Optimizing Keyword Spotting Classifier based on Tiny Machine Learning for Low-Power Embedded Devices

Patrik Medur*, Mart Lubbers†, Goran Mauša*‡
* University of Rijeka/Faculty of Engineering, Rijeka, Croatia
† Radboud University/Institute for Computing and Information Sciences, Nijmegen, The Netherlands
‡ University of Rijeka/Center for Artificial Intelligence and Cybersecurity, Rijeka, Croatia
corresponding_author: goran.mausa@uniri.hr

***Abstract*—Keyword Spotting (KWS) is essential for enabling voice recognition in smart home systems and although deploying reliable and energy-efficient models on low-power embedded devices remains a challenge, tiny machine learning offers a promising solution. In this paper, raw audio data from Google's Speech Commands Dataset v0.02 is used to develop and evaluate neural network architectures optimized for KWS based on Mel-frequency cepstral coefficients (MFCC). The audio signals are formatted as a compact 2D matrix of time-frequency features, making it ideal for convolutional neural networks (CNNs). Ten networks are trained with 10-fold cross-validation, fine-tuned on speaker-specific data and, by using post-training quantization, converted from TensorFlow (TF) to TensorFlow Lite (LiteRT) format for deployment on microcontrollers. The trade-offs between model size, performance and energy efficiency are analyzed, and the results show that LiteRT models considerably reduce the energy consumption, with energy savings ranging from 1–4% for smaller capacity models to 99.73% for larger ones, while maintaining performance similar to TF models. Furthermore, fine-tuning models of greater capacity slightly improves their accuracy (0.51%) and energy efficiency (1.75%), whereas smaller models are unaffected by fine-tuning. This study lays the foundation for the integration of KWS models based on tiny machine learning into microcontrollers for real-time applications.**

***Keywords*—Keyword spotting, Tiny machine learning, TensorFlow lite, Mel-frequency cepstral coefficients**

I. Introduction

Voice recognition has become an essential component of modern smart home systems, enabling users to control devices, access information, and automate tasks using simple voice commands. However, implementing reliable and energy-efficient voice recognition on low-power embedded systems remains challenging. Traditional voice recognition systems often rely on computationally intensive models that are not feasible for resource-constrained environments, especially in battery-powered devices [1].

Voice recognition systems are commonly split into two phases to optimize energy efficiency and use of computational resources. The first stage involves a lightweight Keyword Spotting (KWS) model deployed directly on the edge device, such as a microcontroller [2]. This model, which remains continuously operational, is tasked with identifying specific keywords while keeping power usage at a low level. Once a keyword is identified, the process transitions to the second phase, employing a more advanced and computationally demanding speech recognition system to interpret the command, utilizing either a cloud server or a more potent local device. This two-tiered approach balances low power usage with the ability to handle complex tasks, ensuring both efficiency and responsiveness.

The primary objective of this paper is to develop and optimize neural network-based audio keyword detection models that run efficiently on embedded systems with minimal energy usage. By carefully designing and evaluating different neural network architectures, the aim is to strike a balance between accuracy, model size, and energy efficiency. Given the constraints of embedded platforms, achieving reliable keyword recognition performance while minimizing power consumption is a key challenge.

Several studies explored the challenges and solutions for implementing KWS systems on low-power embedded devices. A study by Cioflan et al. [3], introduced on-device domain adaptation techniques, enabling KWS models to adapt to previously unseen environments. This approach demonstrates an improved accuracy, highlighting the potential for robust performance in real-world scenarios. Similarly, Bernal-Ruiz et al. [4], evaluated KWS prototypes in embedded systems, concluding that although these systems performed well with specific speakers, further optimization is necessary to enhance performance across a broader range of users. J. Wang and S. Li [5], emphasized the challenges of optimizing models for deployment on low-power edge microcontrollers. Their study underlined the critical role of techniques such as pruning and quantization in achieving good performance while maintaining the resource efficiency required for embedded environments.

This paper outlines the methodology for keyword Spotting (KWS), detailing dataset selection and preprocessing for model training. Next, we describe the neural network architectures designed for KWS, focusing on trade-offs between accuracy, model size, and efficiency. Finally, a case study evaluates model performance, emphasizing their suitability for deployment on low-power embedded devices.

