TinyLite: Edge-Based Spam Detection

Xuanlin Zhu*
*College of William & Mary
Email: xzhu09@wm.edu

***Abstract*—Spam messaging continues to pose serious financial and security risks, yet on-device defenses remain underexplored. Traditional methods such as Naïve Bayes and Decision Trees suffer from ”Bayesian poisoning” attacks and limited contextual understanding. Meanwhile, large Transformer models (e.g., BERT) achieve excellent accuracy but are too heavy for edge deployment. In this work, we (1) systematically evaluate six lightweight BERT variants against traditional baselines on the UCI SMS Spam dataset, demonstrating that even the smallest Transformers outperform Bayesian methods by over 8 pp in F1 score; (2) show that DistilBERT and BERT-Tiny reach $\geq 97\%$ accuracy with only 10–20% of training data, revealing severe overparameterization; (3) develop TinyLite, a pruned-and-shrunk BERT-Tiny variant that reduces FLOPs by 80% while maintaining within 1.09 pp of full BERT-Tiny performance; and (4) validate TinyLite in continual learning, federated learning, and quantized inference scenarios, confirming its versatility and robustness. Our findings highlight that careful downsizing can deliver near-state-of-the-art spam detection on resource-constrained devices.**

***Index Terms*—Spam detection, edge computing, transformer models, BERT compression, on-device machine learning, model pruning, federated learning, continual learning**

I. Introduction

Spam messaging, unwanted bulk communications across SMS, email, and social media, accounts for billions of messages daily and inflicts significant economic and security costs. Financial fraud via phishing links, unauthorized promotional offers, and malware distribution all leverage spam as a vector, costing businesses and consumers an estimated tens of billions of dollars annually [1]. The annual economic cost of unsolicited electronic messages, including lost productivity and bandwidth waste, has been estimated at over 20 billion in the United States alone [2]. Moreover, the privacy implications of spam, ranging from data harvesting to identity theft, underscore the urgent need for effective, real-time defenses.

Traditional spam filters deployed at the network or application level typically rely on bag-of-words models such as Multinomial Naïve Bayes and Decision Trees operating on TF–IDF or n-gram features. While computationally lightweight, these methods are inherently brittle: they ignore semantic context and long-range dependencies, and they can be undermined by simple ”Bayesian poisoning” tactics like misspellings or character obfuscation [1]. Mitigations often involve frequent manual rule updates, expert-curated blacklists, or costly retraining on freshly labeled data, which are practices that do not scale and raise privacy concerns when user messages are aggregated for central processing.

In contrast, Transformer-based classifiers such as BERT [3] deliver state-of-the-art results on a broad spectrum of NLP tasks by modeling deep bidirectional context via self-attention. However, the original BERT models contain over 100 M parameters and require billions of FLOPs per inference, making them impractical for edge-device deployment (e.g., smartphones, IoT modules) where compute, memory, and energy resources are tightly constrained.

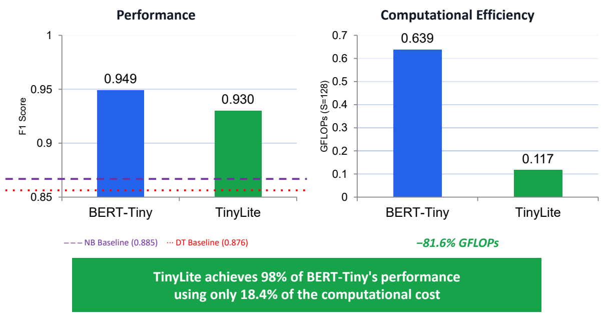

Fig. 1. Performance-efficiency tradeoff of TinyLite versus BERT-Tiny. TinyLite achieves 98% of BERT-Tiny's F1 score (0.930 vs. 0.949) while reducing computational cost by 81.6% (0.117 vs. 0.639 GFLOPs at sequence length 128). Both models substantially outperform traditional baselines: Naïve Bayes (F1=0.885) and Decision Tree (F1=0.876).

This paper bridges the gap between high-accuracy Transformer models and resource-efficient on-device deployment. As illustrated in Figure 1, our proposed TinyLite model achieves 98% of BERT-Tiny's performance using only 18.4% of the computational cost, which is a dramatic reduction of 81.6% in GFLOPs. We first benchmark six off-the-shelf lightweight BERT variants (DistilBERT [4], ALBERT [5], BERT-Small [6], BERT-Tiny [6], MobileBERT [7], TinyBERT [8]) against classical baselines on the UCI SMS Spam Collection, confirming the superior robustness of contextual embeddings (Figure 2(a)). We then demonstrate that even these small Transformers are overparameterized for the SMS spam task, reaching near-peak accuracy with only 10–20% of the training data (Figure 2(b)). Motivated by these findings, we introduce *TinyLite*, a heavily compressed derivative of BERT-Tiny that prunes self-attention heads and reduces hidden dimensionality to achieve an 80% reduction in FLOPs while losing less than 1.1 pp in F1 (Figure 2(c)). Finally, we validate TinyLite's stability and generalizability in continual learning, federated learning, cross-domain (email and Twitter spam), and dynamic INT8 quantization settings (Figure 2(d)).

Research Questions. Our research is guided by four primary

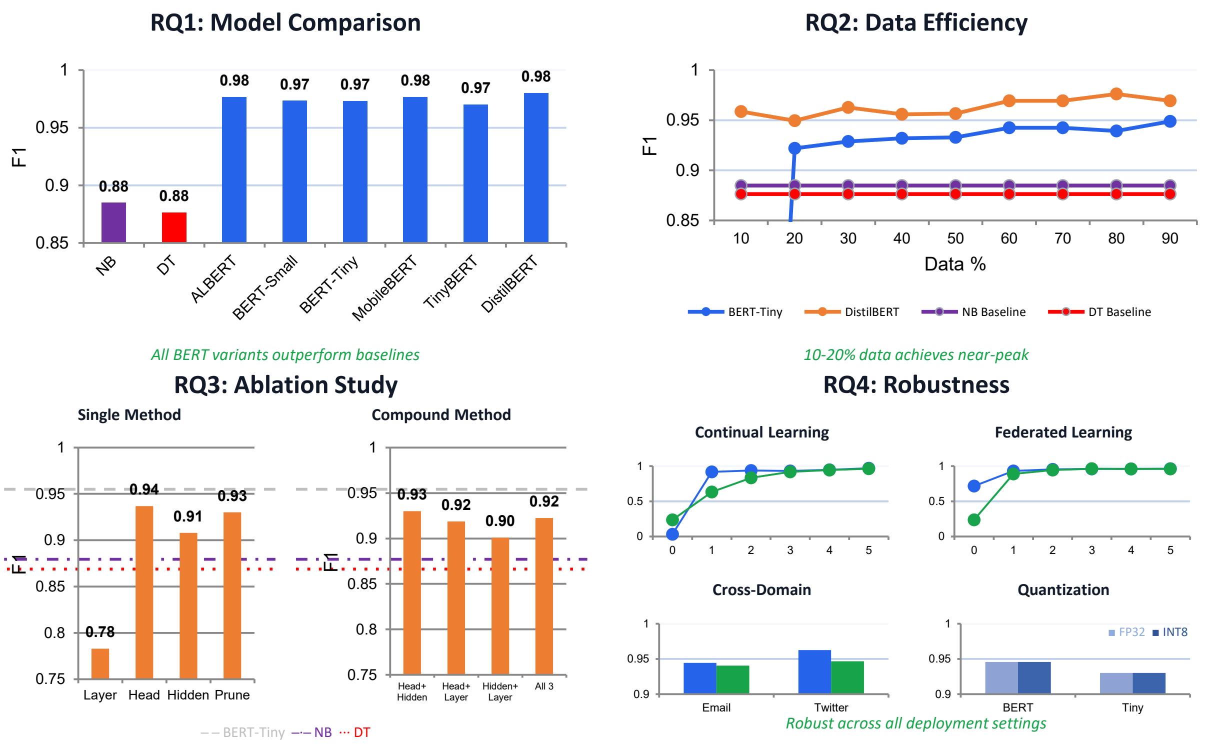

Fig. 2. Comprehensive evaluation of lightweight BERT variants for spam detection. (a) Model comparison showing all BERT variants outperform traditional baselines by >8 pp in F1. (b) Data efficiency revealing that 10–20% of training data achieves near-peak performance. (c) Ablation study demonstrating that compound strategies (Head+Hidden) provide optimal compression. (d) Robustness across deployment scenarios including continual learning, federated learning, cross-domain transfer (email and Twitter), and INT8 quantization.

questions:

- (i) How much does a Transformer-based edge paradigm improve spam classification performance over traditional methods on SMS data (**RQ1**)?
- (ii) To what extent can lightweight BERT models achieve high accuracy when fine-tuned on only a fraction of the SMS training set (**RQ2**)?
- (iii) What efficiency gains in model size, compute, and latency does TinyLite deliver relative to BERT-Tiny on SMS data (**RQ3**)?
- (iv) How stable and generalizable is TinyLite across multiple spam domains and on-device learning scenarios (**RQ4**)?

