Adversarial robustness of keyword spotting on Cortex M4/ Cortex M4-M7 based wearable platform

Dakshit Babbar, *2020EE30163,* Vibhor Sengar, *2020EE30313,*
Prof. Rijurekha Sen, *Supervisor,* Prof. Saurabh Gandhi, *Co-Supervisor*

***Abstract*—Voice-activated devices, prevalent in diverse applications, rely on detecting specific keywords in input audio to initiate designated processes. Keyword spotting (KWS) powered by machine learning (ML) models constitutes a crucial element in the functionality of these devices. While achieving high accuracy is imperative, ensuring adversarial robustness is equally vital to fortify against potential attacks aiming to disrupt the intended processes triggered by detected keywords. While prior research has emphasized implementing keyword spotting with state-of-the-art precision on general-purpose Cortex-A processors using Neural Networks, our work shifts its focus to resource-constrained Cortex-M processors, particularly for wearable platforms. We pursue this objective by employing simpler yet efficient methods such as Tsetlin Machines (TM). Our contributions encompass the identification of the optimal model size for the resource-constrained boards, accuracy enhancement through various techniques, and effective verification of adversarial robustness of the trained model.**

***Index Terms*—Keyword Spotting, Adversarial Robustness, Tsetlin Machine, EdgeML.**

I. Introduction

VOICE-activated devices, such as Google Home and Alexa, detect a variety of keywords from a fixed vocabulary to trigger different functions. For example "play" or "open" to play a song from some library or open the door on command. We term these tasks as Keyword Spotting or KWS. Typically a KWS pipeline takes audio as input, extracts features from it in form of MFCC (Mel-Frequency Cepstral Coefficients), passes these to an ML model which then runs an inference, detects the keyword and passes control to the process that needs to be executed.

Devices that run a KWS pipeline can either run the inference on cloud or on the device itself. Inference on the device has various benefits than inference on cloud. To name a few, *Low Latency* as network communications take more time and *Enhanced User Privacy* as network attacks can cause user data to be leaked. Hence such tasks are increasingly being implemented through on-device computation instead of on cloud. Benefits of audio assistance devices can largely be seen in applications where such devices are wearable. For instance, helping individuals with disabilities or assisting people with tasks that demand situational awareness, like helping to make a call while cycling. Therefore it is required to design an accurate KWS model that fits on small wearable platforms that contain resource-constrained microprocessors like Cortex-M4/M7 series microprocessors.

Models used for KWS are vulnerable to adversarial machine learning based attacks. In such attacks an adversary will try to play some noise from some remote speaker to interfere with the input audio in order to fool the ML model to give a wrong inference. The attack can either contain some random noise that just results in an incorrect inference (untargeted attack) or it could be intelligently crafted to get some specific inference value (targeted attack). Therefore it is required to make sure that the model being deployed for KWS possesses adversarial robustness.

Our Contributions, hence, focus on the aforementioned requirements; identifying the optimal architecture and size of the model to be deployed, enhancing the accuracy of the model by exploring different techniques and effectively verifying the adversarial robustness of the trained model. In this journal, we first discuss the literature we followed in order to full fill our requirements. We then give specifications of the hardware platforms we worked on, followed by the description of our contributions which includes the experiments that we performed for each of those along with their analysis.

II. Literature Survey

We started by understanding the existing work on Adversarial Robustness of KWS [1]. It receives an audio input which is processed using conventional signal processing techniques i.e. extracting MFCC features, and passed on to the trained and stored ML based classification model to detect the keyword on a Raspberry-Pi (Quad-core Cortex-A72 Processor). We realised that a computationally expensive ML model (e.g. CNN, DNN) would not fit well on a resource constrained SensorTile board. Moreover the adversarial robustness of the ML mode deployed was verified by getting the accuracy of inference on a set of representative adversarial inputs which is nothing but the benign dataset perturbed with some noise. This way of verifying the adversarial robustness is not exhaustive as there might be some other adversarial input on which it was not tested that can breach the security. We went ahead to learn about Tsetlin Machine [2] which is a boolean proposition based classification ML algorithm. As compared to traditional Neural Networks, it is energy efficient, have smaller memory footprint and posses better adversarial robustness. We also considered using MEL spectrogram features along with that of MFCC features for training the TM as done by [3].

