Quantized Transformer Language Model Implementations on Edge Devices

Mohammad Wali Ur Rahman*, Murad Mehrab Abrar*, Hunter Gibbons Copening†, Salim Hariri*, Sicong Shao§, Pratik Satam‡, and Soheil Salehi*

*Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721 USA
†Computer Science, University of Arizona, Tucson, AZ 85721 USA
‡Systems and Industrial Engineering, University of Arizona, Tucson, AZ 85721 USA
§Electrical Engineering and Computer Science, University of North Dakota, Grand Forks, ND 58202 USA
Email: {*mwrahman, *abrar, †huntercopening, *hariri, ‡pratiksatam, *ssalehi}@arizona.edu; §sicong.shao@und.edu

***Abstract*—Large-scale transformer-based models like the Bidirectional Encoder Representations from Transformers (BERT) are widely used for Natural Language Processing (NLP) applications, wherein these models are initially pre-trained with a large corpus with millions of parameters and then fine-tuned for a downstream NLP task. One of the major limitations of these large-scale models is that they cannot be deployed on resource-constrained devices due to their large model size and increased inference latency. In order to overcome these limitations, such large-scale models can be converted to an optimized FlatBuffer format, tailored for deployment on resource-constrained edge devices. Herein, we evaluate the performance of such FlatBuffer transformed MobileBERT models on three different edge devices, fine-tuned for Reputation analysis of English language tweets in the RepLab 2013 dataset. In addition, this study encompassed an evaluation of the deployed models, wherein their latency, performance, and resource efficiency were meticulously assessed. Our experiment results show that, compared to the original BERT large model, the converted and quantized MobileBERT models have 160× smaller footprints for a 4.1% drop in accuracy while analyzing at least one tweet per second on edge devices. Furthermore, our study highlights the privacy-preserving aspect of TinyML systems as all data is processed locally within a serverless environment.**

***Index Terms*—IoT, Natural Language Processing, Machine Learning, BERT, Reputation Polarity, Social Media, Embedded Systems, TinyML, Privacy.**

I. Introduction

Pre-trained large-scale Natural Language Processing (NLP) models have been exhibiting remarkable performance in most NLP tasks using transformer-based architectures. By stacking multiple encoder/decoder layers, combined with attention mechanism [27], these architectures are producing promising results in the field of NLP. Models such as BERT [11], RoBERTa [18], XLNet [33], and GPT-4 [22] have been increasingly popular in the commercial development of various smart AI systems to analyze audio/text input. These services will be integrated into mobile computing and Internet of Things (IoT) devices to improve user experience, making it imperative to deploy such NLP services on resource-constrained edge devices to improve the service response times [20].

However, these transformer-based NLP models are pre-trained using TensorFlow (or similar) end-to-end machine learning platform and they are optimized for classification accuracy, making them contain thousands of layers of neurons with a large number of optimization parameters. Such models are large in size and require significant memory for storage and processing. [11, 18, 33, 23]. Accommodation of such large-scale models in edge devices with smaller storage is a major challenge. In addition to the storage needs, the latency, and the computational cost also prove to be huge obstacles to the deployment of traditional machine learning (ML) models [28]. Due to the increased latency resulting from the constrained resources of edge devices, the conventional deep learning models often fail to meet the real-time requirements [20]. To address these challenges, TinyML has proven to be a promising candidate.

One of the main focuses of the field of TinyML is on developing and deploying ML models on resource-constrained, small, and low-power devices such as microcontrollers, sensors, and edge devices [30]. Traditionally IoT device services rely on sending data to a remote server for ML analysis (to provide services), adding performance delays, increasing the service's dependence on the availability and quality of the communication network, and posing security challenges, including those concerning user privacy [19]. Integration of TinyML-based models into the service allows the deployment of these ML-based services on the device itself or an edge node, mitigating the aforementioned challenges. TinyML allows for real-time data processing at the edge, enabling a more efficient and faster decision-making process, which can be crucial for some applications, such as autonomous systems, robotics, and industrial automation [25].

TensorFlow-Lite [7] is an example of a TinyML-based algorithm that is optimized for deployment on embedded devices [9]. It includes a number of features that make it well-suited for implementing TinyML, such as support for on-device ML, quantization and pruning of models to reduce their size and improve performance, and a small footprint that allows it to run on devices with limited memory and storage. To the best of our knowledge, no previous studies have thoroughly analyzed the performance and resource requirements of smaller BERT variants such as MobileBERT [26] on resource-constrained devices. Our research aims to

provide insight into the potential capabilities and limitations of utilizing MobileBERT on Raspberry Pi devices for NLP tasks. The contributions of this paper are as follows:

- Provide a comparative performance and resource usage analysis of BERT Large and its lightweight variant, MobileBERT.
- Develop a novel framework for evaluating MobileBERT models in TensorFlow-Lite format on resource-constrained devices, both with and without quantization applied.
- Implement a low-cost and privacy-preserving system that processes data locally on edge devices using a TinyML model.
- Demonstrate the performance of MobileBERT in classifying social media texts based on their reputation polarity.

II. Related Works

TinyML has seen significant growth in recent years, with many research studies addressing the challenges and opportunities associated with deploying ML models on small, low-power devices. In this section, we will review the most relevant literature on TinyML, focusing on recent advancements and challenges in the field. Table I compares the related works qualitatively. There have been many studies with the aim of compression of large neural networks. Previous studies have highlighted the importance of model compression techniques for deploying ML models on resource-constrained devices. In [12], Han et al. present a technique for compressing deep neural networks by using three methods, pruning, trained quantization, and Huffman coding, to reduce the size and computational cost of the model while maintaining good accuracy. Iandola et al. in [15] present a new Convolutional Neural Network (CNN) architecture that achieves AlexNet-level accuracy while having 50× fewer parameters and a model size of less than 0.5MB. Additionally, in [16], Jacob et al. first showed that quantizing neural networks to perform inference using integer-only arithmetic without a significant loss in accuracy is possible. The paper proposes a training method for quantization that combines quantization-aware training and post-training quantization.