II. Methodology

A. Keyword Spotting System

KWS plays a critical role in enabling seamless human-computer interaction. At its core, this system is designed to detect specific predefined words or phrases, often referred to as *trigger words*, within an audio stream. For example, phrases like "hey google" or "Alexa" activate virtual assistants, setting the stage for further interaction [6], [7].

KWS operates by analyzing incoming audio data, typically represented as a sequence of features derived from raw waveforms, such as Mel-frequency cepstral coefficients (MFCC) [8] or spectrograms. These visual representations of audio data allow deep learning models, particularly convolutional neural networks (CNN), to train more effectively and better learn patterns corresponding to specific keywords [9]. At the end of the KWS pipeline, there is a light-weight classifier which should be able to detect each keyword (Fig. 1).

In real-world scenarios, KWS systems must operate in real-time, often on low-power devices like embedded systems or mobile processors [10]. This imposes constraints on both computational resources and memory. Techniques such as model quantization, pruning, and lightweight architectures (e.g., MobileNet or TinyML models) [11] are commonly employed to optimize performance.

B. Classifier

To evaluate KWS architectures, we develop a series of 10 CNN models. Each model processes the same input size of 13×50 matrix `float32` data, representing the MFCCs of a $1\,\mathrm{s}$ audio segment. This input is first expanded to $13 \times 50 \times 1$ to match the dimensional requirements of CNNs. The models are structured with varying levels of complexity, with each subsequent model introducing additional layers or more sophisticated configurations. All models include fundamental components such as 2D convolution layers, max pooling, and a flattening layer to transition from feature extraction to the final classification stage. In some models, there is a dropout layer used for regularization. The number of convolutional layers, filter sizes, and pooling operations progressively increases across the models, allowing for more intricate feature extraction in the bigger architectures.

By incrementally increasing complexity, this approach provides insight into the trade-offs between model size, accuracy, and computational demands. While simpler models are more efficient and lightweight, the more complex models are designed to capture higher-level features, potentially leading to improved accuracy. This systematic scaling enables a comprehensive evaluation of how architectural depth and layer configuration impact performance in the context of KWS tasks.

In the following sections, we present our TensorFlow (TF) models alongside their lightweight counterparts, TensorFlow Lite (LiteRT) models, and conduct a comparative analysis of their performance metrics. LiteRT is a lightweight framework designed for mobile and embedded systems, enabling the efficient execution of machine learning models on constrained hardware. Furthermore, we examine the relationship between model size and accuracy, highlighting the trade-offs and advantages of quantization in creating compact yet efficient models.

C. Dataset

The dataset used for training and testing is Google's Speech Commands Dataset v0.02 [12]. Designed specifically for KWS tasks, this dataset provides a diverse and standardized collection of audio samples that represent common spoken commands. It serves as a benchmark for developing and evaluating machine learning models aimed at detecting specific keywords in real-time audio streams. This version of the dataset was released on 11 April 2018, and includes $105\,829$ audio files organized into 36 directories. Each of 35 directories represents a specific spoken command (e.g., "stop", "yes", "no", "up", "down"), with most speakers repeating each word up to five times. An additional directory labeled "background noise" contains non-verbal audio samples.

Each audio file is labeled based on its directory name and contains a waveform represented by $16\,000$ samples (corresponding to $1\,\mathrm{s}$ of audio at a $16\,\mathrm{kHz}$ sampling rate). The amplitude of the waveform is normalized to fall within the range of $[-1, 1]$. The metadata for each audio file includes the speaker ID and the utterance number. The waveform data for the spoken command "zero" is illustrated in Fig. 1, appearing as the first image in the pipeline.

D. Data preprocessing

The raw audio data are not directly used for model training but preprocessed to enhance its suitability for machine learning. First, the audio is down-sampled from $16\,\mathrm{kHz}$ to $8\,\mathrm{kHz}$, as this reduction maintains sufficient information for model training while improving computational efficiency. The audio is then transformed into MFCCs. The key parameters used in the extraction of MFCC included:

- Window size: $25\,\mathrm{ms}$
- Step size: $20\,\mathrm{ms}$
- Number of coefficients: 13
- Number of Mel filter banks: 20
- Fast Fourier Transform (FFT) size: 256
- Window function: Hanning

This preprocessing step results in a 2D representation of the audio signal, specifically a 13×50 matrix, which serves as an image-like input for machine learning models. MFCC data for the spoken command "zero" is illustrated in Fig. 1, appearing as the second image in the pipeline. Samples that do not match this size are discarded, and the percentage of discarded samples is $0.084\,\%$.