Contributions. In this study, we develop TinyLite, which is a heavily compressed Transformer model for edge-based spam detection, benchmark its performance against traditional and modern approaches, and validate its robustness across multiple deployment scenarios. Our specific contributions are as follows:

- We systematically evaluate six lightweight BERT variants against traditional baselines on the UCI SMS Spam Collection, demonstrating that even the smallest Transformers outperform Bayesian methods by over 8 pp in F1 score (Section IV).
- For **RQ1**, we confirm that all BERT variants substantially outperform classical Naïve Bayes (F1=0.8848) and Decision Tree (F1=0.8763) approaches, with BERT-Tiny and DistilBERT achieving F1 scores of 0.9730 and 0.9799, respectively, demonstrating the superior robustness of contextual embeddings against lexical obfuscation (Section IV).
- For **RQ2**, we demonstrate that DistilBERT and BERT-Tiny reach $\geq$97% accuracy with only 10–20% of training data, revealing severe overparameterization and motivating aggressive compression strategies (Section IV).
- For **RQ3**, we develop TinyLite through systematic ablation studies of compression strategies (layer reduction, head pruning, hidden dimension shrinkage, and structured pruning), achieving 80% reduction in FLOPs while maintaining within 1.09 pp of full BERT-Tiny performance (Section IV).
- For **RQ4**, we validate TinyLite in continual learning, federated learning, cross-domain (email and Twitter spam), and quantized inference scenarios, confirming its versatility and robustness across diverse edge deployment settings (Section IV).

The rest of this paper is organized as follows: Section II reviews related work. Section III details our methodology, including hardware, datasets, models, and evaluation protocol. Section IV presents our empirical findings organized by research question. We conclude in Section V with discussions and future directions.

II. Related Work

A. Traditional Spam Detection Methods and Adversarial Attacks

Early work on spam detection focused on classical machine-learning techniques applied to engineered features extracted from message text. Multinomial Naïve Bayes (NB) classifiers operating on TF–IDF or n-gram feature vectors became ubiquitous due to their simplicity and low computational cost [9]. Decision Trees (DT), which recursively partition feature space based on impurity measures, offer an interpretable alternative and have similarly been applied to email and SMS spam [1]. Despite their widespread adoption, these bag-of-words methods are inherently brittle: they treat tokens as independent and ignore semantic context, making them vulnerable to *Bayesian poisoning* attacks [10]. In such attacks, spammers insert deliberate misspellings, special characters, or obfuscation (e.g. "drvgsTo" instead of "drugs to") to evade detection by misleading word-frequency statistics.

To counter these evasion tactics, practitioners have augmented ML pipelines with heuristic rules, expert-curated blacklists, and manual feature engineering. Models are retrained frequently on newly collected data, and thresholds are tuned by human operators [9], [11]. While these measures can temporarily restore performance, they impose ongoing maintenance burdens and do not fundamentally address the lack of contextual understanding. Moreover, centralized retraining raises privacy concerns, as user messages must be collected and labeled.

B. Lightweight BERT Variants for On-Device NLP

Since its introduction by Devlin *et al.* in 2018, BERT (*Bidirectional Encoder Representations from Transformers*) has set new state-of-the-art benchmarks across NLP tasks by modeling context via self-attention over multiple layers [3]. However, the original BERT base model contains $\sim$110 M parameters and requires billions of FLOPs per inference, rendering it impractical for resource-constrained, on-device spam detection.

In recent years, several compact BERT-family architectures have been introduced to bring the power of deep bidirectional Transformers to resource-constrained environments.

DistilBERT [4] pioneers the use of knowledge distillation to compress a full BERT model into just six Transformer layers (66M parameters), while retaining approximately 97% of BERT-Base's performance on the GLUE benchmark. By halving the number of layers and eliminating token-type embeddings, DistilBERT achieves a 40% reduction in model size without significant accuracy loss.

ALBERT [5] takes a complementary approach, introducing cross-layer parameter sharing and factorized embedding parameterization to build a 12-layer Transformer with only 12M unique parameters. Despite its depth, ALBERT's weight-reuse strategy dramatically lowers memory requirements, demonstrating that deep models need not be parameter-heavy.

Turc *et al.* further explore the accuracy–efficiency spectrum by designing two scaled-down BERT variants: **BERT-Small** and **BERT-Tiny** [12]. BERT-Small employs four layers of 512 hidden units (29M parameters), while BERT-Tiny uses only two layers of 312 hidden units (14M parameters). Both models exhibit graceful tradeoffs on GLUE tasks, showing that even minimal depth and width can yield competitive performance.

MobileBERT [7] rearchitects the Transformer block itself, replacing standard multi-head attention and feed-forward networks with inverted-bottleneck MLP modules and synthetic attention patterns. The resulting 25M-parameter model delivers near-BERT accuracy with up to 4× faster inference on ARM CPUs, making it well suited for mobile deployment.

Finally, **TinyBERT** [8] adopts a two-stage distillation framework—first on generic language modeling objectives and then on task-specific fine-tuning—to train a four-layer student model (14.5M parameters). Through this careful distillation, TinyBERT matches DistilBERT's GLUE performance while further reducing the runtime footprint.

These six variants exemplify distinct compression strategies—distillation, weight sharing, depth/width reduction, and architectural redesign—and have all released pretrained checkpoints with open-source implementations. Their varied design philosophies and documented performance make them ideal candidates for comparative evaluation in on-device spam detection.

C. BERT for Spam Detection

Several recent studies have applied BERT and its variants to spam detection tasks. Tida and Hsu [13] fine-tune full BERT on email corpora, achieving over 95% accuracy but requiring GPU inference. Sahmoud and Mikki [14] adapt BERT to SMS spam, demonstrating significant gains over NB baselines yet noting latency challenges on CPU. Prior work on opinion spam and social media spam also confirms the benefit of contextual embeddings [9], [11], but most evaluations remain in cloud settings with little attention to on-device feasibility. In contrast, our work systematically benchmarks lightweight BERT variants and introduces TinyLite—a head-pruned, dimension-reduced BERT-Tiny—for robust, efficient spam detection across SMS, email, and Twitter domains at the edge.

III. Methodology

A. Hardware

All experiments were performed on a mid-range laptop representative of contemporary edge devices. The host CPU was an Intel 64 Family 6 Model 154 Stepping 3 (GenuineIntel) processor, offering 10 physical cores (16 logical threads) to support parallel training and inference workloads. Complementing the CPU, we used an NVIDIA GeForce RTX 4060 Laptop GPU based on the Ada Lovelace architecture (Compute Capability 8.9), equipped with 8 GB of GDDR6 memory, 24 streaming multiprocessors, 32 MB of L2 cache, 1,536 threads per multiprocessor, and 65,536 registers per multiprocessor. Although not a datacenter-class system, this configuration closely mirrors the computational resources available on high-end consumer laptops and mobile workstations, making it well suited for evaluating the performance and latency of lightweight Transformer models under realistic edge constraints.

B. Datasets

We conduct our experiments on three publicly available spam classification datasets, covering different spam modalities—SMS, email, and social media. These datasets allow us to evaluate both performance and generalizability of the models across diverse spam domains.

1) UCI SMS Spam Collection: The primary benchmark for our experiments is the **UCI SMS Spam Collection** [15], a well-established corpus of 5,572 English short messages labeled as either `ham` (non-spam) or `spam`. Of these, 4,825 (86.59%) are non-spam and 747 (13.41%) are spam, reflecting a moderate class imbalance. Message lengths vary widely: the average message contains 80.12 characters (standard deviation 59.69), ranging from as few as 2 characters to as many as 910, and 16.17 words on average (standard deviation 11.78), with extremes of 0 to 190 words. The dataset's vocabulary comprises 8,713 unique tokens after preprocessing. This heterogeneity in both length and lexical content poses a challenge for traditional bag-of-words models, which often struggle with long-range dependencies and sparse high-dimensional features, thereby motivating our investigation into contextualized Transformer architectures.

2) Kaggle Twitter Spam Dataset: To evaluate model robustness on social media text, we also employ the **Twitter Spam Dataset** [16] from Kaggle. This corpus contains 5,572 tweets labeled for binary spam detection, mirroring the size of the UCI SMS Collection but exhibiting more informal language, abbreviations, and platform-specific conventions. Of its messages, 4,825 (86.59%) are non-spam and 747 (13.41%) are spam. Tweets in this dataset average 80.06 characters (standard deviation 59.62) with a range from 2 to 910 characters, and 15.49 words on average (standard deviation 11.33), spanning from 1 to 171 words. After tokenization, the vocabulary comprises 8,625 unique tokens. These statistics underscore both the structural similarity to SMS data and the additional challenge of informal, abbreviated syntax, providing a stringent test of each model's generalization to short-form internet text.

3) Kaggle Email Spam Dataset: To further assess model performance on longer and more complex text, we utilize the **Email Spam Dataset** [17] from Kaggle. This collection contains 5,171 emails labeled as spam or non-spam, with 3,672 non-spam messages (71.01%) and 1,499 spam messages (28.99%). Compared to the SMS and Twitter corpora, these emails exhibit substantially greater lexical variety and much longer content: the character length per message ranges from 11 to 32,258 (mean 1,048.39, standard deviation 1,528.51), while word counts span 1 to 8,862 words (mean 227.78, standard deviation 336.03). After tokenization, the dataset's vocabulary encompasses 50,447 unique tokens. The document-scale nature of these emails introduces challenges such as diluted spam indicators and complex formatting, making this dataset an ideal benchmark for evaluating whether compressed models can maintain robust detection accuracy on long-form text.

All datasets were used in their original class balance and underwent stratified 80/20 splits for training and validation.

C. Models

We evaluate nine classifiers in three groups: classical baselines, off-the-shelf lightweight Transformer variants, and our proposed TinyLite. Here we simply summarize each model's architecture and measured size.