Moreover, Wang et al. present a method for automatically quantizing deep learning models for efficient deployment on hardware devices such as mobile phones, embedded systems, and IoT edge devices. Their proposed method, named Hardware-Aware Automated Quantization (HAQ), uses reinforcement learning (RL) and evolution algorithms to explore the quantization search space and find the best quantization method for a given hardware target [29]. Additionally, the AMC method proposed in [13] uses a hardware-aware evaluation function and a resource constraint such as FLOPs to control the search space and find the best trade-off between model size and performance. It also uses an RL algorithm to search for the best combination of model compression techniques and hyperparameters that maximize the trade-off between model performance and resource efficiency.

Furthermore, there have been some recent advancements of TinyML in the NLP area as well, where the smaller NLP models have been able to achieve comparable performance in contrast with full-precision models while using significantly fewer parameters. Houlsby et al. in [14] present an adapter-dependent method for transfer learning in NLP tasks that is efficient in terms of the number of parameters used, where only the higher layers are fine-tuned for the specific task at hand to achieve parameter efficiency, as they are useful in building task-specific features. In [31], Wu et al. present a novel transformer architecture called LITE Transformer, where the authors introduced a long-short-range attention mechanism, which selectively attends to different ranges of positions in the input sequence based on their relevance to the task. This reduces the number of attention calculations required, resulting in a more efficient model. In the work presented in [32], Yan et al. proposed a transformer-based architecture called Micronet, a parameter and computation-efficient language model. The architecture is based on a combination of techniques such as adaptive embedding, knowledge distillation, network pruning, low-bit quantization, and differentiable non-parametric cache. The approach performs similarly to other full-precision models in various NLP tasks with fewer parameters.

In [28], the authors propose an efficient and adaptive transformer architecture that takes into account the characteristics of hardware, such as memory bandwidth, computation power, and energy consumption. The proposed model uses a combination of techniques such as knowledge distillation, quantization, and model pruning to reduce computation and memory requirements. Both TinyBERT [17] and MobileBERT [26] are compact and efficient versions of BERT that can run on resource-constrained devices, and both can be used in a wide range of applications, such as offline natural language understanding on mobile devices, voice assistants, and language-based IoT applications. Moreover, Niu et al. [21] deployed their own compiler-aware neural architecture optimization models in addition to other BERT variants, including MobileBERT in TensorFlow-Lite format, and compared the performances in Question-answering and Text Generation tasks. Their proposed framework, as well as the other architectures, were evaluated using the Samsung Galaxy S10 cell phone, which has a Qualcomm Snapdragon 865 processor.

Many works have analyzed the performance and resource requirements of TensorFlow-Lite MobileBERT models on resource-constrained devices. Despite their valuable efforts, these works fail to comprehensively analyze their implementations' resource utilization. Herein, we provide a com-

Table I: Qualitative Comparison of the Related Works

Proposed System	Experiment Resources	Transformer-based Large Architectures	CPU Utilization Details Presented	Memory Utilization Details Presented	Power Dissipation Details Presented
Deep Compression [12]	Intel Core i7 5930K NVIDIA GeForce GTX Titan X, NVIDIA Tegra	✗	✓	✗	✓
AMC [13]	Qualcomm Snapdragon 821, NVIDIA Titan XP GPU	✗	✓	✗	✗
Houlsby et al. [14]	Google Cloud TPU	✓	✗	✗	✗
Lite Transformers [31]	ARM Cortex-A72 mobile CPU	✓	✗	✗	✗
HAT [28]	Intel Xeon, NVIDIA Titan, ARM Cortex A72	✓	✗	✗	✗
Niu et al. [21]	Qualcomm Snapdragon 865	✓	✓	✗	✗
This Work	Intel Core i5-7500 Broadcom BCM2837 1.2 Ghz Broadcom BCM2837 1.4 Ghz Broadcom BCM2711 1.5 Ghz	✓	✓	✓	✓