TABLE I
METRICS OF DIFFERENT SIZES OF TM DEPLOYED ON SENSORTILE

Clauses	Bit	RAM	Flash	Inference Time
10	32	26.27KB	51.25KB	0.586ms
1000	16	25.96KB	938.53KB	71ms
2000	16	-	1.08MB	-
2000	32	-	2.00MB	-

TABLE II
ACCURACIES OF DIFFERENT TM

Keywords	Clauses	PyTM-MEL	PyTM-MFCC	TMU-MEL	TMU-MFCC
4	500	82.41	87.64	81.76	86.53
4	1000	81.65	88.33	82.91	86.98
4	2000	81.53	88.62	83.35	87.42
6	1000	70.61	81.20	72.40	80.29
6	2000	70.05	82.42	73.98	81.29
10	1000	60.55	71.29	62.33	70.39
10	2000	57.52	71.57	63.50	71.47

We then read about how to effectively verify the adversarial robustness of TM's. This is important because adversarial attacks may compromise malware detection systems, influence user-entered instructions using keyword spotting tools, and present security risks in areas like self-driving vehicles, medical diagnostics, IoT devices, and more. The strategy we adopted was to formally verify the robustness of TM's via an exact encoding into a SAT problem as the one proposed by [4]. The authors used the translation of sequential counters into a logic formula [5] for the encoding of TM's into a propositional logic. Once the robustness property is converted into a SAT problem, one can formally verify robustness using SAT solvers. This method is more exhaustive as compared to the method of verifying robustness with representative adversarial inputs as we are not crafting any such adversrial inputs ourselves but allowing the SAT solvers to find vulnerabilities in our model objectively.

III. HARDWARE ACQUAINTANCE

The board that we use to deploy the KWS inference pipeline is the SensorTile (STEVAL-STLCS01V1). It consists of a 32-bit ultra-low-power ARM-based CortexM4 microprocessor along with various other on-chip sensors. It holds up to 1MB of flash space and 128KB RAM. In order to program this we used the STM Nucleo board (L073RZ) which has a built in ST-Link debugger that helps in programming any other board after connecting to our PC. We made use of the Software Suite that STM provides to program the board. In order to make ourselves familiar with the hardware and the software suite, we implemented some standard example programs provided by STM itself which includes an AudioLoop application, a DataLog application and a Bluetooth application.

IV. CONTRIBUTIONS

A. *Model Size Optimization*

The first task is to identify what is the optimal size of the TM that fits within the constraints of our hardware. There is no open source code for a TM that has been trained on audio inputs in our knowledge. Hence for the initial testing and identifying the optimal model size we considered using a TM trained on the MNIST dataset. We did so because the MNIST *Handwritten Digit Recognition* problem involving classification within 10 digits is very similar to the *Keyword Spotting Problem* involving classification within, say, 10 keywords.

We started with deploying a dummy TM with just 10 clauses per class in order to test the working of the model on the board. We then went ahead and deployed a bulkier TM with 2000 clauses per class which was unable to fit on the board. It's flash usage was double the flash present on the board. We hence tried to reduce the size of the model without changing its structure as explained next. Initially the model parameters that were being stored in the flash were stored in a int32 format. We tried to change it to int16 but that too was overflowing the flash by 80KB. We could not go below int16 as the maximum value of the parameter to be stored was 1568 (i.e. number of all literals that need to stored which is $28 \times 28 \times 2$).

As we were unable to fit the 2000 clause TM on the board we had to work with a TM with lesser number of clauses. We then trained and deployed a TM with 1000 clauses and int16 parameters. This TM was able to fit on the board using close to 91% flash, leaving room for any other information to be stored as well. Table I shows a compilation of all the attempts on fitting the TM that were made, along with their corresponding RAM and Flash usage and inference time on board.