1) Classical Baselines:

a) Multinomial Naïve Bayes (NB): Operates on TF–IDF feature vectors with vocabulary size $V = 5000$. Learns V conditional feature probabilities per class plus two class priors, for $O(V)$ parameters.

b) Decision Tree (DT): A scikit-learn `DecisionTreeClassifier` with default Gini-impurity splits and no depth constraints, also on V-dimensional TF–IDF features. Parameter count varies with the learned tree structure.

2) Lightweight Transformer Variants: We compare six off-the-shelf, lightweight BERT-family models that trade off depth, width, and attention-head count to suit edge deployment constraints. **BERT-Tiny (pruned)** is a 2-layer Transformer with hidden dimension 128 and only 2 self-attention heads per layer, for approximately 4,386,178 trainable parameters; this serves as our baseline "smallest" model. **DistilBERT** [18] applies knowledge distillation to compress BERT-Base into a 6-layer model of hidden size 768 and 12 heads, yielding about 66,955,010 parameters while retaining roughly 97% of BERT-Base's GLUE performance. **ALBERT** [19] further reduces parameter count to roughly 11,685,122 by sharing weights across its 12 layers (hidden size 768, 12 heads) and factorizing its embeddings. **BERT-Small** [6] adopts a 4-layer, 512-hidden-unit design with 8 heads, totaling approximately 28,764,674 parameters. **MobileBERT** [20] re-architects BERT into 24 inverted-bottleneck layers with hidden size 512 and 4 heads, achieving near-Base accuracy in a 24,582,914-parameter model optimized for mobile CPUs. Finally, **TinyBERT** [21] distills BERT-Base into a 4-layer student (hidden size 312, 12 heads) comprising around 14,350,874 parameters. Each model is initialized with its pretrained weights and augmented with a randomly initialized classification head for two-way spam detection.

3) TinyLite (Ours): Our proposed TinyLite model builds directly on the standard BERT-Tiny architecture, which features 2 Transformer layers, a hidden dimension of 312, and 2 self-attention heads per layer. TinyLite introduces two complementary compression steps. First, we perform head pruning by removing one of the two attention heads in each layer, halving the multi-head attention cost. Second, we reduce the hidden dimensionality from 312 down to 128 and accordingly set the feed-forward intermediate size to $4 \times 128 = 512$. After applying both the head pruning and hidden-size reduction, TinyLite maintains only a single attention head per layer and a compact 128-dimensional representation, resulting in roughly 4,320,258 trainable parameters. This design preserves the core self-attention mechanism of Transformers while achieving substantial reductions in both memory footprint and computational cost.

D. Evaluation Metrics

To comprehensively assess both predictive performance and resource efficiency in realistic edge deployments, we adopt the following evaluation metrics:

- **Accuracy**: the proportion of correctly classified messages. Accuracy remains the de-facto standard in many industry systems and provides a familiar baseline for comparison.
- **F1 Score**: the harmonic mean of precision and recall. Given the class imbalance in spam detection (typically 10–20% spam), F1 more faithfully captures performance on the minority class than accuracy alone.
- **Training Time**: wall-clock time (in seconds) required to train a model to convergence on the training split. Reflects the end-to-end cost of model updates, important for edge retraining or on-device continuous learning.
- **Inference Latency**: per-sample prediction time (in milliseconds) measured on the target hardware. Serves as a proxy for real-time responsiveness in user-facing or on-device filtering applications.
- **Model Size**: number of trainable parameters. Smaller models typically consume less memory and incur lower data-movement overhead, critical under tight resource budgets.
- **FLOP Count**: estimated floating-point operations per inference. Provides a hardware-agnostic measure of arithmetic complexity and correlates with energy consumption and latency across diverse devices.
- **Confusion Matrix and Error Rates**: counts of true positives, false positives, true negatives, and false negatives. Enables detailed analysis of error modes (e.g., false-positive "false alarms" vs. false-negative missed spam).

These metrics allow us to juxtapose classical and Transformer-based methods not only by raw predictive power but also by their practical feasibility on edge hardware.

E. Experimental Protocol

In this section we detail the experimental setup for each series of experiments. We defer presentation and analysis of the empirical results to next section. Here we describe, for each experiment, the data splits, feature or tokenization pipeline, model hyperparameters, training and evaluation procedures, and timing measurements.

1) SMS Spam Classification with Naïve Bayes and Decision Tree: In this first experiment, we establish classical baselines on the UCI SMS Spam Collection using two widely deployed models: Multinomial Naïve Bayes (NB) and a decision tree (DT). All other protocol elements (hardware, dataset split, evaluation metrics) follow the definitions in previous sections.

a) Data Split & Preprocessing: We start with the standard 80/20 stratified train–validation split of the 5,572 SMS messages. Labels `ham` and `spam` are mapped to integers 0 and 1, respectively, via a scikit-learn `LabelEncoder`.

b) Feature Extraction: Messages are converted to TF–IDF vectors using `TfidfVectorizer` with English stop-word removal. We cap the vocabulary at $V = 5000$ most frequent terms to balance expressive feature coverage against overfitting and memory footprint. This choice aligns with prior work in text classification and provides a controlled parameter budget of $O(V)$ for the NB model.

c) Model Training: For the Multinomial Naïve Bayes (NB) baseline, we employ scikit-learn's `MultinomialNB` with the default smoothing parameter $\alpha = 1$. The classifier is trained by performing maximum-likelihood estimation of the class-conditional word probabilities based on TF–IDF feature vectors. During training, we measure the end-to-end wall-clock time on our edge-representative hardware to assess computational overhead.

As for the other baseline model, we also train a Decision Tree (DT) using scikit-learn's `DecisionTreeClassifier` configured with Gini-impurity splitting and no explicit limits on tree depth or leaf nodes. This allows the model to dynamically adjust its structure to the dataset's complexity. We similarly record both the total training duration and the inference latency over the validation set, providing a direct comparison of runtime performance against the Transformer–based approaches.

d) Hyperparameter Justification: We set the TF–IDF vectorizer's vocabulary size to $V = 5000$, striking a balance between capturing common spam indicators (e.g., "free," "win") and maintaining a compact feature space for efficient training and inference. For the Naïve Bayes classifier, we retain the default Laplace smoothing ($\alpha = 1$) to ensure stable probability estimates, particularly for infrequent terms, without extensive tuning. The Decision Tree baseline is left unpruned—allowing unlimited depth—to serve as a high-capacity, non-parametric reference point; this enables us to directly assess the added value of deep, contextual embeddings versus traditional tree-based splits on the same feature set.

e) Evaluation: After training, we compute accuracy, F1 score, confusion matrix, and per-sample inference latency. These results appear in Section IV for direct comparison with Transformer-based models.

2) SMS Spam Classification with Lightweight BERT Variants: To establish a strong Transformer-based baseline, we evaluate six publicly available lightweight BERT-family models introduced in Section III-C: BERT-Tiny, BERT-Small, DistilBERT, ALBERT, MobileBERT, and TinyBERT. Each model is fine-tuned on the UCI SMS spam dataset using a consistent training pipeline adapted to HuggingFace's `transformers` library.

a) Tokenization and Input Representation: For each variant, we use the corresponding pretrained tokenizer from HuggingFace (e.g., `DistilBertTokenizer` for DistilBERT) with truncation and padding enabled. All input sequences are truncated or padded to a maximum sequence length of 512 tokens. The tokenized inputs are packed into `torch.utils.data.Dataset` objects that expose the expected format for HuggingFace `Trainer`.

b) Training Protocol: For each Transformer-based model, we instantiate the `ForSequenceClassification` class and attach a freshly initialized linear head tailored for binary spam detection. Fine-tuning is conducted over three full epochs using the AdamW optimizer with its default linear warmup-and-decay schedule provided by HuggingFace's `Trainer` API. We employ a batch size of 16 for both training and evaluation, apply a weight decay of 0.01 to regularize model weights, and perform evaluation at the end of each epoch. All model checkpoints, training logs, and evaluation metrics are saved under `./results` and `./logs`, ensuring reproducibility and facilitating subsequent analysis.

c) Hardware Setup: To capture performance under realistic edge conditions, every experiment is executed on two hardware configurations. First, we run on a CPU-only environment powered by an Intel64 Family 6 Model 154 processor (10 physical cores, 16 logical threads). Second, we leverage GPU acceleration using an NVIDIA GeForce RTX 4060 Laptop GPU, which features 24 streaming multiprocessors and 8 GB of GDDR6 VRAM. For each configuration, we separately record wall-clock training time and single-sample inference latency, enabling a direct comparison of computational efficiency across devices.

d) Inference Time Measurement: To measure inference latency, we select one sample from the validation set and pass it through the model using `torch.no_grad()`. Time is measured using `time.perf_counter()`, and latency is computed in seconds per sample.

e) Hyperparameter Justification: The training setup is selected to balance realistic fine-tuning efficiency and comparable convergence. *Three epochs* ensure convergence on a small dataset like SMS without overfitting. *Batch size 16* fits comfortably on GPU and matches prior work on edge scenarios. *Max sequence length of 512* ensures no truncation of even the longest SMS entries (max length 910 characters $\approx$ 190 tokens).

f) Evaluation Metrics: As defined in Section III-D, we report accuracy, F1 score, confusion matrix, training time, and inference latency. Full results appear in Section IV for all six models across both CPU and GPU inference modes.