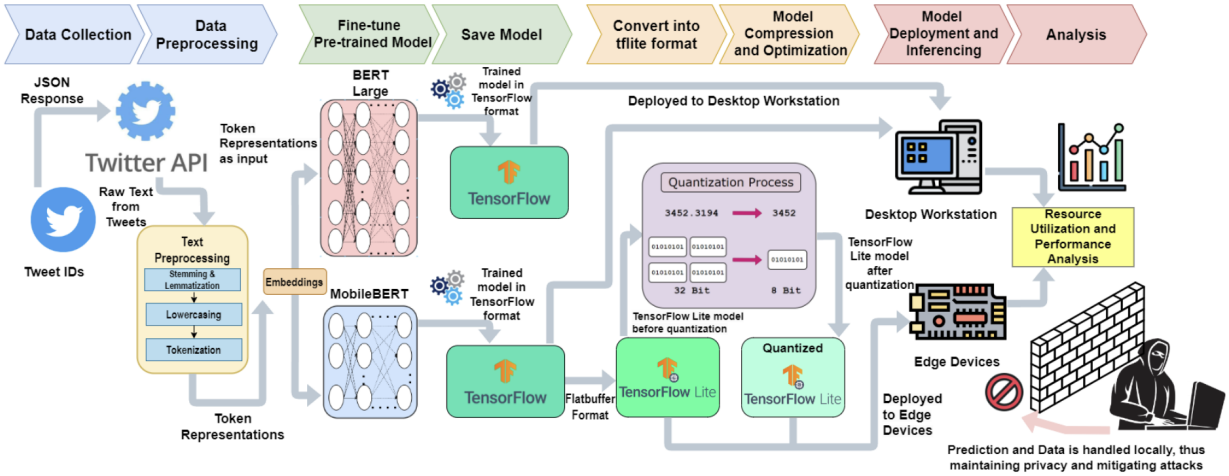

Figure 1: Proposed Framework Architecture

prehensive assessment of the capabilities and limitations of MobileBERT on Raspberry Pi devices for NLP tasks, thereby filling the existing gap in the literature. In our study, for the specific NLP task under consideration, we fine-tuned both the BERT Large and MobileBERT models to perform multiclass classification based on the polarity of reputation, similar to the approach presented in [24]. Furthermore, we demonstrate the trade-off between performance and model size, as well as latency reduction, when utilizing quantized TensorFlow-Lite models. Our findings indicate that significant improvements in terms of model size and latency are achieved while incurring a negligible decrease in performance.

III. Framework Architecture

In this section, we present the architecture of our framework. The overall architecture of our proposed framework is demonstrated in Figure 1, which illustrates the various stages of the framework. The first stage of the framework involves data collection and preprocessing, which serves as an essential step for fine-tuning the BERT large and MobileBERT models. Subsequently, the fine-tuning phase is carried out using the preprocessed data. After the training process is completed, the MobileBERT model is converted to the TensorFlow-Lite model format, with subsequent optimization and compression achieved through the application of quantization. The analysis module then evaluates the performance and resource utilization of the models deployed on the target machines. Throughout the system, all processing occurs on the local edge device and does not require communication with a central server, reducing the risk of data interception and ensuring data privacy.

A. Data Collection and Preprocessing

Herein, our primary objective was to evaluate the performance of TensorFlow Lite models compared to TensorFlow models on edge devices. To accomplish this, first, we conducted fine-tuning of the BERT Large and MobileBERT models on a supervised dataset [10] that was specifically curated for reputation research. The multiclass classification task based on reputation polarity served as a means to assess and compare the performance of the TensorFlow Lite models within the context of our experiments. To accomplish this, we have utilized the RepLab 2013 [10] dataset, which comprises tweets about 61 entities from 4 different domains. For the fine-tuning process, we have focused exclusively on the English tweets within the dataset, as MobileBERT is not optimized for multilingual tasks.

In this study, we used the Twitter API [8] to collect tweets for our analysis. The Twitter API is a powerful tool that allows developers to access a wide range of data from the Twitter platform, such as tweets and their associated metadata. The Twitter API returns JSON objects containing the tweets that match our search criteria. Then we proceed to pre-process the extracted texts. Text pre-processing is a vital stage in reputation polarity tasks as it converts social media text into a more consumable format that is more suitable for ML models. Through this process, tweets from the RepLab 2013 dataset are cleaned and prepared for model training. In particular, we remove redundant spaces, symbols, emojis, URL links, and punctuation marks to ensure the data is in a compatible format for the ML model. The texts afterward are tokenized using the appropriate BERT tokenizer for the BERT models.

B. Fine-tuning of the Pre-trained Models

BERT [11] is a significant innovation in contextualized representation learning for NLP. In their work [11], authors demonstrate that even though the word embedding layer in traditional deep learning models for NLP tasks is trained on large language corpora, training a range of neural network architectures that encode contextual representations solely based on the limited supervised data for end NLP tasks is still inadequate. BERT employs a fine-tuning process that requires minimal architecture modifications for each end NLP task. BERT offers two parameter-intensive configurations, BERT Base and BERT Large. BERT Base comprises 768 hidden dimensions, 12 transformer blocks, and 12 attention heads, with a total of 110 million parameters. BERT Large, on the other hand, has 1024 hidden dimensions, 24 transformer blocks, and 16 attention heads, totaling 340 million parameters. The pre-training process with BERT models involves two key methods: masked language modeling and next-sentence prediction. In order to provide a comprehensive assessment of the capabilities of MobileBERT, we have selected BERT large as the baseline model for comparison in our study. This

decision is based on the fact that MobileBERT was derived from the inverted bottleneck version of BERT Large (IB-BERT) [26] through the process of knowledge distillation. In essence, MobileBERT represents a lightweight version of IB-BERT, specifically optimized for use on resource-constrained edge devices. This comparison allows us to evaluate the trade-offs between the performance and resource demands of MobileBERT and BERT large and to provide insight into the potential of MobileBERT for NLP tasks on edge devices.

Both BERT large and MobileBERT models use token representation vectors as input during the fine-tuning process. Each token is represented by a sum of three representation vectors: a positional embedding vector, which encodes information about the token's location in the sequence; a sentence vector is used when a single sentence is not sufficient to convey the context.; and a typical word embedding vector, which is a vector representation of the word in context. Additionally, BERT extends the input sentence by incorporating the [SEP] token and the [CLS] token. The [CLS] token carries the embedding for specific classification tasks, whereas the [SEP] token is responsible for separating segments. For our reputation polarity task, BERT utilizes the last hidden state h derived from the initial token [CLS] to encapsulate the entire input sequence. To predict the probability of reputation polarity class c, we augment BERT with a softmax classifier positioned atop, employing

$$p(c|h) = softmax(Vh), \tag{1}$$

where the parameter matrix V corresponds to the reputation polarity prediction task. Through fine-tuning the reputation polarity training data, we simultaneously optimize all parameters of BERT and the parameter matrix V. After completing the training process, the BERT Large TensorFlow model is deployed on a desktop workstation for further evaluation. Meanwhile, the MobileBERT model undergoes a conversion process to TensorFlow-Lite format, and compression techniques are applied to optimize it for deployment on resource-constrained edge devices.

