CPU, Python).

D. Vocabulary-Size Study

Terms	SVM F1	SVM KiB	NB F1	NB KiB
250	0.901	12	0.905	17
500	0.924	22	0.925	34
1,000	0.935	44	0.929	68
2,000	0.947	88	0.939	135
5,000	0.954	223	0.955	340
10,000	0.958	451	0.952	685

Table 5. Effect of the vocabulary limit (5-fold CV, fixed settings; size in KiB).

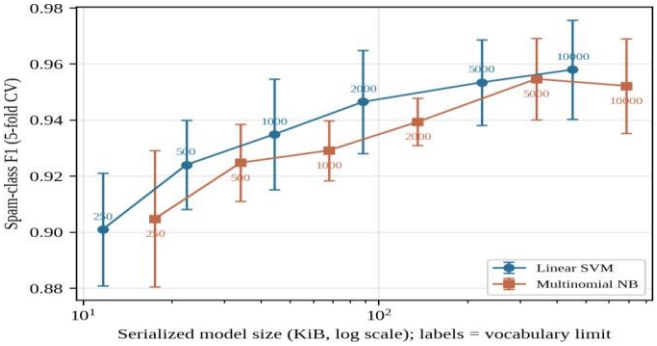


Fig. 5. Spam-class F1 versus serialized model size for different vocabulary limits (mean ± SD over 5 folds).

Because vocabulary dominates model size, we varied the vocabulary limit from 250 to 10,000 terms for the linear SVM and Multinomial NB (5-fold cross-validation with fixed, untuned settings: SVM $C = 1$, NB $\alpha = 0.1$; Table 5 and Fig. 5). Quality rises steeply up to about 2,000 terms and then flattens. For the SVM, F1 grows from 0.901 (12 KiB) at 250 terms to 0.947 (88 KiB) at 2,000 terms and 0.954 (223 KiB) at 5,000 terms; the further gain to 0.958 at 10,000 terms (451 KiB)

is far smaller than the fold-level standard deviation of about 0.018. Multinomial NB shows no benefit beyond 5,000 terms (F1 0.955 at 340 KiB versus 0.952 at 685 KiB). A 2,000-term SVM is therefore an attractive operating point: it is 2.5 times smaller than the 5,000-term model for a drop of about 0.007 F1, and at 1,000 terms the model is only 44 KiB (F1 0.935), which may suit very constrained settings.

E. Error Analysis

Multinomial NB made 12 errors on the hold-out set: 2 false positives and 10 missed spam messages. The false positives were a very short, generic comment about a mobile phone and a fragment that sounds like spam ("Unlimited texts. Limited minutes.") but is in fact legitimate. Of the 10 missed spam messages, three were adult-themed (two flirtatious, colloquial messages and one built around a web address and some unusual characters); two were very short, alert-style messages (for example "I've got a ringtone and missed-call alert") containing only a shortcode or mobile number; three were regular-looking announcements or news-style promotions (tickets to an event, a burger-for-breakfast promotion and a joke news-feed subscription); one was a competition entry of the quiz style; and one used the sender-keyword-in-the-sender's-name style ("TN5"), which is another type of distinctive sender feature. These are typified by the absence of classical spam words ("free", "win", "urgent"), by the association with sender-level data, semantics or contextuality, and by the fact that several would have been hard for a human reader, unaided, without sender information.

F. Portable Model Export

The exported Multinomial NB, Logistic Regression and linear SVM files are 278, 281 and 273 KiB. An independent scorer that used only the JSON content—re-implementing the cleaning, tokenization, n-gram construction, sublinear tf-idf weighting and L2 normalization of Section III-B—produced the same prediction as the scikit-learn pipeline on all 1,035 hold-out messages (100.00% agreement) for each of the three models. The exported description can therefore serve as a specification for a Kotlin or Java implementation, whose run-time work is one dictionary look-up per n-gram and one sparse dot product, with no machine-learning library. Packaging and timing such an implementation on Android devices is the main item of future work.

G. Discussion and Comparison with Prior Work

Our absolute scores are slightly below many published figures on this collection: accuracies of about 98.8% and a spam-class F1 of about 0.95 for the best models. Two factors probably explain much of this. First, we removed duplicates, and near-identical messages that straddle a split make a benchmark easier. Second, several published results rely on augmentation or resampling, on different splits, or on accuracy as the headline metric, which is dominated by the majority class [6,7]; such numbers are not directly comparable with ours. Within one study, DistilBERT-based features improved F1 over TF-IDF SVMs by roughly 0.04–0.05 [7]. Whether such a gain justifies a transformer on the phone depends on the application. Where storage, latency, energy and privacy dominate, our results show that tuned linear models already deliver an F1 near 0.95 in 223 KiB and about a millisecond (desktop Python), and that shrinking to under 100 KiB costs about 0.007 F1. This agrees with the conclusion of [11] that lightweight models are a practical alternative to heavy architectures for real-time spam mitigation.

