Primo documento che riguarda l'utilizzo di TinyML per PHM

TinyML-based approach for Remaining Useful Life Prediction of Turbofan Engines

Georgios Athanasakis[1], Gabriel Filios[1,2], Ioannis Katsidimas[1,2], Sotiris Nikoletseas[1,2], Stefanos H. Panagiotou[1]

[1]Computer Engineering and Informatics Department, University of Patras, Greece
Email: g_athanasakis@upatras.gr and {filiosg, ikatsidima, spanagiotou}@ceid.upatras.gr

[2]Computer Technology Institute and Press "Diophantus" (CTI), Greece
Email: nikole@cti.gr

2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA) | 978-1-6654-9996-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ETFA52439.2022.9921629

***Abstract*—In the recent years, artificial intelligence, machine learning and IoT technologies have enabled a great number of industrial applications with profitable results. Predicting the remaining useful life (RUL) of turbofan engines constitutes a successful example of industrial AI, and it has received thorough attention from the researchers worldwide, with numerous novel and effective methods being proposed in the literature. Meanwhile, TinyML is a recent trend that has emerged in the AI field and demonstrates, amongst others, promising potential to break through the existing barriers of trusting and deploying real-time critical industrial AI solutions. In this context, this paper aims to further contribute to the literature and demonstrate the realization of RUL predictions in the extreme edge via TinyML, using the popular C-MAPSS dataset from NASA Ames Research Center, X-CUBE-AI tool and STMF767ZI microcontroller for the deployment of ML models. We benchmark different ML algorithms, with a special focus on deep learning algorithms (LSTMs and CNNs). The results indicate that there is potential for deploying machine learning models for RUL prediction in resource-scarce IoT devices, with acceptable accuracy loss, while taking advantage of the benefits TinyML has to offer over cloud-based AI inference.**

***Index Terms*—Tiny Machine Learning, Remaining Useful Life (RUL), C-MAPSS, Prognostics and Health Management**

I. Introduction

A. Remaining useful life prediction

In the past decades, prognostics and health management (PHM) of complex engineering systems has attracted the research community and industrial practitioners. PHM is a framework that offers comprehensive yet individualized solutions for managing system health. Prognostics is essential for PHM because it supplies the decision maker with early warning about the expected time to system/subsystem/component failure and let him decide about appropriate actions to deal with this failure [1].

Prognostics focus on prediction making when a system or a component stops to perform as required. In other words, the prognostic algorithms predict the future performance or the remaining useful life (RUL) of a system or component by analyzing the extent of deviation and degradation from its expected normal operating conditions. In general, the health state of an item degrades linearly with its usage or operating cycle. However, the task of prognostics is not trivial due to the variation of operation conditions, environment, and complex nature of different parameters. Prognosis requires intensive degradation data of an item, such as lifetime data or run-to-failure history [2].

The increased availability of system condition monitoring data has facilitated the broader use of data-driven approaches for prognostics and health management (PHM). The underlying assumption of data-driven approaches is that the relevant information concerning the evolution of the system health and the failure time can be learned from past data. Consequently, in the recent years data-driven methods such as machine learning and deep learning have been widely used to solve RUL problems [3].

Accurately predicting the remaining useful life of turbofan engines is one of the most popular RUL problems in the literature and it is of great significance for improving safety of aero-engine systems. Complex sensor data exhibited by these engines can be used to predict possible failures beforehand, increasing machine reliability and availability, reducing maintenance costs and saving downtime or loss of equipment.

B. Tiny Machine Learning

Tiny Machine Learning (TinyML) is an emerging area of research that lies in the intersection of ML and embedded systems and enables ML capabilities on lightweight IoT devices, including cheap microcontroller units (MCUs) or digital signal processors, which operate at an average power of 1 mW or less [4]. The major constraints that TinyML paradigm has to encounter consists of the following limited hardware requirements: processing power, memory and clock speed.

Typically, TinyML runs on 32 bit MCUs with fewer than 500 KB of SRAM, a few MB of on-board flash memory, and a clock frequency lower than 200 MHz.

Machine learning algorithms are embedded into TinyML systems by adapting them to their resource constraints through a variety of techniques [5]. For example, to optimize Artificial Neural Networks (ANNs) pruning is used to remove some of the synapses (connections) and neurons (nodes) and quantization is employed to reduce the memory required to store numerical values, for instance by translating floating-point numbers (4 bytes each) to 8-bit integers (1 byte each). These techniques reduce the granularity with which a network can capture relationships and infer results. Hence, there's a necessary trade-off between model size and accuracy in TinyML, and the way in which these strategies are implemented is an important part of TinyML system design.

Given the great potential of TinyML to revolutionize multiple sectors, a number of libraries and tools for easing the implementation of ML algorithms on constrained platforms are available in the market [6]. Indicatively, TensorFlow Lite Micro (TFLM) is an open source, flexible, interpreter-based solution created by Google, that is designed for wide range of MCUs with only few kilobytes of memory. The core runtime just fits in 16 KB on an Arm Cortex M3 and can run many basic models. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation [7]. STMicroelectronics has developed the X-CUBE-AI proprietary toolkit that permits to integrate pre-trained NNs within STM32 ARM Cortex-M-based microcontrollers. It generates STM32-compatible C code from NN models generated by Tensorflow and Keras, or in the ONNX format [8]. Concerning frameworks for conventional ML models, m2cgen is one of tools available that can be used for transpiling Scikit-learn-trained models into a native code, e.g., Python, JAVA, etc [9].