C. Model Compression and Optimization

The trained TensorFlow MobileBERT models need to be converted to TensorFlow-Lite format. To achieve this, the first step is to export the TensorFlow model to a file format that TensorFlow-Lite can read, such as a TensorFlow SavedModel or a frozen TensorFlow GraphDef. This can be done using TensorFlow's built-in export functions. Freezing the model involves converting the variables in the model to constants so that the model's weights are embedded in the model graph. The TensorFlow-Lite Converter is then used to convert the frozen model to a TensorFlow-Lite FlatBuffer file. The converter takes the frozen model as input and generates a TensorFlow-Lite model. The converted TensorFlow-Lite models go through further quantization. We have used the Dynamic Range Quantization technique [4] as our quantization process. In DRQ, the range of the weights and activations are adjusted based on the data range and are converted from float points to 8-bit integers. This allows for more efficient use of the available bits while offering a smaller model size and lower computational requirements without a significant loss of performance. In contrast, in the traditional quantization method, the range of the weights and activations of the models are fixed, which can lead to information loss and a significant drop in performance. Both the quantized and non-quantized models are then deployed in target machines, and their performance and resource utilization data are analyzed.

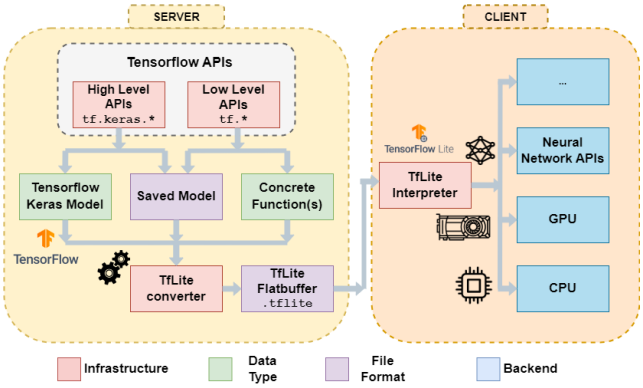

Figure 2: TensorFlow-Lite Conversion and Deployment.

Table II: Hardware Specifications of Devices

Machine	CPU	RAM	Memory
Embedded Raspberry Pi 3B	Broadcom BCM2837 SoC @1.2GHz	1GB	32GB μSD
Embedded Raspberry Pi 3B+	Broadcom BCM2837 SoC @1.4GHz	1GB	32GB μSD
Embedded Raspberry Pi 4B	Broadcom BCM2711 SoC @1.5GHz	4GB	32GB μSD

D. Model Deployment

To deploy the TensorFlow-Lite models on resource-constrained devices, we have utilized the TensorFlow-Lite (TFLite) interpreter [3]. TFLite interpreter is a library that enables developers to run TensorFlow-Lite models on edge devices with limited computational resources. It takes a TensorFlow-Lite model as input and performs the computations defined in the model's graph by loading the model into memory and converting it into a format that can be executed on the device's hardware. It provides an API that enables developers to interact with the model, such as inputting data, running the computations, and retrieving the output. Thus, the TFLite interpreter bridges the gap between the TensorFlow-Lite model and the device's hardware. Figure 2 illustrates the process of converting TensorFlow models to TensorFlow-Lite format and the deployment of the converted models on target machines. In this study, the TensorFlow models were deployed on a desktop workstation, and the inference was performed solely using the CPU on the test dataset. Additionally, both quantized and non-quantized versions of the TensorFlow models were deployed on Linux-based embedded devices. Data pertaining to performance metrics and resource utilization obtained from the inference operation on these devices were collected and utilized for comparative analysis.

Table III: Experimental Results

Model	Hardware	Avg. Accuracy	Avg. F-Score	Max Accuracy	Max F-Score
TensorFlow-Lite MobileBERT (32 bit)	Embedded Device	0.685±0.0031	0.602±0.004	0.70	0.61
TensorFlow-Lite MobileBERT (16 bit)	Embedded Device	0.683±0.0179	0.601±0.002	0.69	0.61
TensorFlow-Lite MobileBERT (8 bit)	Embedded Device	0.684±0.0027	0.603±0.004	0.69	0.61

E. Model Evaluation and Analysis

To conduct a comprehensive evaluation of the deployed models, we introduce metrics that considered model latency, performance, and efficiency. These metrics, namely the Speed Index (SI), Model Performance Index (MPI), and Resource Efficiency Ratio (RER), were designed to provide a holistic assessment of the models' effectiveness.

1) Speed Index (SI): The Speed Index (SI) metric captures the trade-off between the speed of the model, represented by FLOPS (Floating-Point Operations Per Second), quantization bits, and total time in seconds. SI metric is computed using the following equation:

$$SI = \frac{FLOPS}{Q \times t}, \tag{2}$$

where Q is the Quantization Bits, and t is the total time in *seconds*. This metric quantifies how fast the model performs in relation to the number of operations, quantization, and time.