MFCC parameters are chosen to balance computational efficiency and feature quality for deployment in resource-

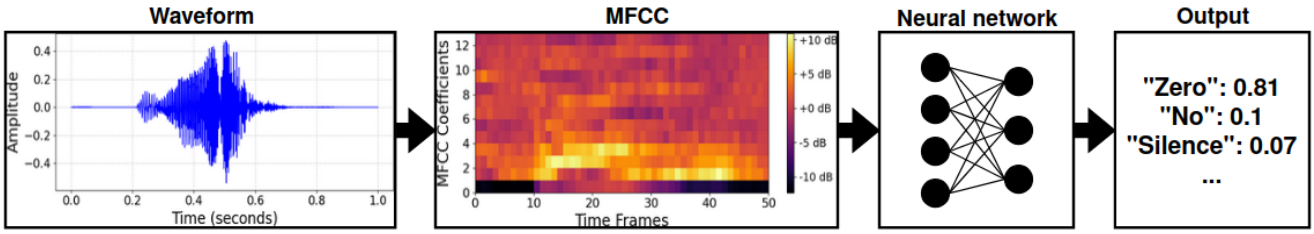


Fig. 1. KWS pipeline.

constrained devices [13]. The number of frames on MFCC is calculated as seen in equation (1).

$$x = \left\lceil \frac{T - (T_w[ms] * T)}{T_s[ms] * T} + 1 \right\rceil \tag{1}$$

Where:

- T is the total duration of the audio in samples.
- T_w is the window size.
- T_s is the step size.

In KWS, distinguishing between speech, silence, and background noise is essential for performance. The *0th* MFCC coefficient, representing audio signal energy, helps identify when keywords are spoken [14], while the remaining 12 coefficients capture phonetic details necessary for actual keyword classification. This balance of energy detection and phonetic discrimination is particularly important for low-power embedded devices where computational efficiency is crucial.

For the purpose of testing, the data is split into training, validation and testing subsets for 10-fold cross-validation. Each iteration uses $80\,\%$ of data (8 folds) for training, $10\,\%$ of data (1 fold) for validation, and the remaining $10\,\%$ of data (1 fold) for testing. To ensure consistency, all utterances from the same speaker for a given command are grouped together in the same fold. Additionally, the testing set is further subdivided: $30\,\%$ of the testing data is reserved for fine-tuning the models. For this fine-tuning subset, we specifically selected instances where users had recorded multiple utterances of the same word, ensuring that different articulations from the same speaker were used for fine-tuning and testing. The purpose of extracting a subset from the testing set is to mimic a real world scenario and estimate whether additional training on utterances from the same speaker (but different recordings) would have a positive or negative effect on model performance.

III. Results

Three main experiments are conducted to evaluate the performance, efficiency, and generalization capabilities of KWS:

1) Evaluating models for memory — performance trade-offs,
2) Fine-tuning on speaker-specific data,
3) Converting models to LiteRT with and without post-training quantization.

A. Preparing the Models

TF is an open-source machine learning framework, designed for building and training deep learning models. To explore the performance and efficiency of various CNN architectures, 10 TF models are developed and evaluated using 10-fold cross-validation. Each model maintains the same basic structure, with variations introduced in the number of 2D convolutional layers, each followed by activation functions and a max pooling layer. Larger-capacity models incorporate dropout layer during training to mitigate overfitting. The number of convolutional blocks and the presence of a dropout layer in each model are summarized in Table I, while the overall architecture used in every model can be seen in Fig. 2.

Each model is trained for 30 epochs with early stopping based on validation loss, ensuring strong generalization to unseen data while preventing overfitting. The models are evaluated based on accuracy, F1 score, and GPU energy consumption measured using NVIDIA's pynvml library, which provides real-time power usage data in watts (W). The F1 score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance, particularly useful when dealing with imbalanced datasets.

To investigate the impact of fine-tuning, the TF models are further trained and compared to the original versions to analyze differences in model size, performance, and energy efficiency.