C. *Contribution and motivation of this work*

TinyML, as a subset of embedded machine learning and edge AI, brings several benefits over cloud AI characteristics, that constitute significant barriers for trusting and deploying, in real time, AI solutions in the Industrial Internet of Things (IIoT) [10], [6], [11]. By performing on-device and near-sensor inference, data are not transferred and processed on the server-side. Thus, TinyML enables greater responsiveness with low latency, less risks for data breaches and privacy leaks, as well as increased autonomy due to reduced power consumption and energy cost associated with wireless communication, which at this scale is more power hungry than processing. In this way, TinyML-based approaches can contribute to the realization of real-time decision making systems for RUL prognosis, that must be decoupled from centralized infrastructure.

This constitutes the **motivation** of this work, as we aim to move beyond the conventional literature work on novel techniques and algorithms for RUL prognosis and investigate the development and deployment of TinyML models on low-power MCUs for this type of problems. However, we also state that TinyML techniques are not only useful for MCU-based deployments but for any large ML model that runs in the cloud and we wish to bring closer to the edge and the source of data under resource-constraints.

At the same time, we chose the C-MAPSS dataset since it considered as the benchmark dataset for RUL prognosis and its properties allow to relate the findings to a variety of other use cases in research and industry: the dataset contains multivariate time series, multiple fault modes and operating conditions, as well as different lifespans, and natural failures. Thus, we argue that by using the C-MAPSS dataset we derive general challenges that apply to industrial IoT applications.

Consequently, our focus is not only on the dataset itself, but on its properties and their implications for TinyML-driven approaches. Overall, we focus on testing and demonstrating the feasibility of RUL prognosis in the extreme edge through TinyML techniques and hence, absolute model accuracy is not the priority of this work and we do not implement and compare with state of the art models that won't fit to low-power MCUs.

Our contribution is summarized as follows:

- This work is the first, to the best of our knowledge, to explore TinyML techniques for RUL prognosis of turbofan engines.
- We demonstrate the exploitation of industrial AI application at the far edge using resource-constrained devices.
- We benchmark machine learning and deep learning models in the context of Industrial TinyML applications.

Roadmap of the paper. The rest of this paper is organized as follows. Section II elaborates on the most recent related works in turbofan engine RUL prediction and TinyML research. We present the methodology of our work in Section III, by describing the dataset used in the experiments, the models used as well as the overview of the TinyML optimization procedure. Afterwards, section IV contains the outcomes of the experiments and finally, in V we summarize the subject of the work and report our next steps for further exploitation.

II. Related Work

In this section we present some research works which can be considered as the most related to the current paper. The section is divided in two parts, covering related work in RUL estimation for turbofan engines and industrial TinyML research, accordingly.

A. *Data-driven estimation of remaining useful life*

The estimation of Remaining Useful Life in turbofan engines using data-driven approaches has been extensively studied in the last decade. Conventional machine learning algorithms, for example tree-based models like Random Forest, Gradient Boosting Machine and especially deep learning methods, such as CNN and LSTM, have been shown to be effective for turbofan engine RUL prediction [2], [12], [13].

Long short-term memory (LSTM) based methods have achieved state of the art performance for RUL prediction due to their inherent and robust capability of modeling sequential sensor data [14], [15]. Besides standard LSTM employment, several LSTM-based approaches, such as attention-based LSTM [16], or semi-supervised setup with restricted boltzmann machine, LSTM and genetic algrotihm [17], were proposed to further improve RUL prediction accuracy. Deep Convolutional Neural Network (DCNN) has excellent learning ability, which is mainly achieved using multiple nonlinear feature extractions. It can automatically learn hierarchical representations from data. Time window approaches for DCNNs are proposed in [18] and [19] as well as synthetic approaches such as attention-based DCNN [20]. In addition, to solve the issues in DCNN prediction methods, a model combining DCNN and LightGBM (an improvement of XGBoost algorithm) for predicting the RUL of aircraft engines is proposed in [21]. The authors use the deep features extracted by DCNN as the input of LightGBM to get more accurate prediction results.

Overall, the lower bound of predicting the turbofan engine RUL with ML techniques is summarized with the following best scores per dataset for the RMSE and Score evaluation metrics: FD001 (11.81/223), FD002 (18.34/2550), FD003 (12.51/279.36), FD004(19.41/2930.65) [20]. However, the different use of test sets, data preprocessing decisions in some or all the datasets, and error metrics, amongst the literature works, results to variations in the top results and constitutes an important quantitative comparability challenge for the research on C-MAPSS dataset, as the authors of [12] also note.