2) Model Performance Index (MPI): The Model Performance Index (MPI) metric evaluates the overall performance of the model by considering the average accuracy ($Accuracy_{avg}$), average F-Score ($(F-Score)_{avg}$), and total power dissipation ($Power_{tot}$) in *Kilo Watts*. The MPI is computed using the following equation:

$$MPI = \frac{Accuracy_{avg} + (F-Score)_{avg}}{Power_{tot}}, \tag{3}$$

which quantifies the model's performance in terms of accuracy and energy efficiency.

3) Resource Efficiency Ratio (RER): The Resource Efficiency Ratio metric measures the efficiency of resource utilization by considering CPU utilization ($CPU\%$), memory utilization ($MEM\%$), and energy consumption ($Energy_{tot}$). The Resource Efficiency Ratio is computed as follows:

$$RER = \frac{Energy_{tot}}{CPU\% \times MEM\%}, \tag{4}$$

which quantifies how efficiently the model utilizes resources.

IV. Experiment Results

This Section describes the experimental settings and presents a comprehensive discussion of the results.

A. Experiment Settings

In this study, the performance of the models is evaluated using the Accuracy and F-score metrics, consistent with the evaluation methods employed in RepLab 2013 [10]. In both models, the learning rate employed was 1×10^{-5} and the batch size utilized was 32. To provide the reproducibility of results, the experiments were performed with five different random seeds on all devices during each iteration of the experiments. Additionally, resource utilization data were collected and monitored in three stages.

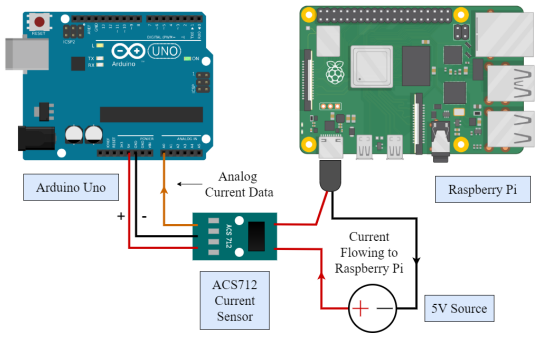

Figure 3: Current Sensing Hardware Setup Diagram.

1) CPU and Memory Utilization Data using PSUTIL: Prior to the initiation of the experiment, system-wide CPU and memory utilization percentage data were gathered. During the experiment, simultaneous collection of both system-wide and process-specific CPU and memory utilization data was conducted. After the experiment's conclusion and the process's termination, system-wide CPU and memory utilization data were gathered once more. In order to maintain the integrity and fairness of the experimental results, all experiments were initiated simultaneously. However, due to variations in latency among the different models, each model concluded its execution at different points in time. For these experiments, the PSUTIL package was utilized [5]. PSUTIL is a Python cross-platform library that provides an interface to retrieve information on system utilization, resources, and processes.The size of the BERT large TensorFlow, MobileBERT TensorFlow, non-quantized TensorFlow-Lite MobileBERT, quantized 16 and 8-bit TensorFlow-Lite MobileBERT models are 4GB, 299MB, 98MB, 49MB, and 25MB respectively. For the purpose of experimentation on Linux-based embedded devices, Raspberry Pi 3B, 3B+, and 4B devices [6] were utilized. The specifications of the hardware used in this work are presented in Table II. The BERT Large and MobileBERT TensorFlow models were deployed on a desktop workstation, while the 8-bit quantized, 16-bit quantized, and non-quantized (32-bit) versions of the MobileBERT TensorFlow-Lite models were deployed on Raspberry Pi devices. This resulted in a total of eleven models for comparative analysis. To facilitate the reproducibility of the results, the same random seeds were utilized for experiments on each device.

2) Current Sensing using ACS712 Sensor: The power dissipation of Raspberry Pi modules can be accurately estimated using an external current sensor, such as the ACS712 [1]. This sensor operates based on the Hall effect principle to measure the current flowing through a circuit by detecting the generated Hall voltage. Fig. 3 illustrates the schematic diagram of the circuitry used for current measurement. By connecting the ACS712 current sensor in series with the load, the sensor can measure the analog hall voltage magnitude corresponding to the instantaneous current. To convert this analog magnitude into a digital format, an Arduino Uno board [2] with a 10-