In practice the evidence supports a simple recipe: TF-IDF unigrams and bigrams with 2,000–5,000 terms and a tuned, regularized linear classifier (Logistic Regression or linear SVM) for the best balance of recall and precision, or Multinomial NB when avoiding false alarms matters most (cross-validated precision 0.987 versus 0.954–0.963 for Logistic Regression and the SVM). The decision threshold can then be moved to trade missed spam against false alarms.

H. Limitations

The study employs a single public collection of messages (off-line, English only, collected from older messages), so results are not generalizable to new spam, to other languages or code-mixed content; this is future work. Latency and size were measured using Python on a desktop CPU, not on phones. No test was performed on robustness to deliberate noise or intentional obfuscation; the models are trained on message text alone (no sender, no URL reputation, no timing information). The grids searched were small, p-values from overlapping cross-validation folds are approximate and not corrected for multiple hypothesis testing, and the hold-out set contained only 131 spam messages. Several hyperparameters defined at the edges of the search grid (Table 1) had varied best values across the individual cross-validation folds, and larger grids would likely cause the scores to change slightly. The nested cross-validation and the hold-out evaluation sets overlap in that the 1,035 hold-out messages are part of the 5,171 message corpora used in cross-validation; as a result, the two experimental results are not independent. Conducting cross-validation only on the 4,136 training messages and leaving the hold-out set completely unseen would have prevented this overlap and is considered future work.

V. CONCLUSION AND FUTURE SCOPE

This paper presents a comparison of eight classical classifiers for SMS spam detection with deployment cost as a main criterion. With nested cross-validated tuning, the linear SVM, Logistic Regression and Multinomial Naive Bayes performed comparably at the top (F1 0.950–0.951, with no statistically significant differences detected), required just 223–340 KiB, and on a desktop CPU needed about 1 ms per message in a Python pipeline. The Random Forest and the neural network were roughly 13 and 18 times larger than the linear models and offered no accuracy benefit. A study of vocabulary

size found that a 2,000-term SVM takes just 88 KiB at a cost of about 0.007 F1, and an exported JSON definition was found to reproduce the reference models exactly on the hold-out messages, which is expected to make a library-free Android implementation relatively straightforward (but it has not yet been implemented or timed on a phone).

Future work will implement and time the scorer on Android devices, evaluate cross-dataset and multilingual (including code-mixed) generalization, test robustness to obfuscated text, and explore calibrated thresholds, character n-gram features and distilled transformer embeddings against the same cost metrics.

Data Availability

The SMS Spam Collection is publicly available from the UCI Machine Learning Repository [12]. All experiments used a fixed random seed (42); the preprocessing, model configurations and evaluation procedure are described in Section III.