B. *TinyML in industrial applications*

To the best of our knowledge, there is a gap in the literature concerning RUL prognosis with TinyML techniques, and in general similar research in industrial applications is also limited. We therefore present an indicative list of the most relevant works, which on the one hand are associated to industrial use cases, in the broad context, and on the other hand provide similar methodology with our work.

In [22], the authors have deployed a TinyML solution for visual defect detection of mechanical components in die-cast aluminum. Using the free STM32Cube.AI tool they designed an optimal ANN according to STM32 micro-controller resource constraints, resulting in 9% accuracy loss and 87% drop in RAM consumption.

Similarly, in [23], an embedded implementation of a Deep Neural Network using the STM32Cube.AI tool is presented to detect and classify powertrain load on a STM32F469AI microcontroller. The authors used vibration signal as input data for the DNN and compared classification accuracy of a memory compressed DNN against an uncompressed DNN, demonstrating an overall acceptable accuracy from the compressed DNN, with small deviations from the uncompressed one.

In [24], the authors present a comparison of different NNs architectures (CNN, LSTM, CNN LSTM, GRU, and CNN GRU), for estimating maximum releasable capacity of Li-Ion batteries using prognostic Li-Ion battery datasets provided by NASA. STM32Cube.AI tool and TensorFlow Lite for Microcontrollers (TFLM) are compared for CNN models, and the results demonstrate that the former outperforms TFLM in terms of average inference time [ms], RAM size including inputs buffer [KB], and Flash size [KB].

MLPerf Tiny is a suite of benchmarks for assessing the energy, latency, and accuracy of TinyML hardware, models, and runtimes to properly evaluate the tradeoffs between systems [25]. Anomaly detection of manufacturing audio is one of the problems included in this benchmark, using machine operating sounds from ToyADMOS dataset. The authors use autoencoder model (AE) which outputs an anomaly score and the benchmark results conclude to an AUC of 0.88.

The authors of [26] leverage tinyCNNs to control mini-vehicles for autonomous driving. In particular, they introduce an online predictor that can choose between different tinyCNN models at runtime—trading accuracy and latency—which minimises the inference's energy consumption by up to 3.2×. They compare different MCUs and TinyML platforms: STM32L476 (STM32Cube.AI), STM32L476 (CMISS-NN), NXPk64f (CMISS-NN, GAP8 (PULP-NN), and produced the best results on GAP8 MCU, reducing the latency by over 13× and the energy consumption by 92%.

III. Methodology

A. *Problem description and C-MAPSS dataset*

The turbofan engine degradation simulation dataset was released by the Prognostics CoE at NASA Ames [27]. The dataset was generated with the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) tool and it is based on the damage propagation model of the aeroengine, providing a realistic simulation for commercial turbofan engines.

Initially, each engine is operating normally and at some point during operation a fault occurs. Every engine has a different degree of wear at the beginning of the simulation. The dataset consists of the following features: time (in cycles), 3 operational settings and 21 sensor measurements, all in time series order and contaminated with sensor noise. In train set all engines are simulated until system failure. In test set the simulation stops before system failure. The dataset is further divided to 4 subsets FD001, FD002, FD003 and FD004 as shown in Table I. Fault mode 1 corresponds to HPC (High Pressure Compressor) Degradation and fault mode 2 corresponds to HPC and Fan Degradation.

Dataset	FD001	FD002	FD003	FD004
Train trajectories	100	260	100	248
Test trajectories	100	259	100	249
Train samples	20631	53759	24720	61249
Test samples	13096	33991	16596	41214
Conditions	1	6	1	6
Fault Modes	1	1	2	2

TABLE I: C-MAPSS dataset attributes

Potrebbe interessare dopo !

B. *Data Preprocessing*

After reviewing all sensors, it is found that some of them have constant values, hence we can remove them. Therefore, 14 sensors remained for each sub-dataset (sensor 2, 3, 4, 7, 8, 9, 11, 12, 12, 13, 14, 15, 17 and 20). All sensor values are then normalized with Min-Max normalization (Eq. 1).

$$x' = \frac{x - min(x)}{max(x) - min(x)} \quad (1)$$

A sliding window is applied to take advantage of time series information resulting to segmented data. Larger sliding window includes more information but also a higher risk of overfitting. We chose a sliding window with window size of 30 and step size of 1 as proposed.

Last cycles of an engine life are more significant than the initial cycles. Thus, a piecewise linear RUL function is applied, where a max RUL value is set if the true RUL is greater than this max value, as shown in Eq. 2. In this way, we ignore data whose true RULs are greater than the maximum limit to pay attention to the degradation data and we adopt a max RUL value of 125, as used in [19] and other related works.

$$RUL_{pw} = \begin{cases} RUL_{max}, & \text{if } RUL_{true} > RUL_{max} \\ RUL_{true}, & \text{otherwise} \end{cases} \quad (2)$$

Finally, after the aforementioned data preprocessing steps, each train set is splitted into the following datasets: train (for ML model building) and validation (for ML model to validate on unseen data) with a 80/20 % split, with caution that the segment of each engine is not separated.