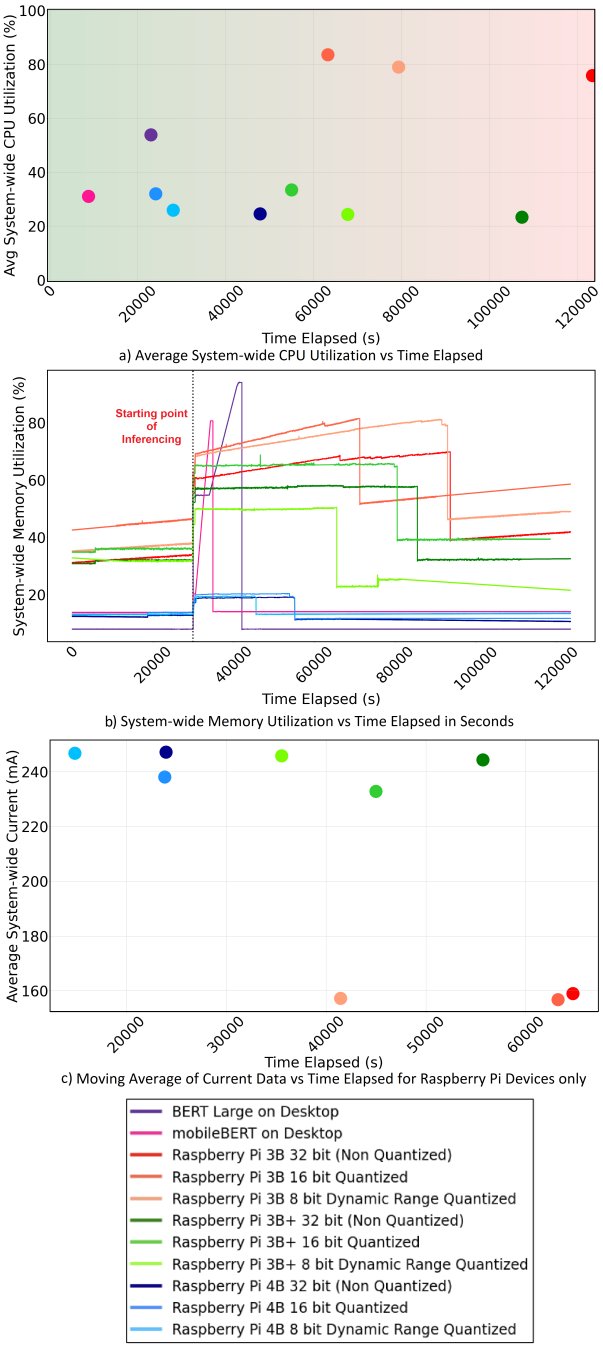

Figure 4: CPU, memory, and power utilization of the deployed models.

bit analog-to-digital converter (ADC) is used. The formula for calculating the instantaneous current $I(t)$ is:

$$I(t) = \frac{V_h \times V_{ref}}{ADC_{resolution}}, \tag{5}$$

where V_h represents the instantaneous current sensor reading (Hall Voltage), V_{ref} is the reference voltage used by Arduino Uno (5V), and $ADC_{resolution}$ refers to the resolution of the ADC, which is 10 bits (resulting in 1024 possible values).

B. Results from Comparative Analysis

The quantitative results of our experiments are provided in Table III. The results show that the conversion of TensorFlow-Lite models resulted in a relatively small (4.1%) drop in

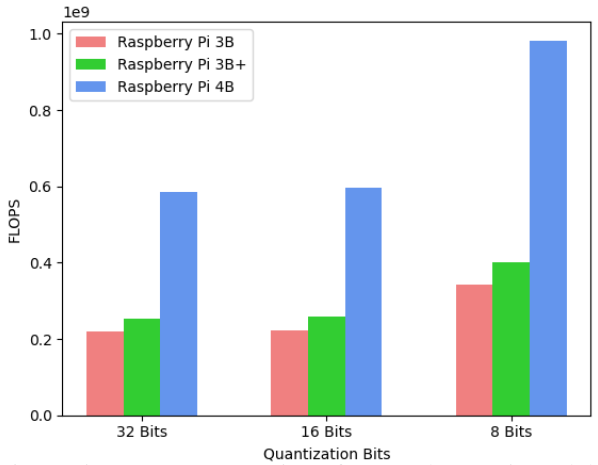

Figure 5: FLOPS Comparison for Raspberry Pi Models.

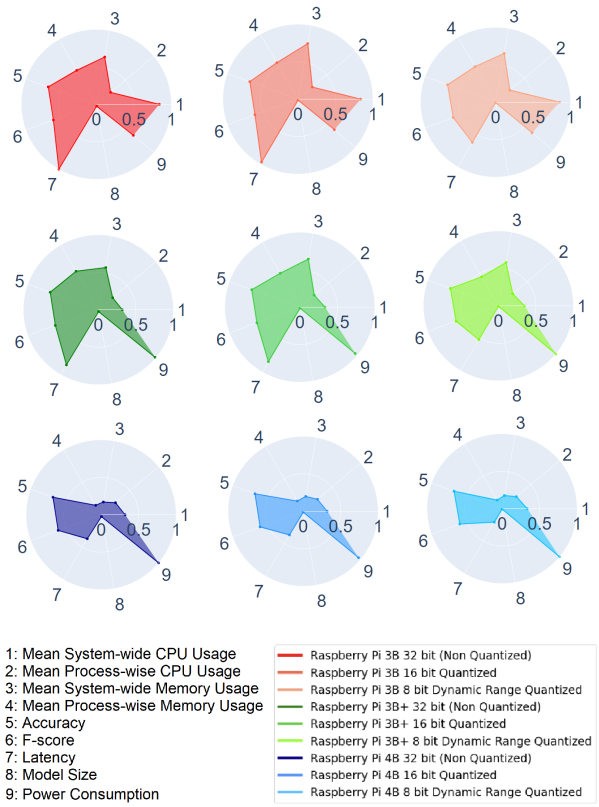

Figure 6: Comparative Analysis of the deployed models.

performance. The main distinction between the models can be observed in their resource utilization. The TensorFlow-Lite models, particularly the 8-bit quantized versions, exhibit significantly lower resource usage, as shown in Figure 4(a). In the context of Raspberry Pi devices, it has been observed that TensorFlow Lite models exhibit an average CPU utilization of approximately 25% across various Raspberry Pi versions. This utilization corresponds to the utilization of a single core out of the available four cores on these devices. The BERT Large model running on the Desktop workstation has an average process-wise CPU utilization of 71.8%. Figure 4(a) indicates that the TensorFlow-Lite models deployed on the Raspberry Pi 3B showed higher system-wide CPU usage with high latency as well. This observation can be attributed to the delay added for accessing data from memory sources due to paging. Addi-

Table IV: Model Evaluation Results.