REFERENCES

1. S. J. Delany, M. Buckley, and D. Greene, "SMS spam filtering: Methods and data," Expert Syst. Appl., vol. 39, no. 10, pp. 9899–9908, 2012.
2. S. M. Abdulhamid et al., "A review on mobile SMS spam filtering techniques," IEEE Access, vol. 5, pp. 15650–15666, 2017.
3. P. K. Roy, J. P. Singh, and S. Banerjee, "Deep learning to filter SMS spam," Future Gener. Comput. Syst., vol. 102, pp. 524–533, 2020.
4. X. Liu, H. Lu, and A. Nayak, "A spam transformer model for SMS spam detection," IEEE Access, vol. 9, pp. 80253–80263, 2021.
5. C. Oswald, S. E. Simon, and A. Bhattacharya, "SpotSpam: Intention analysis–driven SMS spam detection using BERT embeddings," ACM Trans. Web, vol. 16, no. 3, 2022, doi: 10.1145/3538491.
6. M. A. Uddin, M. N. Islam, L. Maglaras, H. Janicke, and I. H. Sarker, "ExplainableDetector: Exploring transformer-based language modeling approach for SMS spam detection with explainability analysis," arXiv:2405.08026, 2024.
7. S. N. Ilyasa and A. O. Khadidos, "Optimized SMS spam detection using SVM-DistilBERT and voting classifier: A comparative study on the impact of lemmatization," Int. J. Adv. Comput. Sci. Appl., vol. 15, no. 11, pp. 1323–1333, 2024.
8. V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," arXiv:1910.01108, 2019.
9. Harichandana B. S. S., S. Kumar, M. B. Ujjinakoppa, and B. R. K. Raja, "COPS: A compact on-device pipeline for real-time smishing detection," in Proc. IEEE 21st Consumer Communications & Networking Conf. (CCNC), Las Vegas, NV, USA, 2024, pp. 172–179, doi: 10.1109/CCNC51664.2024.10454635.
10. J. W. Seo, J. S. Lee, H. Kim, J. Lee, S. Han, J. Cho, and C.-H. Lee, "On-device smishing classifier resistant to text evasion attack," IEEE Access, vol. 12, pp. 4762–4779, 2024, doi: 10.1109/ACCESS.2024.3349577.
11. R. K. Verma and S. Sadana, "Textual, non-textual, and hybrid feature engineering for SMS spam classification," IEEE Access, vol. 13, pp. 176901–176914, 2025, doi: 10.1109/ACCESS.2025.3620751.
12. T. A. Almeida and J. M. Gómez Hidalgo, "SMS Spam Collection," UCI Machine Learning Repository. [Online]. Available: https://archive.ics.uci.edu/dataset/228/sms+spam+collection (accessed Nov. 1, 2026).
13. T. A. Almeida, J. M. Gómez Hidalgo, and A. Yamakami, "Contributions to the study of SMS spam filtering: New collection and results," in Proc. ACM Symp. Document Engineering (DocEng), 2011.
14. V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive Bayes—Which naive Bayes?" in Proc. 3rd Conf. Email and Anti-Spam (CEAS), 2006.
15. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," in Proc. AAAI-98 Workshop Learning for Text Categorization, 1998, pp. 41–48.
16. T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering," Expert Syst. Appl., vol. 36, no. 7, pp. 10206–10222, 2009. .
17. A. Abid, S. Ullah, M. A. Siddique, M. F. Mushtaq, W. Aljedaani, and F. Rustam, "Spam SMS filtering based on text features and supervised machine learning techniques," Multimedia Tools Appl., vol. 81, pp. 39853–39871, 2022.
18. P. Ozoh et al., "SMS spam detection using machine learning approach," Int. STEM J., vol. 6, no. 1, pp. 10–27, Jun. 2025.
19. M. Birthare, N. Jain, and A. Meena, "Machine learning techniques on mobile SMS spam detection," in Proc. Int. Conf. Recent Advancements and Modernisations in Sustainable Intelligent Technologies and Applications (RAMSITA 2025), Atlantis Press, May 2025, doi: 10.2991/978-94-6463-716-8_7.
20. S. Mishra and D. Soni, "DSmishSMS—A system to detect smishing SMS," Neural Comput. Appl., vol. 35, pp. 4975–4992, 2023.
21. F. Wei and U. T. Nguyen, "A lightweight deep neural model for SMS spam detection," in Proc. Int. Symp. Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, Oct. 2020, pp. 1–6, doi: 10.1109/ISNCC49221.2020.9297350.
22. K. Spärck Jones, "A statistical interpretation of term specificity and its application in retrieval," J. Documentation, vol. 28, no. 1, pp. 11–21, 1972.
23. G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," Inf. Process. Manage., vol. 24, no. 5, pp. 513–523, 1988.
24. J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger, "Tackling the poor assumptions of naive Bayes text classifiers," in Proc. 20th Int. Conf. Machine Learning (ICML), 2003, pp. 616–623.
25. C. Cortes and V. Vapnik, "Support-vector networks," Mach. Learn., vol. 20, no. 3, pp. 273–297, 1995.
26. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," J. Mach. Learn. Res., vol. 9, pp. 1871–1874, 2008.
27. L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.
28. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794.
29. F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.
30. F. Wilcoxon, "Individual comparisons by ranking methods," Biometrics Bulletin, vol. 1, no. 6, pp. 80–83, 1945.
31. J. Demšar, "Statistical comparisons of classifiers over multiple data sets," J. Mach. Learn. Res., vol. 7, pp. 1–30, 2006.

Author Profile

1. **Anas Saidu Ibrahim** is a postgraduate student pursuing a Master of Science in Computer Science at the Swarrnim School of Computing & IT, Swarrnim Startup & Innovation University, Gandhinagar, Gujarat, India. His research interests include machine learning, artificial intelligence, natural language processing and lightweight intelligent systems.
2. **Mahmud Umar** is a Master of Science student in Computer Science at the Swarrnim School of Computing & IT, Swarrnim Startup & Innovation University, Gandhinagar, Gujarat, India. His research interest is Retrieval-Augmented Generation (RAG).
3. **Auwalu Zubairu Abdullahi** is a Master of Science student in Biotechnology at the Department of Biotechnology, Swarrnim Startup & Innovation University, Gandhinagar, Gujarat, India.
4. **Abdullahi Isah Abdussalam** is affiliated with the Department of Biotechnology, Swarrnim Startup & Innovation University, Gandhinagar, Gujarat, India.
5. **Dr. Kalpesh R. Rakholia** is a faculty member and the research supervisor in the Swarrnim School of Computing & IT, Swarrnim Startup & Innovation University, Gandhinagar, Gujarat, India