C. Models and algorithms

In our experiments we implemented two different architectures, a smaller and a bigger one (in proportion to the resource-constraints of MCUs), for each algorithm of the following: LSTM, CNN, XGBoost, Random Forest. The rationale is to explore the model behavior in terms of evaluation and footprint metrics depending on the model size (see Section IV for differences between Small and Large models) and the impact of the optimization techniques (described in Section III-D).

For each model we present the exact architecture (see Tables II, III, V) and especially for the LSTM and CNN models we summarize their hyperparameters in Table IV. For Random Forest and XGBoost Large variations, the model parameters are selected after grid search cross validation. Furthermore, for the LSTMs and CNNs, we preferred an asymmetric loss function (Eq. 3) as proposed in [28], instead of a typical one (e.g. MSE), since it provides higher penalty on late predictions.

$$QUAD - QUAD = \begin{cases} 2ad_i, & \text{if } d_i < 0 \\ 2\left(a + (1 - 2a)\right) d_i, & \text{otherwise} \end{cases} \quad (3)$$

Where d_i = Predicted RUL - True RUL.

1) Long Short-Term Memory Networks (LSTM): LSTMs are the evolution of Recurrent Neural Networks (RNNs), that handle better the exploding and vanishing gradient problem. They constitute a very effective algorithm for time series data due to their ability to process sequences of data.

2) Convolutional Neural Networks (CNN): CNNs have the ability to handle input data of high dimensions and they are also able to perform remarkably on RUL predictions. CNNs have three essential layers: convolutional, pooling and a traditional fully connected layer. A convolutional layer uses sliding filters to convolve the input data, creating a feature map. Pooling layers reduce data dimensionality for the next layer and a fully connected layer receives the flattened features to output a prediction.

	Layer	Units	Activation
Small	(input_shape=30, 14)		
	LSTM	60	tanh
	LSTM	30	tanh
	Flatten	-	-
	Dense	30	relu
	Dense	15	relu
	Dense	1	relu
Large	(input_shape=30, 14)		
	LSTM	128	tanh
	Dropout(20%)	-	-
	LSTM	64	tanh
	Dropout(20%)	-	-
	LSTM	32	tanh
	Dropout(20%)	-	-
	LSTM	16	tanh
	Dropout(20%)	-	-
	Flatten	-	-
	Dense	64	relu
	Dense	32	relu
	Dense	1	relu

TABLE II: Small & Large LSTM architectures

	Layer	Filters	Units	Kernel size	Activ.
Small	(input_shape=30, 14)				
	Conv1D	64	-	8	relu
	Conv1D	32	-	6	relu
	Conv1D	16	-	3	relu
	MaxPooling1D (pool_size=2)	-	-	-	-
	Flatten	-	-	-	-
	Dense	-	32	-	relu
	Dense	-	1	-	relu
Large	(input_shape=30, 14)				
	Conv1D	128	-	8	relu
	Conv1D	64	-	6	relu
	Conv1D	32	-	4	relu
	Conv1D	16	-	2	relu
	MaxPooling1D (pool_size=2)	-	-	-	-
	Flatten	-	-	-	-
	Dense	-	32	-	relu
	Dense	-	1	-	relu

TABLE III: Small & Large CNN architectures

Hyperparam.	LSTM		CNN	
	Small	Large	Small	Large
Optimizer	Adam			
Init. learn. rate	0.01			
Learn. rate div. every X epochs	10/10	10/30	-	-
Epochs	50	100	80	150
Batch size	256	256	256	512
Loss function	$a = 0.2$	$a = 0.4$	$a = 0.2$	$a = 0.2$

TABLE IV: LSTM & CNN models hyperparameters

3) XGBoost: XGBoost (eXtreme Gradient Boosting) is an ensemble algorithm used for classification and regression tasks. It is a variant of Gradient Boosted Decision Trees with high performance in speed and prediction.

4) Random Forest (RF): Random Forest is an ensemble algorithm which builds multiple decision trees based on different samples. The output value is the mean prediction value of decision trees output in case of regression tasks.

	Parameters	Small Model	Large Model
XGBoost	n_estimators	15	100
	max_depth	5	5
	colsample_bytree	0.5	0.5
	learning_rate	0.4	0.1
	objective	QUAD-QUAD	QUAD-QUAD
Random Forest	n_estimators	10	30
	max_depth	10	10
	criterion	squared error	squared error
	max_features	sqrt	sqrt

TABLE V: XGBoost & Random Forest parameters

D. TinyML optimization techniques

In this section we outline the techniques used to optimize and reduce the model size for both of the small and large LSTMs and CNNs.

1) *Pruning*: is an optimization technique that zeroes out insignificant weights. It can potentially reduce size and latency. In this paper, 25% of the weights were gradually set to zero with a polynomial decay function.
2) *Weight clustering*: first weight clusters are created, and then the centroid value of each cluster is shared to reduce unique weight values, bringing potential for footprint improvements. We applied 12 clusters, and initialized cluster centroids with K-means++ algorithm.
3) *Quantization*: Model parameters are represented by default to 32-floating point numbers. Quantization reduces the precision of the parameters, resulting to smaller model size with decreased latency. To achieve best results we converted the parameters representation to 8-bit integer numbers.
4) *Sparsity preserving clustering*: First, pruning is applied with 25% sparsity. Then, 12 clusters are created with one centroid having the zero value preserving the 25% sparsity.
5) *Sparsity preserving quantization*: Initially, 25% sparsity was achieved through pruning, which was preserved after 8-bit integer quantization.