Device	Quantization Bits	Power (KW)	Time (s)	Power per Inference (W/sample)	Time per Inference (s/sample)	FLOPS	Avg Accuracy	Avg F-Score	Avg CPU Util. (%)	Avg Memory Util. (%)
RP3B	32	2919.80	64719	74.9	1.66	$2.18 \cdot 10^8$	0.685	0.602	83.9	64.1
RP3B	16	2845.32	63219	72.99	1.62	$2.23 \cdot 10^8$	0.683	0.601	83.5	75.3
RP3B	8	1851.31	41449	47.49	1.06	$3.42 \cdot 10^8$	0.684	0.603	85.2	66.1
RP3B+	32	3848.44	55693	98.72	1.42	$2.54 \cdot 10^8$	0.685	0.602	31.9	57.1
RP3B+	16	3651.06	44999	93.66	1.15	$2.57 \cdot 10^8$	0.683	0.601	33.4	65.2
RP3B+	8	2471.69	35549	63.4	0.91	$4.00 \cdot 10^8$	0.684	0.603	34.7	58.7
RP4B	32	1675.51	23969	42.98	0.61	$5.84 \cdot 10^8$	0.685	0.602	31.9	17.2
RP4B	16	1653.73	23849	42.42	0.6	$5.97 \cdot 10^8$	0.683	0.601	31.9	20.1
RP4B	8	1052.62	14851	27	0.38	$9.82 \cdot 10^8$	0.684	0.603	33.4	17.8

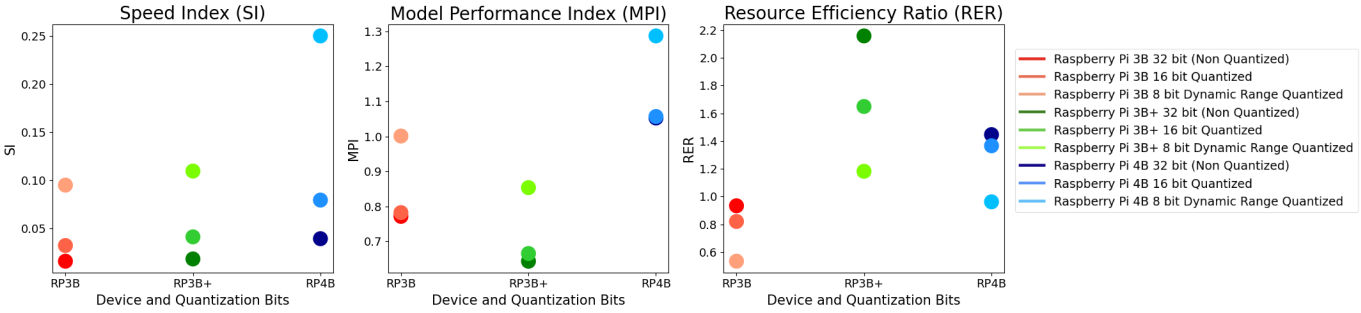

Figure 7: The SI, MPI, and RER values of the deployed models

tionally, Figure 4(b) demonstrates that the memory utilization of both BERT Large and MobileBERT models is significantly higher in comparison to the TensorFlow-Lite models.

The evaluation results of the nine models deployed on edge devices are presented in Figure 6, addressing the capabilities and limitations of each model by presenting normalized values between 0 and 1 for a set of attributes. The power dissipation data were collected for the nine TensorFlow-lite models only. According to the figure, the latency of the TensorFlow-Lite models is the only area of concern. Specifically, the non-quantized 32-bit versions of TensorFlow-Lite models on Raspberry Pi 3B and 3B+ are slower than the BERT Large model by a factor of 5.55× and 4.76×, respectively. However, all of the 8-bit quantized TensorFlow-Lite models on the three devices manage to produce at least one prediction per second. The fastest quantized TensorFlow-Lite model, deployed on Raspberry Pi 4B, is only 1.15× slower than the BERT Large model while offering 160× smaller footprint. This demonstrates that it is feasible to deploy large NLP models like BERT variants on edge devices and achieve comparable performance and latency efficiently using the TensorFlow-Lite models.

The presented scatter plot in 4(c) and spider graphs in 6 reveals notable power dissipation disparities between the Raspberry Pi 3B and the Raspberry Pi 3B+ as well as Raspberry Pi 4B. Specifically, it is evident that the latter two models exhibit elevated power dissipation levels in comparison to the Raspberry Pi 3B. This discrepancy may be attributed to the increased floating-point operations per second (FLOPS) and random-access memory (RAM) capacities inherent in the Raspberry Pi 4B and 3B+ models, which contribute to relatively higher power requirements during operation. Table IV and Figure 7 present evaluation results for different devices and quantization bits, including Raspberry Pi 3B, 3B+, and 4B. power dissipation and inference time were measured, with Raspberry Pi 3B devices generally exhibiting lower power requirements but longer inference times compared to Raspberry Pi 3B+ and Raspberry Pi 4B. FLOPS as presented in Figure 5 , a measure of computational performance, was highest for Raspberry Pi 4B across all quantization options. The devices' average accuracy and F-score were comparable, with slight variations based on quantization bits. Resource utilization showed variations, with RP3B+ demonstrating lower average CPU utilization and Raspberry Pi 4B having the lowest average memory utilization.