B. *Improving Accuracy*

Once we had an optimised size of the TM that was able to fit on SensorTile, we then went ahead to train a model for the *Keyword Spotting Problem*. The dataset that we used for training the TM is the speech command dataset provided by Google which contains 1 second audio samples of keywords along with their corresponding labels. We considered a subset of this dataset with just 10 keywords. The 10 keywords that we considered are 'yes', 'no', 'stop', 'seven', 'left', 'right', 'up', 'down', 'backward', 'forward'. We initially took the contemporary approach of extracting features from audio files i.e. by taking the MFCC features. But we also experimented with the MEL spectrogram features of the audio to analyse the effect on the accuracies that these two features have. Figure 1 shows the MEL spectrogram features of 5 random audios of each of the 10 keywords that we considered

We used used two different libraries in python, one pyTsetlinMachine (PyTM) and the other Tsetlin Machine Unified (TMU). These provide with an implementation of

TABLE III
ROBUSTNESS ON THE PYTM-MEL

100 test instances												
	4 keywords 500 clauses			4 keywords 1000 clauses			6 keywords 1000 clauses			10 keywords 1000 clauses		
ϵ	solved	ϵ-robust	time (sec)	solved	ϵ-robust	time (sec)	solved	ϵ-robust	time (sec)	solved	ϵ-robust	time (sec)
1	100	65	1.02	100	77	20.01	100	67	9.97	100	77	10.42
3	76	14	129.59	50	0	215.06	56	0	219.75	52	0	241.60
5	74	0	120.49	58	0	200.95	69	0	158.87	66	0	184.23

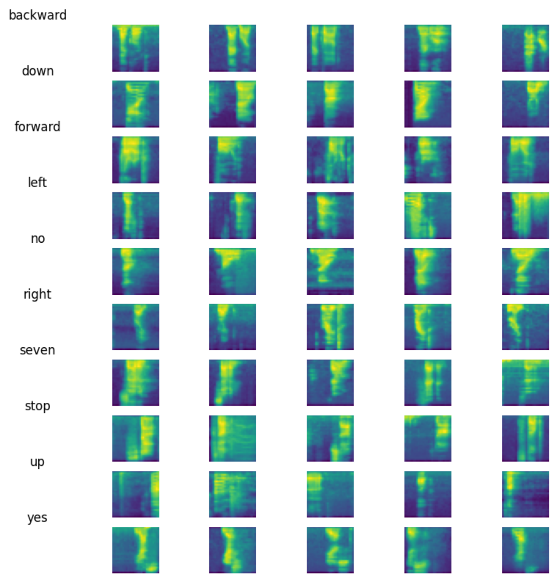

Fig. 1. MEL Spectrograms of keywords used for training the TM

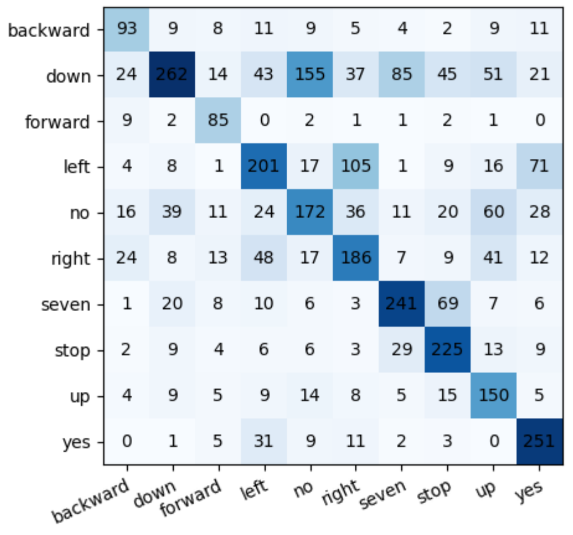

Fig. 2. Confusion Matrix for the 10 keywords detected using the trained TM

TM that we can train on our required dataset. The TMU library is a recent one which includes various new researches done on improving the architecture of the TM. We hence considered using different libraries in order to compare the results of the two.