TensorFlow Lite and the TensorFlow Model Optimization Toolkit are used for these optimization techniques and are applied after training (post training optimizations). The ML models (Random Forest and XGBoost) are only converted to optimized C code using the m2cgen library. All the aforementioned models of the previous section and the corresponding optimizations are first implemented offline, in a local development workstation and only the inference part is conducted inside the MCU. In particular, inference is executed on the target device (STM32F67ZI MCU, 2 MB Flash, 512 KB RAM, 216 MHz clock speed) with X-CUBE-AI tool using the corresponding test set for obtaining the quality and footprint metrics, which are described in the following section.

As TensorFlow reports, pruning and clustering techniques bring size improvements via model compression (e.g. via gzip), which is not a compatible file type for MCUs. In the future, framework support for these techniques will bring footprint improvements for MCUs (structural pruning and structural clustering for LSTM and Conv1d layers) [29].

E. Metrics

Quality metrics

In alignment to state of the art, the metrics used to assess the quality (accuracy) of the models are Root Mean Squared Error (RMSE) and a Score function proposed by NASA for the given problem [30].

For engine remaining useful life estimation, early predictions are more useful than late ones, and predicting engine failure in advance can effectively avoid immense consequences. For this reason, the scoring function is asymmetric in terms of the true time of failure and emphasizes on late predictions by giving them higher penalty than early ones and a lower score means better performance. On the other side, RMSE is chosen as it gives equal weight to both early and late predictions.

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N} d_i^2} \tag{4}$$

$$Score = \begin{cases} \sum_{n=1}^{N} e^{-d_i/13} - 1, & \text{if } d_i < 0 \\ \sum_{n=1}^{N} e^{d_i/10} - 1, & \text{otherwise} \end{cases} \tag{5}$$

Where d_i = Predicted RUL - True RUL.

Footprint metrics

Flash and RAM memory are estimated in KBs with STM32CubeIDE. All test sets are passed through the target device with clock speed of 216 MHz to determine mean inference (prediction) time in milliseconds scale. Power Consumption Calculator (tool of STM32CubeIDE) is used to provide us with the mean energy consumption (in mA).

IV. Experimental Results

A. Evaluation on Quality metrics

In Table VI we provide the results for each model and its variation (small, large) and for each of the five aforementioned optimization techniques, per dataset (FD001-04). We have highlighted in bold the best results per dataset for each of the LSTM and CNN models and their variations.

Optimizing models leads to a slight worse score overall, but there are some cases where it leads to better score due to generalization. First of all, conventional machine learning algorithms (Random Forest and XGBoost) perform very poor for all datasets. Concerning neural networks (NNs), the Small LSTM performs best when unoptimized in FD002 and FD004 datasets, after pruning in FD001 and after sparsity preserving quantization in FD003. The Large LSTM has the best performance when unoptimized for all datasets except FD001, where the best score comes after clustering. For the Small CNN, best performance in FD001 and FD004 datasets results from the unoptimized model, in FD002 after quantization and in FD003 after clustering. Large CNN performs the best in FD001 and FD003 when unoptimized, in FD002 after clustering and in FD004 after pruning.

In Figure 1, we illustrate the tendency of the Score metric for all models and after every optimization technique indicatively for the FD001 dataset (similar illustration would be for the majority of datasets and models as well). After pruning, clustering and sparsity preserving clustering, the score remains relatively stable for all models. When quantization and pruning quantization is applied, the score tendency of Small LSTM and Large CNN is slightly increased, but it grows considerably on the Small CNN, and it remains stable for the Large LSTM.

B. Evaluation on Footprint metrics

In Table VII, we group the results in the "Optimization technique" column in Unoptimized and Quantization categories, as there are currently no differences in terms of footprint between the unoptimized model and the models that result after the application of pruning, clustering and sparsity preserving clustering. Similarly, there are no differences between the quantization and sparsity preserving quantization (the reasons are mentioned in Section III-D).

Footprint-wise, the best performing NN model is the small CNN, as shown in Table VII. We observe that the biggest reduction in terms of footprint takes place after the quantization of the Large CNN and we note that even its 85KB of Flash memory is an acceptable size for numerous MCUs. In particular, we freed 74.7% of Flash Memory and 25.7% of RAM, reducing mean inference time by 55.7% and mean energy consumption by 7%. However, the Small CNN has greater reduction in mean energy consuption (22.8%). XGBoost and Random Forest have very low RAM, mean inference time and mean energy consumption but quite high requirements in Flash storage. All in all, quantization results to the greatest reductions and thus it can be regarded as the most effective optimization technique. For example, we notice that the quantized Large CNN is smaller than the unoptimized Small CNN.