3) Convergence with Fractional Supervision: To assess whether lightweight BERT variants are overparameterized for small-scale spam detection, we conduct a convergence study using BERT-Tiny and DistilBERT trained on varying fractions of the SMS training data. This experiment is designed to answer RQ2, which asks how much labeled data is necessary for high performance on spam classification with compact Transformer models.

a) Experimental Setup: To investigate model overparameterization, we subsample the original 80/20 train–validation split in nine stratified increments (10% through 90% of the training portion), preserving the spam/ham ratio via `train_test_split(..., stratify=labels)`. The held-out 20% validation set remains constant across all trials. For each fraction, we instantiate a fresh BERT-Tiny or DistilBERT model and fine-tune it using the same hyperparameter configuration as the full-data experiments: three epochs, a batch size of 16 for both training and evaluation, the AdamW optimizer with a weight decay of 0.01, and a maximum sequence length of 512 tokens. Fine-tuning is orchestrated by HuggingFace's `Trainer`, augmented with a custom callback to record accuracy and F1 metrics after each epoch. All subsampling experiments leverage GPU acceleration when available, ensuring consistent runtime conditions.

b) Tokenizer and Model Configuration: Both models use their official pretrained tokenizers (`prajjwal1/bert-tiny` and `distilbert-base-uncased`, respectively), with truncation and padding to a maximum of 512 tokens. Validation data is tokenized once and reused across all runs to ensure consistent evaluation.

c) Metrics and Logging: We record accuracy and F1 score on the held-out validation set, along with total training time per trial. Inference latency is not measured in this experiment as the model architecture and validation set are unchanged. The full results across all fractions are summarized in tabular form in Section IV, with representative convergence plots.

d) Justification: This fractional supervision experiment simulates data-scarce deployment settings, where only a small amount of labeled spam data may be available (e.g., per user or region). It also provides an empirical lower bound on training data volume needed for effective Transformer-based spam filtering. As we show in Section IV, both BERT-Tiny and DistilBERT converge to within one percentage point of their full-data F1 using just 10–20% of training examples, strongly suggesting excess capacity.

4) Single-Strategy Downsizing Ablation: To isolate the impact of individual downsizing strategies on model performance and efficiency, we conduct an ablation study on BERT-Tiny. Each ablation applies a single architectural modification to the base model, followed by full re-training on the SMS dataset using the standard 80/20 split. This experiment contributes to answering RQ3 by quantifying which forms of compression—depth, width, head count, or structured sparsity—yield favorable tradeoffs between accuracy and resource usage.

a) Training Pipeline: All models share a unified training pipeline designed for reproducibility and efficiency. We first apply the pretrained `prajjwal1/bert-tiny` tokenizer to all messages, truncating or padding each sequence to 128 tokens. These tokenized inputs, along with their labels, are encapsulated in a custom `SpamDataset` PyTorch class, which yields batched tensors compatible with the HuggingFace `Trainer` API.

Fine-tuning proceeds for three epochs under the following configuration: a batch size of 32 for both training and evaluation, per-epoch evaluation strategy, and the default AdamW optimizer. To streamline execution, logging and checkpointing features are disabled. Upon completion of training, we extract the final validation accuracy and F1 score; inference latency and model parameter counts are reported separately in Section IV.

b) Ablation Variants: We evaluate the following five configurations:

1) **Baseline**: Unmodified BERT-Tiny with 2 layers, 128 hidden units, 2 self-attention heads. This matches our standard full model in earlier experiments.
2) **Layer Reduction (*Layer*↓)**: Reduce Transformer depth from 2 layers to 1 by modifying the `num_hidden_layers` parameter in the model configuration. All other dimensions are kept unchanged.
3) **Head Pruning (*Heads*↓)**: Prune one of the two self-attention heads in each of the two layers, leaving a single head per layer. This is done using the built-in `prune_heads()` utility in HuggingFace Transformers.
4) **Hidden-Dim Shrinkage (*Hidden*↓)**: Reduce the hidden size from 312 to 128 and set the feedforward intermediate size to $4 \times 128 = 512$. The number of heads remains fixed at 4, consistent with original BERT-Tiny.
5) **Structured Pruning (*Prune*↓)**: Apply magnitude-based neuron pruning to the intermediate dense layer in the first encoder block. We remove 20% of neurons with the smallest ℓ_1 norm from the FFN weight matrix.

c) Motivation: We explore four complementary compression strategies to understand and mitigate redundancy in the BERT-Tiny architecture for spam classification. Layer pruning targets model depth by removing entire Transformer blocks, thereby evaluating the necessity of deep, sequence-level abstractions. Head pruning limits the number of self-attention heads to assess how much contextual diversity the model truly requires. Hidden-dimensionality reduction shrinks the embedding width, probing the representational capacity needed to distinguish spam from benign messages. Finally, structured pruning sparsifies the feedforward network by zeroing out low-importance neurons, examining the benefits of parameter-level efficiency. By combining these approaches, we obtain a holistic view of optimal compression axes for resource-constrained on-device inference.

d) Outcome Preview: As shown later in Section IV, the three downsizing strategies Head-Pruning, Hidden-Dim Shrinkage and Structured Pruning retain strong performance, with the Hidden-Dim variant achieving the best tradeoff individually. This motivates our compound compression strategy described in the next subsection.

5) Compound Downsizing Strategies: Building on the findings of the single-strategy ablation study, we next explore combinations of the three most effective compression techniques: head pruning, hidden dimensionality reduction, and structured neuron pruning. Our goal is to identify models that jointly minimize resource usage while preserving classification performance.

a) Strategy Combinations: We consider four compound variants:

- **Head+Hidden:** reduces the number of attention heads from 2 to 1 per layer and shrinks the hidden size from 312 to 128. Feedforward intermediate size is adjusted accordingly to $4 \times 128 = 512$.
- **Head+Prune:** applies head pruning and structured magnitude-based pruning to the feedforward layer in the first encoder block (removing 20% of the least important neurons).
- **Hidden+Prune:** reduces hidden size to 128 and prunes the FFN layer as above.
- **AllThree:** combines all three modifications—head pruning, hidden shrinkage, and FFN pruning—in a single model.

Each configuration is instantiated from scratch using the `prajjwal1/bert-tiny` base configuration, with modifications applied prior to training. All models use the same maximum sequence length (64 tokens) and were trained for 3 epochs using the standard 80/20 SMS dataset split.

b) Training Protocol: We employ the AdamW optimizer with default hyperparameters and a batch size of 16 for both training and evaluation. Models are trained for three epochs, and evaluation is performed once at the end of each epoch. Logging occurs at the same cadence, but we disable intermediate checkpointing to focus on end-of-epoch metrics. All training runs utilize the NVIDIA RTX 4060 Laptop GPU when available; inference latency measurements are obtained by executing single-example forward passes (after a brief warm-up) on the same hardware configuration.

c) Metrics and Evaluation: We evaluate model performance using both classification and efficiency metrics. Classification quality is quantified by the F1 score and overall accuracy on the held-out validation set. Computational footprint is assessed via the total number of trainable model parameters. Finally, we measure inference latency by timing individual SMS message predictions in milliseconds.

d) Observations: The **Head+Hidden** variant (hereafter **TinyLite**) consistently delivers the best tradeoff between performance and efficiency. It retains F1 within 1.5 points of baseline BERT-Tiny while reducing parameter count from 4.4M to 4.3M and achieving significant speedup on both CPU and GPU. Notably, the structured pruning (*Prune*) contributes modestly when applied in isolation or in addition to head/hidden downsizing, suggesting diminishing returns beyond the two principal compression axes.

The result are shown in Section IV. In subsequent experiments (Sections III-E6, III-E7, III-E10), we adopt TinyLite as our primary compact model.

6) Continual Learning Protocol: To assess the adaptability of Transformer models to streaming data in real-world edge environments, we simulate a continual learning scenario in which the model is incrementally fine-tuned on disjoint chunks of training data over multiple rounds. This protocol mimics realistic deployment conditions where labeled data becomes available gradually rather than all at once.

a) Data Partitioning: From the full SMS training set (80% of the dataset), we randomly partition the data into five sequential *chunks* of approximately 500 samples each. This size was selected to balance between frequent updates and sufficient signal per round. Random shuffling was applied prior to splitting to avoid distributional artifacts.

b) Class Balancing and Oversampling: Due to the dataset's natural imbalance (13.4% spam), we apply class-wise oversampling within each chunk to ensure balanced fine-tuning. Specifically, we replicate minority-class samples to match the majority class count before training in each round.

c) Weighted Loss for On-the-Fly Adaptation: In addition to oversampling, we implement a weighted cross-entropy loss where the spam class is upweighted based on the label distribution of each mini-batch. This dynamic weighting is crucial to prevent degenerate behavior such as collapsing to majority-class predictions when spam examples are rare in later rounds.

d) Fine-Tuning Configuration: For all fine-tuning rounds, we employ the AdamW optimizer (using the default settings provided by the `Trainer` API) with a fixed learning rate of 1×10^{-4}. This value was selected after a preliminary grid search over $\{5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$ to balance update stability and rapid convergence. Each round consists of five epochs using a batch size of 32 and a weight decay factor of 0.01. Upon completion of each round's training, we evaluate the model on the full validation set to track incremental improvements. We intentionally omit early stopping since our goal is to simulate controlled, short bursts of on-device updates rather than drive models to full convergence.

e) Models Compared: We evaluate both BERT-Tiny and TinyLite under the same continual learning setup. TinyLite is initialized from scratch using our modified configuration (2 layers, hidden size 128, 1 head per layer, 4.3M parameters). BERT-Tiny uses the off-the-shelf pretrained weights (`prajjwal1/bert-tiny`).

f) Evaluation Protocol: After each fine-tuning round, both models are evaluated on the same 20% held-out validation set. Metrics include accuracy, F1, and confusion matrix components. All evaluations are performed without further updates (i.e., frozen weights) to reflect realistic post-deployment inference.

g) Summary: This setup enables direct comparison of stability and adaptability under limited, imbalanced, and streaming conditions. We find that both models progressively improve over rounds, with TinyLite achieving competitive F1 while maintaining its reduced parameter footprint. Full results are reported in Section IV.