During the conversion process, several techniques are applied to improve efficiency. Firstly, TF optimizes the trained model by pruning redundant operations and fusing compatible layers. Secondly, additional optimizations such as operator fusion, memory reuse, and quantization are performed when converting to LiteRT. These transformations reduce computational complexity and memory usage while maintaining model accuracy. The conversion to LiteRT involved creating two versions for each model. The first version retains the original floating-point (`float32`) precision, while the second utilizes full integer quantization. Quantization is a process that reduces the precision of a model's numerical representations, such as converting from 32 bit floating-point numbers to 8 bit integers. In this case, weights, activations, target inputs and outputs are all represented using 8 bit integers, significantly reducing the model's size and computational requirements. This enables efficient deployment on resource-constrained devices without substantial losses in accuracy.

TABLE I
MODEL ARCHITECTURE

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8	Model 9	Model 10
Conv_1	8	16	16	16	16	32	32	64	64	128
Conv_2	/	32	/	32	32	/	64	128	128	256
Conv_3	/	/	/	/	/	/	/	/	256	512
Dense_1	32	64	64	128	128	128	128	256	512	1024
Dropout	/	/	/	0.5	0.5	/	/	0.5	0.5	0.5
Dense_2	36	36	36	36	36	36	36	36	36	36

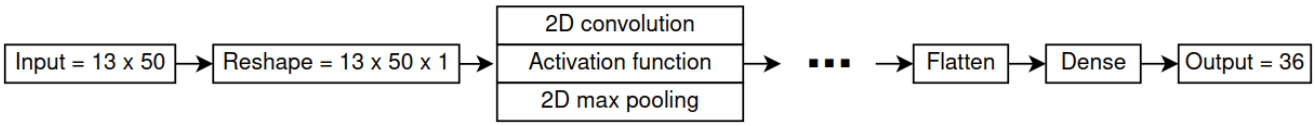


Fig. 2. KWS model architecture.

By creating both quantized and non-quantized LiteRT models, we analyzed the trade-offs between performance and memory size for microcontroller applications.

B. Fine-tuning the models

Building upon the models that were trained for memory-performance trade-off analysis, we explore the effects of fine-tuning on performance and energy efficiency. For this analysis, we utilize the 100 saved models (10 folds for each of the 10 CNN architectures) and fine-tune them using a subset of the testing data. This subset included speaker IDs with at least two utterances per command. Fine-tuning is performed over 30 epochs without early stopping, allowing the models to fully train on the new data. Fine-tuning generally has a positive impact on model performance and energy efficiency (Table II). Across all models, fine-tuning results in an average improvement of $0.14\,\%$ in accuracy, a $0.63\,\%$ increase in the F1 score, and a $1.80\,\%$ decrease in energy consumption. The memory size of fine-tuned models consistently decreases by the same margin of $1.01\,\mathrm{KB}$.

Table II summarizes the relative differences in accuracy, F1 score and energy consumption after fine-tuning the models, averaged across 10 folds. Positive values indicate improvements from fine-tuning, while negative values show the original models performed better. Statistical significance was assessed using the Wilcoxon signed-rank test, which evaluates consistent directional differences across all models. An asterisk (*) accompanying any value denotes statistically significant difference that exceeds random variation with high confidence ($p - value < 0.05$).

TABLE II
AVERAGE DIFFERENCE BETWEEN FINE-TUNED AND ORIGINAL MODEL ACCOMPANIED BY THE WILCOXON SIGNED-RANK TEST (ASTERIX DEMONTING IF A STATISTICALLY SIGNIFICANT DIFFERENCE EXISTS).

	Accuracy (%)	F1 score (%)	Energy consumption (%)
Model 1	−1.16*	1.74	12.96
Model 2	0.28	0.51	−11.17
Model 3	−1.23*	0.15	1.96
Model 4	0.63*	0.71*	−3.66
Model 5	0.67*	0.84*	−1.22
Model 6	−1.30*	−0.72	−3.36
Model 7	0.75*	0.63*	−0.63
Model 8	0.68*	0.73*	−4.76
Model 9	1.00*	0.83*	−2.60
Model 10	1.10*	0.92*	−5.53