In Figure 2 we depict the Flash size needed before and after quantization of NN models, and the differences of small and large conventional machine learning models. The Flash size of CNNs is reduced up to 74.6% for both small and large variations. On the other hand, LSTMs have lower reduction, 35.9% for small and 14.2% for large variation. Similarly, when comparing small and large variations of Random Forest and XGBoost models, we have a difference of 66.4% and 84.6% respectively. In conclusion, Large Random Forest model needs the most, while quantized Small CNN needs the least in terms of Flash Memory.

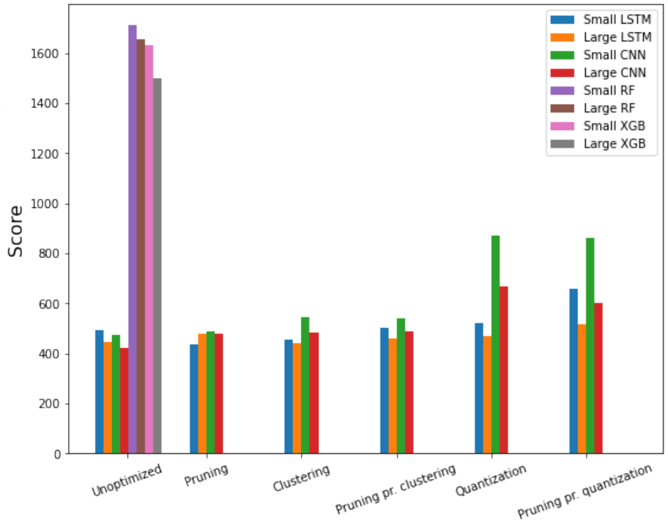

Fig. 1: Scores for FD001 dataset

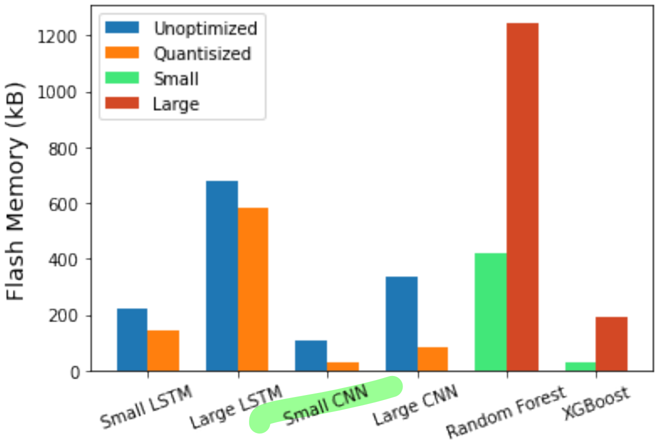

Fig. 2: Flash comparison between models

C. Discussion

Quality-wise, no model or optimization technique is a clear, consistent winner for every dataset. CNN is the best model overall in terms of quality and footprint metrics. In particular, Large CNN performs the best after the optimizations in the quality metrics, due to minimal loss or in some cases better score than unoptimized models. Considering device resource constraints, the optimal model is the Small CNN that is subject to quantization, since it only needs 26.6 KB of Flash Memory, 9.44 KB of RAM and consumes 62.68 mA of energy. At the same time, for the quantized Small CNN the quality-footprint trade-off is promising as there is an average 10% loss in the RMSE across the four datasets. More specifically, comparing the RMSE results of the quantized model to the unoptimized version, the accuracy loss corresponds to 11.8 %, 9.3 %, 2.4 %, 16.6 %, for every of the four datasets respectively.

Considering the resource-constraints of the algorithms in this work, this trade-off is acceptable with respect to the results