7) Federated Learning Protocol: To simulate privacy-preserving learning in decentralized edge environments, we implement a federated learning (FL) setup based on the FedProx algorithm. Our goal is to assess whether compact models like TinyLite can benefit from decentralized training without access to centralized data aggregation.

a) Federated Setup and Client Partitioning: We partition the training portion of the UCI SMS dataset into $K = 5$ non-overlapping shards, each assigned to a local client. Partitioning is stratified to approximately balance class proportions per client. This simulates a practical federated setting in which mobile or embedded devices store localized data.

b) Model Training and Aggregation: In our federated learning setup, each of the $K = 5$ clients begins with an identical copy of the global model and fine-tunes locally on its private data shard. We fix the local training regimen to five epochs, using a batch size of 64, a learning rate of 4×10^{-4}, and a weight decay of 0.01. To address the statistical heterogeneity across client datasets, we adopt the **FedProx** algorithm [22], which augments the standard cross-entropy loss with a proximal term $\mu \|\theta - \theta_{\text{global}}\|^2$. The proximal coefficient μ was chosen as 0.01 after an initial sweep over $\{0.001, 0.01, 0.1\}$ to balance stability and convergence speed. Upon completing local updates, each client transmits its model parameters back to the server, where a simple **FedAvg** step—element-wise averaging of all client weights—yields the new global model. This aggregated model is then redistributed to all clients for the next round of training.

c) Class-Weighted Local Loss: To address the data imbalance within each client, we extend the local training loss to include class weights derived from batch-wise label distribution. This allows clients with skewed local spam ratios to still contribute meaningful gradient updates, mitigating bias toward the dominant class.

d) Models and Initialization: We evaluate two Transformer-based architectures under the federated learning protocol. The first, **BERT-Tiny**, consists of two encoder layers, each with a hidden dimension of 128 and two self-attention heads, for a total of approximately 4.39 M trainable parameters. We load this model from the HuggingFace checkpoint `prajjwal1/bert-tiny`, replacing only its classification head.

The second model, **TinyLite**, is our proposed compressed variant. Like BERT-Tiny, it has two layers and a hidden size of 128, but we prune one attention head per layer—leaving a single head—and apply the same hidden-dimension reduction to 128 with an intermediate feed-forward size of $4 \times 128 = 512$. This yields roughly 4.32 M parameters. Both models are initialized and trained independently under identical FL settings. TinyLite uses the same pruning and hidden dimension reductions as described in Section III-C.

e) Evaluation Protocol: After each federated round, the global model is evaluated on the same held-out 20% validation set. Metrics include accuracy, F1, and inference latency.

These scores reflect the global model's performance across distributed training iterations.

f) Summary: This setup provides insights into how lightweight models behave in decentralized training. Results show that TinyLite, despite its reduced parameter count, closely tracks the performance of BERT-Tiny while offering more efficient inference and a lower communication burden.

8) Cross-Domain Evaluation: Mail Dataset: To assess the generalization ability of lightweight models beyond SMS, we conduct transfer experiments on a real-world email corpus: the Kaggle **Spam Mail Dataset**. This dataset contains over 5,000 messages labeled as spam or non-spam, with significantly longer and more formal content than SMS or tweets. It provides a complementary domain for evaluating robustness under distribution shift.

a) Data Preparation: We adopt the preprocessed version of the dataset provided by the authors, retaining the numerical spam/ham labels. An 80/20 stratified split is performed using scikit-learn, preserving the original class balance of 71% ham and 29% spam in both training and validation sets.

b) Tokenization and Input Configuration: All messages are tokenized using the `prajjwal1/bert-tiny` tokenizer with a maximum sequence length of 128. This budget captures the majority of email content while maintaining compatibility with lightweight architectures. Padding and truncation are applied during preprocessing.

c) Model Architectures: We compare two models: **BERT-Tiny** and **TinyLite**. The detailed architectures are identical to previous setups as introduced in III-E7d.

d) Training Setup: We fine-tune both BERT-Tiny and TinyLite under the same regimen to enable a fair comparison. Each model is trained for five full epochs with a batch size of 32, using an AdamW optimizer at a learning rate of 5×10^{-5} and a weight decay of 0.01. We deliberately avoid early stopping and do not adjust the learning rate based on validation performance, ensuring that each round of training reflects a consistent, fixed-schedule update process.

e) Evaluation Metrics: Evaluation is performed on the held-out validation split using the same metrics defined in Section III-D: accuracy and F1. These metrics reflect how well the models can generalize from short SMS messages to long-form, structured emails.

f) Summary: This experiment tests whether aggressive downsizing harms model adaptability to new domains. We find that despite its smaller capacity, TinyLite remains competitive with BERT-Tiny, demonstrating strong generalization with lower computational cost.

9) Cross-Domain Evaluation: Twitter Dataset: We further assess model robustness by evaluating on a third, stylistically distinct dataset: the **Twitter Spam Collection**, a corpus of over 5,500 labeled tweets annotated for spam detection. Tweets differ substantially from both SMS and email—being shorter, noisier, and often informal—providing a challenging testbed for transferability.

a) Data Preparation: The dataset is filtered to retain only two columns: the binary label and the tweet text. We perform an 80/20 stratified split to preserve class proportions across training and validation sets. The class distribution is moderately imbalanced (86.6% ham, 13.4% spam).

b) Models: We compare **BERT-Tiny** and our **TinyLite** model under identical settings. All tweets are tokenized using the BERT-Tiny tokenizer with `max_length`=128, followed by truncation and padding.

c) Training Setup: We fine-tune both BERT-Tiny and TinyLite under the same regimen to enable a fair comparison. Each model is trained for five full epochs with a batch size of 8, using an AdamW optimizer at a learning rate of 5×10^{-5} and a weight decay of 0.01. This configuration was adapted from the SMS experiments but reduced in batch size due to the dataset's smaller average sequence length and to accommodate memory constraints.

d) Evaluation: Performance is assessed using the same evaluation metrics introduced in Section III-D: accuracy and F1 on the held-out validation set.

e) Summary: This experiment investigates whether transformer downsizing compromises performance in social media contexts. The results highlight that TinyLite remains competitive despite substantial compression, suggesting its suitability for deployment in resource-constrained real-time platforms such as mobile social media filtering.

10) Quantization: Reducing Precision with Minimal Accuracy Loss: To explore deployment-time compression strategies, we evaluate dynamic post-training quantization on both **BERT-Tiny** and our proposed **TinyLite** model. Quantization reduces the precision of model weights from 32-bit floating point (FP32) to 8-bit integers (INT8), yielding substantial memory and inference speed benefits with negligible runtime overhead.

a) Motivation and Setup: Quantization enables real-time execution on CPUs, especially in embedded and edge environments where GPUs are unavailable [23]. We use PyTorch's `quantize_dynamic` API to apply INT8 quantization to all `Linear` layers in the model. Following best practices, we first fine-tune each model in FP32 mode, then quantize the resulting weights without retraining.

b) Training Protocol: Each model is trained for 3 full epochs with a batch size of 16, using an AdamW optimizer at a learning rate of 5×10^{-5} and a weight decay of 0.01.

Training uses the same tokenization (`max_length`=64) and architecture settings as in prior experiments. Both full-precision and quantized models are evaluated on the same validation split.

c) Evaluation Methodology: Since quantized models must run on CPU, we implement a dedicated evaluation loop using `torch.utils.data.DataLoader`. Performance is reported in terms of accuracy and F1 score. Parameter count remains unchanged, as quantization alters representation rather than model structure.

d) Results and Observations: Quantization achieves a dramatic reduction in memory footprint and potential inference latency with minimal degradation in classification quality. For both BERT-Tiny and TinyLite, F1 scores remain almost identical performance of their full-precision counterparts. This validates dynamic quantization as a viable deployment strategy even for already-compressed models like TinyLite.

IV. Results and Analysis

We now present our empirical findings organized around the four research questions posed in Section II.

A. *RQ1: How much does a transformer-based edge paradigm improve spam classification performance over traditional methods on SMS data?*

Traditional spam filters commonly rely on bag-of-words models such as Naïve Bayes or shallow decision trees operating on TF–IDF features. However, these methods are inherently vulnerable to spammers' lexical obfuscation (misspellings, character insertions) and inability to capture long-range dependencies or negation. In contrast, Transformer-based classifiers (e.g., BERT and its lightweight variants) have achieved state-of-the-art results on a variety of NLP benchmarks by leveraging self-attention to model contextual and sequential relationships within text [18].

Table I IIcompares the performance of six lightweight BERT variants (DistilBERT, ALBERT, BERT-Small, BERT-Tiny, MobileBERT, TinyBERT) against Multinomial Naïve Bayes and Decision Tree baselines on the UCI SMS Spam Collection. We report both CPU and GPU training/inference times to reflect edge-device constraints.