C. Memory — performance trade-off

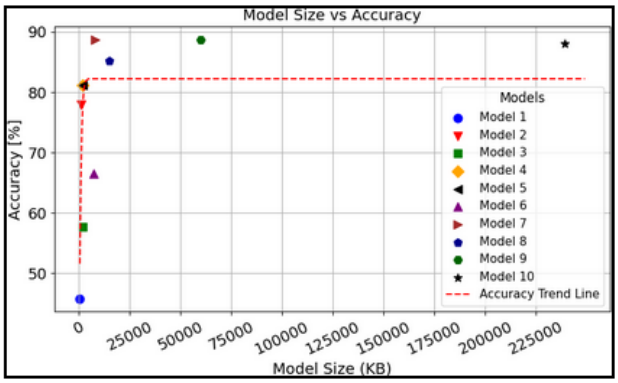
Our testing reveals a clear relationship between model size and accuracy in the context of our CNN architecture. As the model size increases, accuracy consistently improves up to a certain point. However, beyond a size of approximately $4\,\mathrm{MB}$, the accuracy plateaus, achieving just over $80\,\%$. This suggests a diminishing return on accuracy with increasing model size, as additional parameters beyond this threshold no longer particularly enhance performance. This trend is illustrated by the red trend line in Fig. 3. The first plot shows all 10 models with model size on the x-axis and accuracy on the y-axis, while the second focuses on models under $10\,\mathrm{MB}$ for better readability of smaller architectures; in both, the trend line illustrates how accuracy plateaus as model size increases, highlighting diminishing returns in performance gains for larger models.

The process of converting the models to LiteRT significantly reduces their size, but it also introduces a trade-off in terms of performance. As noted earlier, for each TF model, two LiteRT versions are created: one without quantization and one with full integer quantization. As shown in Table III, the results demonstrate that converting TF models to non-quantized LiteRT counterparts reduces their memory size by an average of $67.13\,\%$, while fully integer quantized models achieve an even greater reduction of $91.54\,\%$. This corresponds to a $74.26\,\%$ difference in memory size between the quantized and non-quantized LiteRT models, which closely aligns with theoretical expectations where full integer quantization typically reduces the model size by $75\,\%$.

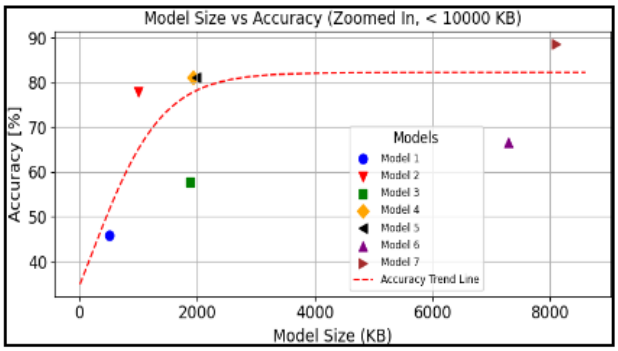
The conversion process also affects model performance metrics. For non-quantized LiteRT models, the accuracy remains unchanged compared to the original TF models,

TABLE III
MODEL SIZE COMPARISON ACROSS FORMATS (MB)

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8	Model 9	Model 10
TF Model [MB]	0.49	0.97	1.82	1.84	1.84	7.13	7.91	14.51	58.50	233.38
LiteRT non-quant [MB]	0.16	0.31	0.60	0.60	0.60	2.37	2.63	4.82	19.49	77.78
LiteRT quant [MB]	0.04	0.08	0.15	0.16	0.16	0.60	0.67	1.22	4.89	19.48



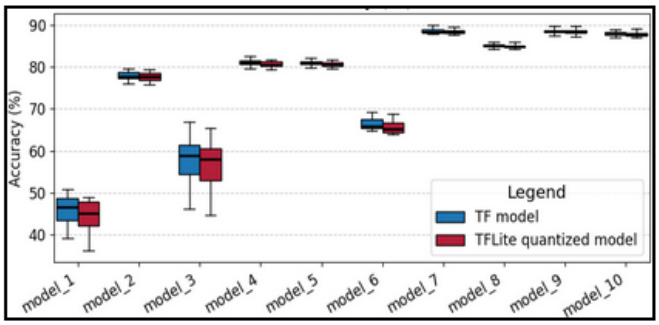
a) Average TF model accuracy in relation to models size



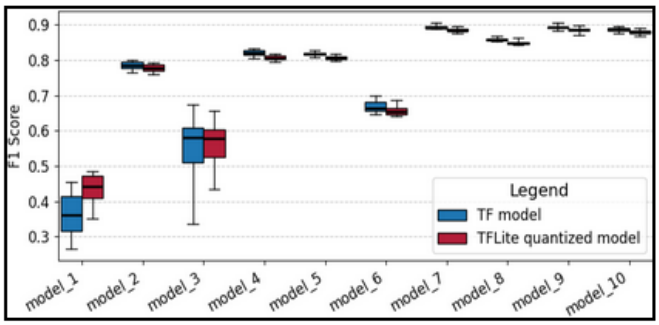
b) Average Accuracy of TF Models Under 10 MB in Relation to Model Size

Fig. 3. CNN models memory — accuracy trade-off.