Model	Variation	Optimization	RMSE				Score			
			FD001	FD002	FD003	FD004	FD001	FD002	FD003	FD004
LSTM	Small	Unoptimized	14.94	23.79	**13.91**	20.92	495	**6531.35**	841.5	**4445.85**
		Pruning	15.17	27.06	13.92	22.6	**437.64**	7780.02	868.29	7718.34
		Clustering	15.36	26.04	14.14	**20.14**	454.56	8570.18	984.68	6910.24
		Spars. preserv. clust.	15.85	**21.91**	13.96	22.92	501.08	8082.80	923.24	6767.67
		Quantization	14.80	64.16	13.98	21.50	523.77	487856.92	929.01	7403.58
		Spars. preserv. quantiz.	**14.06**	68.87	14.12	22.68	658.71	561935.61	**811.09**	7964.29
	Large	Unoptimized	**14.84**	21.42	14.11	21.62	446.89	**4664.01**	**568.84**	**4374.69**
		Pruning	15.13	23.22	**13.63**	23.15	480.98	6409.68	708.14	5313.44
		Clustering	15.22	23.32	13.66	23.98	**442.44**	5800.87	698	5533.50
		Spars. preserv. clust.	15.37	23.61	14.03	23.60	460.82	6188.61	773.06	5837.29
		Quantization	14.86	22.63	13.71	**20.98**	468.40	5505.23	766.43	6033.30
		Spars. preserv. quantiz.	14.99	**19.74**	13.86	21.77	518.80	5983.32	728.39	5662.53
CNN	Small	Unoptimized	**15.08**	28.84	**15.60**	27.40	**472.75**	7857.42	686.83	**4443.28**
		Pruning	15.58	**28.38**	16.01	**24.25**	490.32	6776.59	658.02	5860.12
		Clustering	15.69	28.86	16.37	31.14	547.36	7172.29	**650.54**	6542.02
		Spars. preserv. clust.	15.83	29.74	16.46	30.85	538.55	7449.21	694.03	6256.17
		Quantization	16.87	31.52	15.97	31.95	871.56	**5711.64**	1108.61	5184.10
		Spars. preserv. quantiz.	17.32	30.13	15.90	32.79	862.86	8235.72	1213.02	5822.62
	Large	Unoptimized	**13.84**	29.63	**15.04**	29.95	**422.41**	5782.70	**430.99**	4177.95
		Pruning	14.7	25.85	15.56	**28.71**	479.27	3687.86	439.68	**3425.79**
		Clustering	14.88	**25.40**	15.33	29.05	484.45	**3459.39**	422.32	3568.60
		Spars. preserv. clust.	15.35	25.82	15.95	29.15	486.30	3715.07	440.86	3493.81
		Quantization	15.88	32.42	15.17	29.09	667.05	6402.30	900.65	3917.40
		Spars. preserv. quantiz.	16.46	31.11	15.15	31.79	601.34	5277.99	931.24	4082.32
RF	Small	-	17.30	35.82	15.15	32.17	1711.57	17356.19	1484.81	14294.52
	Large		17.14	33.97	15.00	31.46	1657.24	15148.70	1440.04	13055.38
XGBoost	Small		19.01	36.04	16.18	34.53	1633.63	15033.76	1213.05	13461.22
	Large		18.53	34.24	15.85	32.69	1497.21	12538.32	1194.96	9031.47

TABLE VI: Evaluation metrics of each model on target device (the best results per dataset for each of the LSTM and CNN models and their variations are highlighted in bold)

Model	Variation	Optimization	Flash (KB)	RAM (KB)	Mean inference time (ms)	Mean energy consumption (mA)
LSTM	Small	Unoptimized	224.3	13.36	15.09	91.26
		Quantization	143.84	11.87	8.39	86.95
	Large	Unoptimized	677.07	26.25	15.51	91.42
		Quantization	580.97	24.45	10.82	89.08
CNN	Small	Unoptimized	104.69	10	5.03	81.17
		Quantization	**26.6**	**9.44**	**1.81**	**62.68**
	Large	Unoptimized	333.44	18	15.71	91.50
		Quantization	**84.35**	**13.37**	**6.96**	**85.10**
RF	Small	-	418.71	1.6	0.013	1.39
	Large		1246.27	1.6	0.040	4.13
XGBoost	Small		29.61	1.6	0.011	7.93
	Large		192.92	1.6	0.08	1.06

TABLE VII: Footprint metrics of each model on target device

of the state of the art (see Section II-A), and it is expected to improve, since TensorFlow's optimizations will be enhanced in future work and will bring further reductions in model size, as mentioned in Section III-D.

We note that CNNs are more effectively optimized than LSTMs. The latter have very good results in the quality metrics but can not compete in terms of model footprint. In addition, considering the satisfactory results of the quantized Large CNN, for example in FD002 and FD004 datasets where it performs better than the unoptimized small CNN, we suggest to try creating large baseline models and then attempt to optimize them, to find the optimal trade-off between quality and footprint metrics.

We also discuss the poor performance of ML models (XGBoost and Random Forest) in terms of quality, as they can not compete against the neural networks. However, their small footprint in terms of RAM, energy consumption and inference time is appealing and since they are more interpretable (which is an important challenge for RUL decision-making [12]), we recommend exploring their implementation for TinyML scenarios, as in other problems they might perform better than NNs (see No Free Lunch Theorem) or the trade-off between accuracy and footprint might favor their usage.

Finally, to further prove the value and usefulness of the TinyML approach to RUL prediction, future steps include the benchmarking and deployment of complex models and their hybrid combinations (as listed indicatively in Section II), that have been proposed in the state-of-the-art for this problem and dataset, to low-power MCUs. The models that are most likely to be fit into MCUs are those based on solely NNs, as they can be better optimized and occupate a very small portion of the Flash memory of MCUs, comparing to conventional ML models.

V. Conclusion and future work

In this paper we studied the application of TinyML techniques for optimizing machine learning models in the context of remaining useful life (RUL) prognosis. We used the C-MAPSS dataset from NASA, STMF767ZI microcontroller and the X-CUBE-AI tool, to predict the RUL of turbofan engines using several models, namely LSTM, CNN, XGBoost and Random Forest, with two size variations for each one (small, large) and we compared the impact of model optimization techniques in terms of quality and footprint metrics.

The results demonstrate that it is possible to deploy quantized CNN models with as low as 26KB of Flash and 9KB of RAM memory, with an average 10% quality loss in the RMSE metric for embedded devices. The results are promising and considering future software and hardware improvements, we envision to provide motivation for additional research on TinyML for RUL prognosis and in general other industrial applications using time series sensor data.