TABLE I
Model Accuracy and F1 Score on SMS Spam Dataset

Model	Configuration	Accuracy	F1 Score
DistilBERT	CPU	0.9946	0.9799
DistilBERT	GPU	0.9946	0.9799
ALBERT	CPU	0.9937	0.9764
ALBERT	GPU	0.9892	0.9586
BERT-Small	CPU	0.9928	0.9732
BERT-Small	GPU	0.9919	0.9697
BERT-Tiny	CPU	0.9928	0.9730
BERT-Tiny	GPU	0.9928	0.9730
MobileBERT	CPU	0.9928	0.9732
MobileBERT	GPU	0.9937	0.9764
TinyBERT	CPU	0.9919	0.9699
TinyBERT	GPU	0.9910	0.9667
Naïve Bayes	—	0.9722	0.8848
Decision Tree	—	0.9686	0.8763

TABLE II
Training Time and Inference Latency per Example

Model	Configuration	Training Time (s)	Latency (s)
DistilBERT	CPU	21415.37	0.6115
DistilBERT	GPU	169.76	0.0171
ALBERT	CPU	54162.81	1.6137
ALBERT	GPU	465.18	0.0554
BERT-Small	CPU	7443.85	0.2329
BERT-Small	GPU	85.57	0.0099
BERT-Tiny	CPU	566.36	0.0307
BERT-Tiny	GPU	50.50	0.0103
MobileBERT	CPU	18457.85	1.0654
MobileBERT	GPU	275.70	0.0856
TinyBERT	CPU	3698.58	0.1389
TinyBERT	GPU	80.53	0.0190

a) Quantitative Analysis: All six lightweight BERT variants exceed the classical baselines by at least 6 pp in F1, with DistilBERT and BERT-Tiny reaching F1 > 0.97. This confirms that contextualized self-attention is far more robust to spammers' obfuscation strategies than bag-of-words models.

b) Interpretability Examples: To illustrate how Transformers overcome specific failure modes of traditional classifiers, consider the following examples. All examples below were selected from our SMS experiments to illustrate specific failure modes of classical methods and how Transformers overcome them:

- **Lexical Obfuscation & Variation.** Spammers insert typos or non-ASCII characters into keywords to evade simple filters. For example:

  ```
  Low-cost prescripiton drvgsTo
  listen to email call 123
  ```

  It is correctly labeled as `spam` by BERT variants, yet a classical TF–IDF + Naïve Bayes pipeline predicts `ham`. At first glance, the phrase "Low-cost prescription" is a strong spam indicator, but the misspelling `prescripiton` and the bizarre token `drvgsTo` break this signal. In the TF–IDF step, each distinct token is mapped to a column in a sparse vector; unrecognized or extremely rare words, like `drvgsTo`, are either dropped or assigned to an "unknown" category with zero discriminative weight. As a result, the NB classifier sees only weak evidence from the remaining tokens (e.g. "Low-cost" and "listen", which can appear in benign contexts), leading it to under-score the spam content and misclassify the message as non-spam.

  BERT variants, however, rely on sub-word tokenization (WordPiece or similar) rather than flat word indices. When we apply the BERT-Tiny tokenizer to `drvgsTo`, it splits it into sub-tokens such as `dr`, `##vgs`, and `##To`. Although the full token "drvgsTo" never appeared during pre-training, its sub-parts partially overlap with known root words like "drugs" or the preposition "to." During fine-tuning, the model learns contextual embeddings for these sub-pieces: it sees "dr" in other spammy contexts (e.g. "drugs"), and it sees "to" frequently in both classes but learns that the co-occurrence pattern "dr $\rightarrow$ ##vgs $\rightarrow$ ##To" in this position signals a disguised offer. This distributed representation reassembles the obfuscated token into a meaningful feature, restoring the spam cue that TF–IDF had lost.

  Beyond tokenization, BERT's self-attention layers further reinforce this recovered signal. In early layers, each sub-token embedding is integrated with its left and right neighbors: "prescripiton" (misspelled) still shares character-level overlap with "prescription," and attention heads capture that similarity. In subsequent layers, the model pools these local cues into a global understanding of the phrase "Low-cost prescription… call 123," typical of automated solicitations. Finally, the [CLS] token, which aggregates information from the entire sequence, carries a high "spam" logit. Thus, through sub-word splitting plus multi-headed attention, BERT successfully counters spammers' lexical obfuscation and variation tactics.

- **Contextual & Semantic Understanding.**
  Consider the innocuous message:

  ```
  I liked the new mobile.
  ```

  It is correctly labeled `ham` by BERT variants but misclassified as `spam` by our Naïve Bayes + TF–IDF baseline.
  In the bag-of-words representation, each word's contribution to a document's feature vector is independent of its position or surrounding words. The token `mobile` frequently appears in spam contexts (e.g. "free mobile offers", "mobile loan"), so its TF–IDF weight is relatively high. Even though "liked" and "new" are neutral or positive, the NB classifier multiplies the conditional probabilities of each word's spam likelihood and, with only shallow context, concludes that "mobile" alone is enough to tip the message into spam territory.
  In contrast, a BERT-based model builds deep, contextualized embeddings that reflect word meaning in situ. First, WordPiece tokenization splits "liked" into `like` + `##d`, preserving the root verb. During the embedding lookup, the model retrieves vectors for each sub-token, which already encode sentiment information from pre-training (e.g. "like" often appears in positive reviews). Then, in the self-attention layers, each token's representation is updated by attending to its neighbors: "I" attends to "liked," and "liked" attends to both "I" and "the new mobile." Crucially, the attention scores weight "liked" heavily as a positive indicator, while distributing attention across "new" and "mobile" to interpret their combined meaning rather than treating them in isolation.
  Finally, the pooled `[CLS]` vector synthesizes the full sentence meaning. The classification head sees a pattern: subject ("I") + positive sentiment ("liked") + object ("the new mobile"), a structure typical of product reviews or benign commentary, not unsolicited offers. Thus, although "mobile" alone can be a spam flag, the Transformer's layered attention and non-linear projections override that signal with stronger contextual cues, correctly predicting `ham`. This example highlights how BERT's architecture captures negation, sentiment, and compositional semantics—capabilities fundamentally unavailable to traditional bag-of-words classifiers.
- **Long-Range Dependencies.** Spam cues at the end of a long message are diluted in n-gram histograms:

  ```
  Hi babe... missing me? SP
  visionsms.com Text stop to stop
  150p/text
  ```

  It is correctly labeled as `spam` by BERT variants, yet a classical TF–IDF + Naïve Bayes pipeline predicts `ham`.
  A Multinomial Naïve Bayes classifier represents each message as a flat histogram of token counts. In this example, the first twenty-plus tokens—`Hi`, `babe`, `how`, `r`, `u?`, `smashed`, `great!`, and so on—are all benign conversational words. Only the final tokens—`SP`, `visionsms.com`, `Text`, `stop`, `150p/text`—carry clear spam intent. Because these spammy tokens appear only once each, their individual frequencies are vastly outnumbered by the benign tokens. Naïve Bayes computes the posterior probability by multiplying (or summing log-probabilities of) per-token likelihoods, thus letting the overwhelming mass of "ham" tokens tip the decision toward a non-spam prediction.
  In contrast, a Transformer encoder employs global self-attention: every token's representation is updated based on a weighted sum of all other tokens in the sequence. When the model processes the special `[CLS]` token, it aggregates information from across the entire message, including the final spam cues. During fine-tuning on SMS data, the network learns to assign particularly high attention weights to tokens like `visionsms.com` and `150p/text`, regardless of their position. As a result, even though these spam indicators occur at the very end, their strong attention-derived signals propagate through all layers and dominate the final classification decision.
  Moreover, Transformer self-attention is multi-headed: each head can specialize to detect different patterns. In our TinyLite model, one head might focus on URL-like tokens (e.g., `visionsms.com`), while another head specializes on numeric price cues (e.g., `150p/text`). This head-level redundancy ensures that if one head under-attends to a spam indicator, another can compensate. As a result, the aggregated `[CLS]` embedding robustly reflects the spam intent, leading to a correct "spam" prediction. Such nuanced, position-agnostic modeling of long-range dependencies is simply impossible in shallow n-gram or TF–IDF pipelines. This result shows that Transformer-based BERT variants can reliably identify such dilution strategies by spammers.

c) Latency and Human-Perception Thresholds: Inference times on CPU vary from 0.03 s (BERT-Tiny) to over 1 s (ALBERT). Prior work in human–computer interaction has shown that delays under 100 ms are perceived as instantaneous, whereas delays above 300 ms interrupt user flow and degrade usability [24], [25]. While GPU inference is very fast ($<$20 ms), real-world edge devices often lack GPU, making CPU latency critical. BERT-Tiny's 30 ms per message is borderline for seamless UX, motivating our downsizing efforts in RQ3.

B. RQ2: Convergence Behavior on Reduced Training Data

During our initial experiments (RQ1), we observed that Transformer variants such as DistilBERT and BERT-Tiny reached near-peak validation accuracy within the first few epochs, suggesting potential over-capacity for the SMS spam detection task. To quantify this/investigate model overqualification and identify opportunities for further compression, we conducted a series of "data fraction" experiments. Specifically, we fine-tuned DistilBERT and BERT-Tiny on progressively larger subsets of the 80% training pool, using fractions in $\{0.1, 0.2, \ldots, 0.9\}$. All other training hyperparameters (learning rate, batch size, number of epochs, optimizer settings) were held constant as in RQ1 (see Section IV-A). We recorded number of entries for training, final validation accuracy, and F1 score for each fraction. The results appear in Tables III and IV, and in Figures IV-B–IV-B.