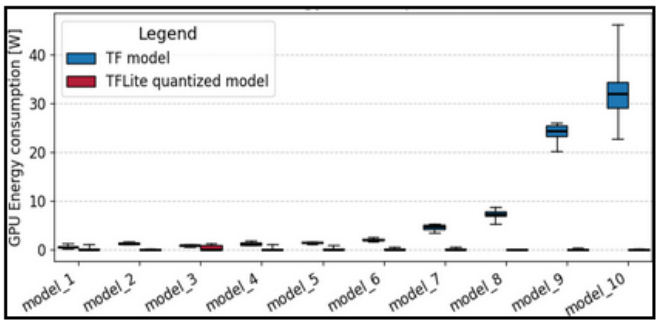
as both use the same numeric precision `float32`. Interestingly, the F1 score improved by an average of 2.10 %, and energy consumption was reduced by an impressive 82.38 %. On the other hand, fully quantized LiteRT models, which use `int8` precision, exhibited different behavior. Accuracy drops on average by 0.82 %, while the F1 score improves by 1.37 %. Energy consumption shows the most significant improvement, decreasing by 86.53 %. This substantial reduction occurs because integer operations require considerably less computation and power than floating-point operations. Additionally, quantization reduces both the model's memory size and the amount of data that needs to be transferred between memory and the processor. As a result, the execution of fully quantized models on embedded hardware is substantially more efficient. Notably, fully quantized LiteRT models consumed 25.86 % less energy than their non-quantized counterparts, highlighting the efficiency gains of full integer quantization. The accompanying Fig. 4 provides a visual comparison of TF models and their quantized LiteRT counterparts, showing the distribution of accuracy, F1 score, and energy consumption across all 10 folds.



a) Accuracy differences



b) F1 score differences



c) GPU energy consumption differences

Fig. 4. Comparison between TF and quantized LiteRT models.

The goal of this analysis was to identify the optimal balance between accuracy, model size, and energy efficiency. While larger models generally exhibit better accuracy and robustness, their larger memory footprint renders them impractical for deployment on resource-constrained devices such as microcontrollers. Conversely, the results show that converting models to LiteRT significantly reduces their size with minimal impact on performance, particularly for non-quantized LiteRT models. Fully quantized models further optimize memory and energy consumption, albeit with a slight trade-off in accuracy.

IV. Conclusion and future work

In this paper, we optimize KWS models for resource-constrained devices by exploring trade-offs between accuracy, model size and energy efficiency. Our results demonstrate that larger models exhibit improved accuracy but with diminishing returns beyond $4\,\mathrm{MB}$ where accuracy plateaus at just over $80\,\%$. This highlights the importance of balancing compactness and performance for resource-constrained device deployment.

To process audio data, we use MFCC features for our KWS models due to their compact, domain-specific representation of spectral characteristics, paired with CNNs for effective structured data processing. While effective, future work will investigate alternative spectral feature extraction methods, including inverted mel-frequency cepstral coefficients (IMFCC), linear frequency cepstral coefficients (LFCC), and power-normalized cepstral coefficients (PNCC) [15].

The conversion of TF models to LiteRT shows notable memory reductions while maintaining acceptable performance. Non-quantized LiteRT models retain the accuracy of their TF counterparts while reducing energy consumption by $82.38\,\%$ and memory by $67\,\%$, while fully quantized LiteRT models provide even greater efficiency gains, with memory reductions averaging over $91\,\%$ and energy consumption improvements of $86.53\,\%$, albeit with a small trade-off in performances. These results demonstrate the potential for deploying such models on low-power devices, emphasizing the feasibility of balancing compactness and accuracy through careful optimization.