Considering future extensions to this work, we first aim to optimize models through additional TinyML libraries and frameworks besides X-CUBE-AI, as well as other TinyML-specific algorithms that have been proposed in the literature, and provide benchmarks for various microcontrollers. In addition, another research venue to be investigated, in the long-term, is the on-device training and continual learning of models, to respond in sensor data distribution shift scenarios, towards robust and resilient embedded AI systems for the industry domain.

Acknowledgment

The present work was financially supported by the "Andreas Mentzelopoulos Foundation" and the zPasteurAIzer subproject under ZDMP H2020 RIA program (n° 825631).

References

[1] H. M. Elattar, H. K. Elminir, and A. Riad, "Prognostics: a literature review," *Complex & Intelligent Systems*, vol. 2, no. 2, pp. 125–154, 2016.

[2] Y. Wen, M. Fashiar Rahman, H. Xu, and T.-L. B. Tseng, "Recent advances and trends of predictive maintenance from data-driven machine prognostics perspective," *Measurement*, vol. 187, p. 110276, 2022.

[3] M. Schwabacher and K. Goebel, "A survey of artificial intelligence for prognostics." in *AAAI fall symposium: artificial intelligence for prognostics*. Arlington, VA, 2007, pp. 108–115.

[4] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly, 2020.

[5] G. Menghani, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better," *arXiv e-prints*, p. arXiv:2106.08962, Jun. 2021.

[6] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[7] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "Tensorflow lite micro: Embedded machine learning on tinyml systems," 2020.

[8] "X-cube-ai." [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html

[9] "m2cgen.ai." [Online]. Available: https://github.com/BayesWitnesses/m2cgen

[10] R. A. Khalil, N. Saeed, M. Masood, Y. M. Fard, M.-S. Alouini, and T. Y. Al-Naffouri, "Deep learning in the industrial internet of things: Potentials, challenges, and emerging applications," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11 016–11 040, 2021.

[11] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[12] S. Vollert and A. Theissler, "Challenges of machine learning-based rul prognosis: A review on nasa's c-mapss data set," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, pp. 1–8.

[13] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2306–2318, 2017.

[14] M. Yuan, Y. Wu, and L. Lin, "Fault diagnosis and remaining useful life estimation of aero engine using lstm neural network," in *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, 2016, pp. 135–140.

[15] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, 2017, pp. 88–95.

[16] Z. Chen, M. Wu, R. Zhao, F. Guretno, R. Yan, and X. Li, "Machine remaining useful life prediction via an attention-based deep learning approach," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 3, pp. 2521–2531, 2021.

[17] A. Listou Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Reliability Engineering & System Safety*, vol. 183, pp. 240–251, 2019.

[18] G. S. Babu, P. Zhao, and X.-L. Li, "Deep convolutional neural network based regression approach for estimation of remaining useful life," in *Database Systems for Advanced Applications*. Springer International Publishing, 2016, pp. 214–228.

[19] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.

[20] A. Muneer, S. M. Taib, S. M. Fati, and H. Alhussian, "Deep-learning based prognosis approach for remaining useful life prediction of turbofan engine," *Symmetry*, vol. 13, no. 10, 2021.

[21] L. Liu, L. Wang, and Z. Yu, "Remaining useful life estimation of aircraft engines based on deep convolution neural network and lightgbm combination model," *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, pp. 1–10, 2021.

[22] D. Pau, F. Previdi, and E. Rota, "Tiny defects identification of mechanical components in die-cast aluminum using artificial neural networks for micro-controllers," in *2021 IEEE International Conference on Consumer Electronics (ICCE)*, 2021, pp. 1–4.

[23] S. Akhtari, F. Pickhardt, D. Pau, A. D. Pietro, and G. Tomarchio, "Intelligent embedded load detection at the edge on industry 4.0 powertrains applications," in *2019 IEEE 5th International forum on Research and Technology for Society and Industry (RTSI)*, 2019, pp. 427–430.

[24] G. Crocioni, D. Pau, J.-M. Delorme, and G. Gruosso, "Li-ion batteries parameter estimation with tiny neural networks embedded on intelligent iot microcontrollers," *IEEE Access*, vol. 8, pp. 122 135–122 146, 2020.

[25] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau *et al.*, "Mlperf tiny benchmark," *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[26] M. de Prado, M. Rusci, A. Capotondi, R. Donze, L. Benini, and N. Pazos, "Robustifying the deployment of tinyml models for autonomous mini-vehicles," *Sensors*, vol. 21, no. 4, 2021.

[27] A. Saxena and K. Goebel, "Phm08 challenge data set," 2008.

[28] D. Rengasamy, B. Rothwell, and G. P. Figueredo, "Asymmetric loss functions for deep learning early predictions of remaining useful life in aerospace gas turbine engines," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.

[29] "Tensorflow model optimization guide." [Online]. Available: https://www.tensorflow.org/model_optimization/guide/clustering#overview

[30] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *2008 International Conference on Prognostics and Health Management*, 2008, pp. 1–9.