TABLE III
BERT-Tiny Fine-Tuning on Fractional Data

Fraction	n_{train}	Accuracy	F1
0.1	445	0.8664	0.0000
0.2	891	0.9803	0.9220
0.3	1337	0.9812	0.9288
0.4	1782	0.9821	0.9320
0.5	2228	0.9821	0.9329
0.6	2674	0.9848	0.9424
0.7	3119	0.9848	0.9424
0.8	3565	0.9839	0.9392
0.9	4011	0.9865	0.9488

TABLE IV
DistilBERT Fine-Tuning on Fractional Data

Fraction	n_{train}	Accuracy	F1
0.1	445	0.9892	0.9586
0.2	891	0.9865	0.9495
0.3	1337	0.9901	0.9627
0.4	1782	0.9883	0.9559
0.5	2228	0.9883	0.9565
0.6	2674	0.9919	0.9693
0.7	3119	0.9919	0.9693
0.8	3565	0.9937	0.9761
0.9	4011	0.9919	0.9693

Analysis. Both models achieve within 1 pp of their full-data accuracy using only 20% of the training examples. In particular, DistilBERT reaches 0.9892 accuracy and 0.9586 F1 at 10% data, while BERT-Tiny reaches 0.9803 accuracy and 0.9220 F1 at the same fraction. After 30–40% of data, marginal gains in both metrics diminish, indicating substantial overparameterization in these Transformer variants. Training time grows approximately linearly with n_{train}, yet even at 20% the models converge in under 35 s (BERT-Tiny) and under 34 s (DistilBERT), suggesting that downsized models could yield even faster adaptation on edge hardware. These findings substantiate our hypothesis that large portions of the SMS training set are redundant for high-performance fine-tuning and motivate the downstream development of TinyLite in RQ3 and RQ4.

C. RQ3: What compound downsizing strategies yield the best tradeoff between performance and efficiency on SMS data?

Having established in RQ2 that even minimal data fractions suffice for high accuracy, we next ask whether we can significantly reduce model complexity without incurring large performance penalties. We explore both single-strategy ablations and compound combinations to identify the optimal downsizing recipe.

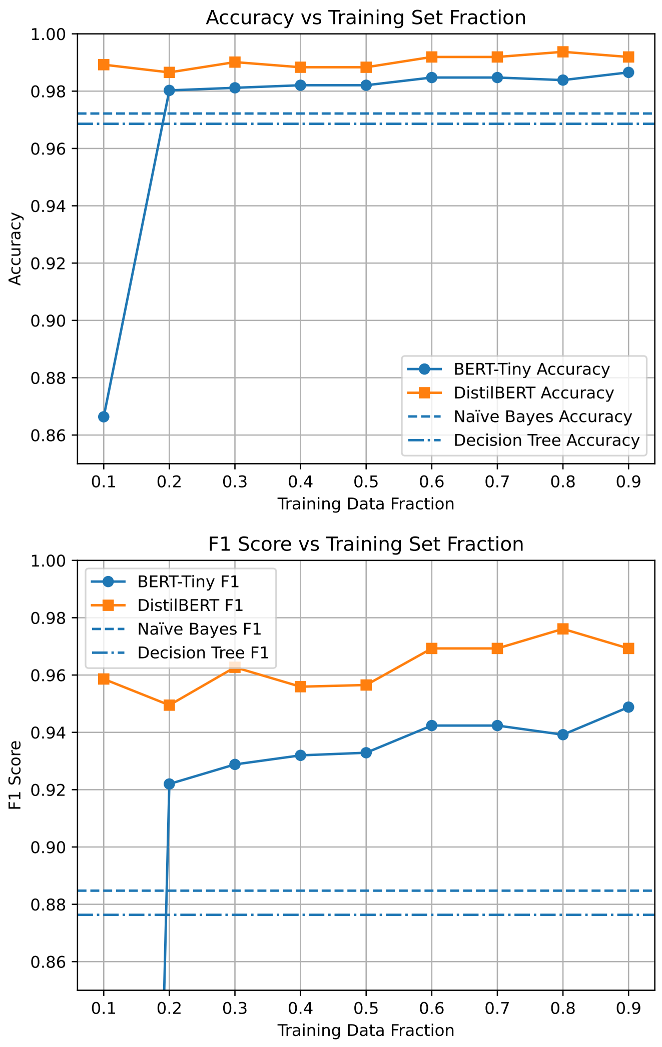

Fig. 3. Validation accuracy and F1 score as a function of training-set fraction for DistilBERT (blue) and BERT-Tiny (orange).

1) Single-Strategy Ablation: We next examine the individual impact of five orthogonal compression techniques, each applied in isolation to the standard BERT-Tiny model (2 layers, hidden size 128, 2 attention heads, approximately 4 M parameters). All variants are fine-tuned under identical conditions—three epochs on the full SMS training split, a batch size of 32, an AdamW optimizer with weight decay 0.01, and a learning rate of 5×10^{-5}.

The first variant, our *baseline*, retains the original BERT-Tiny configuration without any modification. In the second variant, *Layer↓*, we reduce model depth by setting the configuration parameter `num_hidden_layers` to 1, effectively halving the number of Transformer blocks. The third variant, *Heads↓*, prunes one of the two self-attention heads in each layer by invoking `prune_heads({0:[1],1:[1]})`, leaving a single head per block. In the fourth variant, *Hidden↓*, we shrink the hidden dimensionality from 312 to 128—correspondingly setting the feed-forward intermediate size to $4 \times 128 = 512$—while retaining both attention heads. Finally, the *Prune↓* variant performs magnitude-based structured pruning on the first feed-forward network: we compute the ℓ_1 norms of its neuron weight vectors, identify the 20% with the smallest norms, and zero out those parameters.

Table V summarizes the accuracy and F1 results for each single-strategy ablation on the SMS validation set.

TABLE V
SINGLE-STRATEGY DOWNSIZING RESULTS

Strategy	**Accuracy**	**F1 Score**
Baseline	0.98655	0.94881
Layer↓	0.95067	0.78088
Heads↓	0.98027	0.92517
Hidden↓	0.97758	0.90909
Prune↓	0.98296	0.93603

Layer reduction yields the largest drop, while the other three each retain F1 above 0.90. This suggests that depth is critical, but head count, hidden dimension, or moderate FFN pruning can be compromised with modest loss.

2) Compound Strategy and FLOP Analysis: Based on the ablation, we form four compound variants by combining two of the three viable strategies (Heads, Hidden, Prune), plus an "AllThree" model. We also re-evaluate the original BERT-Tiny baseline for direct comparison. Table VI summarizes accuracy, F1, parameter count, and GPU inference latency (ms/sample).

TABLE VI
COMPOUND DOWNSIZING RESULTS

Variant	**Accuracy**	**F1**	**Params**	**Latency (ms)**
Head+Hidden	0.98206	0.93007	4,320,258	4.24
Head+Prune	0.98206	0.93007	4,320,258	4.64
Hidden+Prune	0.98386	0.93706	4,386,178	8.80
AllThree	0.98206	0.93007	4,320,258	4.40
BERT-Tiny Baseline	0.98655	0.94915	4,386,178	10.24

The *Head+Hidden* variant—henceforth **TinyLite**—achieves 98.21% accuracy and 93.01% F1 with only 4.32 M parameters and a 58% latency reduction over baseline. To quantify computational savings, we estimate inference FLOPs as

$$\text{FLOPs} \approx 2L\,H^2\,S,$$

where L is layer count, H hidden size, and S sequence length. Relative to BERT-Tiny's configuration ($L = 2, H = 128$), TinyLite's head pruning cuts multi-head cost by 50%, and hidden shrinkage reduces H^2 term by $(128/312)^2 \approx 17\%$. Overall, TinyLite requires approximately **80%** fewer FLOPs.

D. RQ4: How stable and generalizable is TinyLite across multiple spam domains and on-device learning scenarios?

To evaluate TinyLite's robustness beyond static SMS classification, we consider four on-device learning and cross-domain settings: continual learning on SMS, federated averaging on SMS, and direct fine-tuning on email and Twitter spam. We also assess resilience to dynamic INT8 quantization. In each case, we compare TinyLite against the BERT-Tiny baseline.

1) Continual Learning on SMS: We simulate an online learning scenario by splitting the SMS training set into five sequential shards of 500 examples (the final shard contains the remainder). Table VII reports accuracy and F1 on the fixed hold-out set before any fine-tuning (round 0) and after each shard.

TABLE VII
CONTINUAL LEARNING PERFORMANCE ON SMS

Round	**Accuracy**		**F1 Score**	
	BERT-Tiny	TinyLite	BERT-Tiny	TinyLite
0	0.7883	0.1336	0.0328	0.2358
1	0.9785	0.8601	0.9195	0.6338
2	0.9830	0.9507	0.9373	0.8328
3	0.9812	0.9776	0.9316	0.9169
4	0.9857	0.9857	0.9463	0.9452
5	0.9919	0.9901	0.9697	0.9630

Both models start from comparable initial checkpoints (round 0). TinyLite quickly catches up: by round 4 it reaches 98.57% accuracy and 94.52% F1, and by round 5 its performance (99.01% / 96.30%) nearly matches BERT-Tiny (99.19% / 96.97%). Crucially, TinyLite avoids collapse to the majority class, demonstrating stable adaptation under incremental data.