Based on our comprehensive evaluation, the architecture that achieves a balanced trade-off between performance and computational efficiency for KWS applications on resource-constrained devices is Model 4. This model balances performance and efficiency with both original and converted versions, maintaining accuracy and F1 scores exceeding 0.8 while keeping energy consumption low. When converted to a fully quantized LiteRT format, it dramatically reduces both memory footprint (from $1.84\,\mathrm{MB}$ to just $0.16\,\mathrm{MB}$) and power consumption without compromising performance.

Future work in this research involves the deployment of the optimized KWS models on a microcontroller to validate real-time performance. The deployment platform is the ESP32-S3-EYE, a cost-effective device designed for AI and IoT applications. The LiteRT models will be converted into the C array format and embedded directly into the firmware using the Espressif IoT Development Framework (ESP-IDF) and LiteRT for Microcontrollers. This approach ensures efficient resource management and real-time processing capabilities. Through practical testing, we aim to validate accuracy, inference speed, and energy efficiency, paving the way for further refinements and broader applications. Ultimately, this research contributes to the development of robust, efficient, and scalable tiny machine learning solutions for resource-constrained environments.

Acknowledgment

This research was supported in part by ERASMUS+ Teaching Artificial Intelligence (TAI), 2024-1-Sl01-KA220-HED-000252673 funded by the European Union. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

References

[1] J. Chen, T. H. Teo, C. L. Kok, and Y. Y. Koh, "A novel single-word speech recognition on embedded systems using a convolution neuron network with improved out-of-distribution detection," *Electronics*, vol. 13, no. 3, p. 530, 2024.

[2] I. Froiz-Míguez, P. Fraga-Lamas, and T. M. Fernández-CaramésS, "Design, implementation, and practical evaluation of a voice recognition based iot home automation system for low-resource languages and resource-constrained edge iot devices: A system for galician and mobile opportunistic scenarios," *IEEE Access*, vol. 11, pp. 63 623–63 649, 2023.

[3] C. Cioflan, L. Cavigelli, M. Rusci, M. de Prado, and L. Benini, "On-device domain learning for keyword spotting on low-power extreme edge embedded systems," 2024. [Online]. Available: https://arxiv.org/abs/2403.10549

[4] C. Bernal-Ruiz, F. Garcia-Tapias, B. Martin-del Brio, A. Bono-Nuez, and N. Medrano-Marques, "Microcontroller implementation of a voice command recognition system for human-machine interface in embedded systems," in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, 2005, pp. 5 pp.–591.

[5] J. Wang and S. Li, "Keyword spotting system and evaluation of pruning and quantization methods on low-power edge microcontrollers," 2022. [Online]. Available: https://arxiv.org/abs/2208.02765

[6] A. H. Michaely, X. Zhang, G. Simko, C. Parada, and P. Aleksic, "Keyword spotting for Google Assistant using contextual speech recognition," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2017, pp. 272–278.

[7] E. J. Arthi and M. Jagadeeswari, "Control of electrical appliances through voice commands," *IOSR Journal of Electrical and Electronics Engineering*, vol. 9, pp. 13–18, 01 2014.

[8] V. Tiwari, "MFCC and its applications in speaker recognition," *International journal on emerging technologies*, vol. 1, no. 1, pp. 19–22, 2010.

[9] I. López-Espejo, Z.-H. Tan, J. H. L. Hansen, and J. Jensen, "Deep spoken keyword spotting: An overview," *IEEE Access*, vol. 10, pp. 4169–4199, 2022.

[10] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[11] K. Khotimah, A. B. Santoso, M. Ma'arif, A. N. Azhiimah, B. Suprianto, M. S. Sumbawati, and T. Rijanto, "Validation of voice recognition in various google voice languages using voice recognition module v3 based on microcontroller," in *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, 2020, pp. 1–6.

[12] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[13] V. A. Hadoltikar, V. R. Ratnaparkhe, and R. Kumar, "Optimization of mfcc parameters for mobile phone recognition from audio recordings," in *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, 2019, pp. 777–780.

[14] F. Zheng and G. Zhang, "Integrating the energy information into mfcc." in *INTERSPEECH*, 2000, pp. 389–392.

[15] M. Mohammadi and H. R. S. Mohammadi, "Robust features fusion for text independent speaker verification enhancement in noisy environments," in *2017 Iranian Conference on Electrical Engineering (ICEE)*. IEEE, 2017, pp. 1863–1868.