A higher SI value indicates faster processing speed and higher computational efficiency, as the model is able to perform a larger number of operations (FLOPS) relative to the number of quantization bits and time. Conversely, a lower SI value suggests slower performance and potentially less efficient resource utilization. A higher MPI value indicates better overall performance, reflecting a balance between accuracy and energy consumption. By considering the average accuracy, average F-Score, and power dissipation, the MPI provides a comprehensive assessment of the model's performance. A higher RER value, on the other hand, indicates more efficient utilization of resources, reflecting a better balance between resource consumption and performance. The analysis of the results reveals that the Raspberry Pi 4B equipped with 8-bit quantization exhibited superior values for both SI and MPI metrics. This outcome suggests that the device achieved higher levels of speed and performance compared to other configurations. Conversely, the Raspberry Pi 3B Plus models displayed remarkably high values for the RER metric across the tested configurations, indicating commendable resource efficiency. These findings hold significant implications for guiding the selection of appropriate devices and quantization configurations based on the desired trade-offs including speed, performance, and resource utilization. Considering these metrics, researchers and developers can make informed decisions to strike a balance between the aforementioned factors and meet their specific requirements.

V. Conclusion

With TinyML, intelligent decisions can be made on edge devices such as smart home appliances, sensors, and wearables. In this paper, we have explored the application of

TinyML in NLP through the fine-tuning of BERT models. Our experiments have used TensorFlow-Lite models to identify the reputation polarity from a given text, demonstrating the potential of these models in enabling automation and intelligence on edge devices. Previous works in TinyML have demonstrated the application of BERT variants on Android devices, performing various local NLP tasks without the need for a server. However, more research is needed regarding the deployment of large NLP models on devices with even fewer resources, such as Raspberry Pi. Our paper contributes to this field by providing a thorough evaluation of the capabilities and limitations of MobileBERT TensorFlow-Lite models deployed on Raspberry Pi devices. The results of our experiments demonstrate that these converted and quantized TensorFlow-Lite models can achieve performance comparable to that of the BERT Large model, with significantly lower resource utilization and a smaller code footprint. Our findings provide valuable insight into the deployment of large NLP models on embedded systems using the concepts of TinyML. Our future work will address the integration of TinyML in the context of federated learning, which presents a promising opportunity by enabling the training of ML models on resource-constrained edge devices while preserving privacy.

ACKNOWLEDGMENT

This work is supported by National Science Foundation (NSF) projects 1624668, 1921485, 2213634, and 2335046 the Department of Energy- National Nuclear Security Administration under Award Number DE-NA0003946, the AGILITY project 4263090, sponsored by Korea Institute for Advancement of Technology (KIAT South Korea), and the University of Arizona's Research, Innovation & Impact (RII) award for the 'Future Factory'.

REFERENCES

[1] Acs712: Fully integrated, hall-effect-based linear current sensor ic with 2.1 kvrms voltage isolation and a low-resistance current conductor, available at: https://www.allegromicro.com/en/products/sense/current-sensor-ics/zero-to-fifty-amp-integrated-conductor-sensor-ics/acs712.

[2] Arduino uno rev3, available at: https://store.arduino.cc/products/arduino-uno-rev3.

[3] Interpreter interface for running tensorflow lite models., retrieved: January 2023, available at: https://www.tensorflow.org/api_docs/python/tf/lite/interpreter.

[4] Post-training quantization, retrieved: January 2023, available at: https://www.tensorflow.org/lite/performance/.

[5] psutil documentation, retrieved: January 2023, available at: https://psutil.readthedocs.io/en/latest/.

[6] Raspberry pi products, retrieved: January 2023, available at: https://www.raspberrypi.com/products/.

[7] Tensorflow-lite, retrieved: January 2023, https://www.tensorflow.org/lite/guide.

[8] Twitter api, retrieved: January 2023, available at: https://developer.twitter.com/en/docs/twitter-api.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *Osdi*, volume 16, pages 265–283. Savannah, GA, USA, 2016.

[10] E. Amigó, J. Carrillo de Albornoz, I. Chugur, A. Corujo, J. Gonzalo, T. Martín, E. Meij, M. De Rijke, and D. Spina. Overview of replab 2013: Evaluating online reputation monitoring systems. In *Information Access Evaluation. Multilinguality, Multimodality, and Visualization: 4th International Conference of the CLEF Initiative, CLEF 2013, Valencia, Spain, September 23-26, 2013. Proceedings 4*, pages 333–352. Springer, 2013.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[13] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018.

[14] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[15] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[17] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[19] E. Marin, D. Perino, and R. Di Pietro. Serverless computing: a security perspective. *Journal of Cloud Computing*, 11(1):1–12, 2022.

[20] W. Niu, Z. Kong, G. Yuan, W. Jiang, J. Guan, C. Ding, P. Zhao, S. Liu, B. Ren, and Y. Wang. Real-time execution of large-scale language models on mobile. *arXiv preprint arXiv:2009.06823*, 2020.

[21] W. Niu, Z. Kong, G. Yuan, W. Jiang, J. Guan, C. Ding, P. Zhao, S. Liu, B. Ren, and Y. Wang. A compression-compilation framework for on-mobile real-time bert applications. *arXiv preprint arXiv:2106.00526*, 2021.

[22] OpenAI. Gpt-4 technical report, 2023.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[24] M. W. U. Rahman, S. Shao, P. Satam, S. Hariri, C. Padilla, Z. Taylor, and C. Nevarez. A bert-based deep learning approach for reputation analysis in social media. In *2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–8. IEEE, 2022.

[25] S. Soro. Tinyml for ubiquitous edge ai. 2021.

[26] 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.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[28] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

[29] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[30] P. Warden and D. Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[31] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han. Lite transformer with long-short range attention. *arXiv preprint arXiv:2004.11886*, 2020.

[32] Z. Yan, H. Wang, D. Guo, and S. Han. Micronet for efficient language modeling. In *NeurIPS 2019 Competition and Demonstration Track*, pages 215–231. PMLR, 2020.

[33] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.