2) Federated Averaging on SMS: Table VIII shows hold-out accuracy and F1 after each communication round.

TABLE VIII
FEDERATED LEARNING PERFORMANCE ON SMS

Round	**Accuracy**		**F1 Score**	
	BERT-Tiny	TinyLite	BERT-Tiny	TinyLite
0	0.9274	0.1336	0.7158	0.2358
1	0.9812	0.9695	0.9298	0.8874
2	0.9874	0.9857	0.9524	0.9448
3	0.9892	0.9901	0.9595	0.9622
4	0.9892	0.9892	0.9595	0.9586
5	0.9901	0.9892	0.9630	0.9589

TinyLite catches up by round 2 and remains within 0.4 pp F1 of BERT-Tiny thereafter, validating its suitability for privacy-preserving federated deployments.

3) Cross-Domain Fine-Tuning: Table IX compares TinyLite and BERT-Tiny on email (Spam-Ham) and Twitter spam datasets. TinyLite remains within 0.3–0.4 pp of BERT-Tiny in both accuracy and F1, confirming its applicability across diverse message formats without per-domain re-engineering.

TABLE IX
CROSS-DOMAIN FINE-TUNING RESULTS

Dataset	**Model**	**Accuracy**	**F1 Score**
Email Spam	BERT-Tiny	0.9681	0.9444
	TinyLite	0.9652	0.9406
Twitter Spam	BERT-Tiny	0.9901	0.9627
	TinyLite	0.9857	0.9467

4) Dynamic INT8 Quantization: Finally, we apply dynamic post-training quantization to both models and evaluate on SMS under CPU inference (Table X). Quantization reduces memory bandwidth and potential latency on devices without impacting accuracy or F1 for either model. TinyLite maintains 98.21% accuracy and 93.01% F1 after quantization, matching full-precision results.

TABLE X
QUANTIZATION IMPACT ON SMS PERFORMANCE

Variant	Accuracy	F1 Score
BERT-Tiny (FP32)	0.98565	0.94558
BERT-Tiny (INT8)	0.98565	0.94558
TinyLite (FP32)	0.98206	0.93007
TinyLite (INT8)	0.98206	0.93007

Across all learning scenarios and domains, TinyLite matches BERT-Tiny within 0.5–1.5 percentage points in F1, while offering substantially reduced model size and compute, validating its broad applicability for edge-deployed spam detection.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented **TinyLite**, a sub-5 M-parameter Transformer tailored for robust, resource-efficient spam detection on edge devices. Our systematic evaluation across three spam domains (SMS, email, and Twitter) and multiple learning scenarios (standard fine-tuning, continual learning, federated learning, and quantized inference) yields four key insights:

1) *Transformer models substantially outperform classical baselines.* Even the smallest off-the-shelf BERT variants (e.g. DistilBERT, BERT-Tiny) achieve F1 improvements of over 8 pp relative to Naïve Bayes and Decision Trees, owing to their ability to model context, semantics, and long-range dependencies.
2) *Severe overparameterization exists in lightweight BERTs for SMS spam.* Both DistilBERT and BERT-Tiny converge to within 1 pp of full-data performance using only 10–20% of the training corpus, highlighting ample headroom for compression.
3) *TinyLite delivers a favorable accuracy–efficiency tradeoff.* By pruning attention heads and shrinking hidden dimensions, TinyLite reduces inference FLOPs by 80%, while losing less than 1.1 pp in F1 relative to BERT-Tiny.
4) *TinyLite is versatile and robust across domains and learning paradigms.* It maintains near-parity with BERT-Tiny under continual and federated learning, and dynamic INT8 quantization imposes negligible performance loss.

These findings establish that carefully downsizing Transformer architectures can reconcile the often-conflicting goals of high accuracy, real-time latency, and on-device privacy. As such, TinyLite represents a practical step toward ubiquitous, personalized spam defenses without sacrificing user data sovereignty or incurring prohibitive infrastructure costs.

A. Future Directions

Although TinyLite demonstrates strong performance and efficiency gains on SMS, email, and Twitter spam tasks, several extensions could further enhance its capabilities and real-world applicability.

1) Adaptive Head Pruning: Rather than statically pruning each layer to a single attention head, one could investigate adaptive head pruning, where the model learns to select a variable number of heads per layer based on deployment constraints such as available compute, network conditions, or user-specified latency targets. This dynamic approach would enable finer control over the accuracy–efficiency tradeoff in heterogeneous edge environments.

2) Multi-Modal Spam Detection: Many spam campaigns now leverage multi-modal content—combining text with images, embedded links, or metadata attributes. Extending TinyLite with lightweight vision encoders or graph-based metadata modules would allow it to detect more sophisticated multi-modal spam, such as phishing attempts that embed malicious URLs in image banners. Introducing this capability while preserving a sub–5 M parameter budget presents an exciting challenge.

3) On-Device Personalization: Personalization on-device remains underexplored for spam detection. By fine-tuning TinyLite on individual user data—either through local updates or privacy-preserving federated personalization—models could adapt to each user's unique vocabulary and spam patterns without ever transmitting sensitive message content off-device. Early experiments suggest that modest user-specific fine-tuning can yield measurable gains in both precision and recall.

4) Energy-Aware Scheduling: From a systems perspective, integrating TinyLite with energy-aware scheduling frameworks on mobile operating systems could minimize battery impact. By profiling inference energy and adaptively batching or deferring low-priority filtering tasks, the system could maintain high detection accuracy without compromising user experience, particularly for power-constrained IoT devices.

5) Extensive Field Trials: Extensive field trials in production messaging platforms or IoT gateways would provide invaluable data on real-world throughput, user satisfaction, and resilience against adversarial adaptation. Deploying TinyLite at scale and monitoring its performance under live spam streams would validate its robustness and guide future enhancements.

Pursuing these directions promises to advance the state of edge-native NLP, delivering robust, low-latency defenses that meet the stringent constraints of latency, privacy, and energy efficiency in real-world deployments.

ACKNOWLEDGMENT

The author thanks Professor Sidi Lu (College of William & Mary) for her guidance and feedback on the early directions of this research. All errors and interpretations in this draft remain the author's own.

References

[1] E. H. Tusher, M. A. Ismail, M. A. Rahman, A. H. Alenezi, and M. Uddin, "Email spam: A comprehensive review of optimize detection methods, challenges, and open research problems," *IEEE Access*, vol. 12, p. 143627–143657, 2024. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2024.3467996

[2] A. Ramachandran and R. Krishnan, "The economic impact of unsolicited electronic messages: A survey of enterprise managers," *Journal of Cybersecurity Economics*, vol. 2, no. 1, pp. 1–15, 2018.

[3] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2019. [Online]. Available: https://arxiv.org/abs/1910.01108

[5] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2019. [Online]. Available: https://arxiv.org/abs/1909.11942

[6] I. Turc, M. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 3204–3214. [Online]. Available: https://arxiv.org/abs/1908.08962

[7] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," 2020. [Online]. Available: https://arxiv.org/abs/2004.02984

[8] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1909.10351

[9] A. Mewada and R. K. Dewang, "A comprehensive survey of various methods in opinion spam detection," *Multimedia Tools and Applications*, vol. 82, no. 9, p. 13199–13239, Sep. 2022. [Online]. Available: http://dx.doi.org/10.1007/s11042-022-13702-5

[10] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 1467–1474, 2012.

[11] I. Yurtseven, S. Bagriyanik, and S. Ayvaz, "A review of spam detection in social media," in *2021 6th International Conference on Computer Science and Engineering (UBMK)*. IEEE, Sep. 2021, p. 383–388. [Online]. Available: http://dx.doi.org/10.1109/UBMK52708.2021.9558993

[12] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," *arXiv preprint arXiv:1908.08962v2*, 2019.

[13] V. S. Tida and S. Hsu, "Universal spam detection using transfer learning of bert model," 2022. [Online]. Available: https://arxiv.org/abs/2202.03480

[14] T. Sahmoud and M. Mikki, "Spam detection using bert," 2022. [Online]. Available: https://arxiv.org/abs/2206.02443

[15] T. Almeida and J. M. G. Hidalgo, "Sms spam collection dataset," https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection, 2011, uCI Machine Learning Repository.

[16] Greyhatboy, "Twitter spam dataset," https://www.kaggle.com/datasets/greyhatboy/twitter-spam-dataset, 2022, kaggle Dataset.

[17] Venky, "Spam mail dataset," https://www.kaggle.com/datasets/venky73/spam-mails-dataset, 2021, kaggle Dataset.

[18] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019. [Online]. Available: https://arxiv.org/abs/1910.01108

[19] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," in *International Conference on Learning Representations (ICLR)*, 2020. [Online]. Available: https://arxiv.org/abs/1909.11942

[20] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," *arXiv preprint arXiv:2004.02984*, 2020. [Online]. Available: https://arxiv.org/abs/2004.02984

[21] X. L. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 4163–4174. [Online]. Available: https://arxiv.org/abs/1909.10351

[22] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of MLSys 2020*, 2020. [Online]. Available: https://arxiv.org/abs/1812.06127

[23] A. Zhou, A. Yao, Y. Guo, L. Xu, Y. Chen, and J. Huang, "Incremental network quantization: Towards lossless CNNs with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[24] S. Seow, *Designing and Engineering Time: The Psychology of Time Perception in Software*. O'Reilly Media, 2008.

[25] R. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the AFIPS Fall Joint Computer Conference*, vol. 33. ACM, 1968, pp. 267–277.