The test accuracies for the respective model sizes is shown in table II. As we can see that increasing the number of clauses for a fixed number of keywords increases the accuracy. But as we increase the complexity of the problem by increasing the number of keywords to be spotted, the accuracies go down. the TMU library gives better results as compared to PyTM library for MEL spectrogram features. Whereas for the MFCC features PyTM gives better results. Overall, the MFCC features give better accuracies as compared to the MEL spectrogram features.

We also discovered that the keywords that sound very similar ('down' and 'no'), or those which even though sound different but have significantly similar looking spectrogram features ('seven' and 'stop') are the root cause of miss-classifications. Figure 2 shows the confusion matrix of the predictions made by the trained TM on he 10 keywords. It is clear that major miss-classification happens for the aforementioned keywords.

C. Verifying Adversarial Robustness

We demonstrate adversarial robustness using PyTM-MEL datasets. To assess adversarial robustness on input I, we test ϵ perturbation values $\epsilon \in \{1, 3, 5\}$. These numbers are the maximum bit-flips on I used by the SAT solver to assess robustness. We impose a timeout of 300 seconds on the solver for each test instance in case it takes exponential time. This setup is used for all experiments in this section.

TMs now construct one TM M for each target class during pre-processing. This implies the final trained model is a sorted team of TMs where Mi computes class i votes separately. The TM team outputs the class with the most vote given I (its vector representation). We run all robustness experiments on an Intel(R) Core(TM) i5-8300H CPU at 2.30GHz with 4 cores and 8 logical processors, Nvidia GeForce GTX 1050 Ti, and 8GB RAM

We conducted the robustness experiment for the PyTM-MEL dataset containing 4, 6, or 10 classes by randomly selecting 100 inputs. Table III presents the robustness results for the four PyTM-MEL models (excluding TMs with 2000 clauses) from Table II. Columns show the maximum number of bit-flips ϵ, the number of instances solved by the SAT solver, the amount of them being ϵ-robust, and the average time in seconds to solve these instances. "Solved" in this context means that the SAT solver either finds a valid perturbation leading to the change in the classification of the input image or that the model is ϵ-robust on this image. For $\epsilon = 1$, all instances were solved within the timeout of 300 seconds, and Models that are trained with more monomials showed more excellent resistance to input perturbations. However, increasing ϵ yields a more complex robustness encoding and solving more complex formulas built from larger models requires more time from the SAT solver.

V. Conclusion and Further Work

We worked on finding the optimal size of the model that fits under the hardware constraints. We used various methods to check which one will provide us with the best accuracy of the model to deployed. We also worked on effectively verifying the adversarial robustness of the TM to be deployed on the board. Future work after our contributions include finding a way to encode the conventional NN for verifying their adversarial robustness using SAT solvers and then comparing it with the robustness of the TM. We could also explore more ways to extract features from the input audio to get even better accuracies for more number of keywords.

References

[1] SpotOn: Adversarially Robust Keyword Spotting on Resource-Constrained IoT Platforms.

[2] A. Bakar, T. Rahman, R. Shafik, F. Kawsar, and A. Montanar, 'Adaptive Intelligence for Batteryless Sensors Using Software-Accelerated Tsetlin Machines', *in Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, Boston, Massachusetts, 2023, pp. 236–249.

[3] S. K. Gouda, S. Kanetkar, D. Harrison, and M. K. Warmuth, "Speech Recognition: Keyword Spotting Through Image Recognition," arXiv preprint arXiv:1803.03759, 2020.

[4] E. Przybysz, B. Bhattarai, C. Persia, A. Ozaki, O. C. Granmo, and J. Sharma, "Verifying Properties of Tsetlin Machines," arXiv preprint arXiv:2303.14464, 2023.

[5] C. Sinz, 'Towards an Optimal CNF Encoding of Boolean Cardinality Constraints', *in Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, (CP'05), Sitges, Spain, 2005, pp. 827–831. doi: 10.1007/11564751_73.