Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.
Digital Object Identifier XX.XXXX/YYY.20XX.ZZZZZZZ

Real-time series arc fault detection and appliances classification in AC networks based on competing convolutional kernels

FRANCESCO FERRACUTI[1] , RICCARDO FELICETTI[1] , LUCA CAVANINI[1,2] , PATRICK SCHWEITZER[3] , AND ANDREA MONTERIÙ[1] .

[1]Department of Information Engineering, Università Politecnica delle Marche, Ancona, 60131, Italy, (e-mail: f.ferracuti,r.felicetti,l.cavanini,a.monteriu@staff.univpm.it)
[2]Industrial Systems and Control Ltd, Culzean House, 36 Renfield Street, Glasgow G2 1LU, UK (e-mail: l.cavanini@isc-ltd.com)
[3]Université de Lorraine, Institut Jean Lamour (IJL), CNRS, Nancy F-54000, France (e-mail: patrick.schweitzer@univ-lorraine.fr)

Corresponding author: Francesco Ferracuti (e-mail: f.ferracuti@staff.univpm.it).

This work was not supported by any organization.

ABSTRACT This paper presents a method to detect and classify series arc faults affecting domestic AC electrical circuits by the analysis of electric current time series data, based on the HYDRA (HYbrid Dictionary-Rocket Architecture) algorithm, a fast dictionary method for time series classification employing competing convolutional kernels. The key novel contributions are twofold: i) competing convolutional kernels are suitable to effectively extract features representing an effective set of arc fault detection indicators, and ii) the classification performed in this way is feasible to be executed in real time. The proposed method is validated using a public database, where data from 13 different types of loads is collected according to the IEC 62606 standard. To reduce inference time and optimize the algorithm for embedded control units, a feature reduction strategy is employed. The effectiveness of the proposed method is demonstrated through experimental tests conducted under both arcing and non-arcing conditions and across different load types. Moreover, its accuracy is also tested in case of transients caused by operational changes in common electrical appliances. Achieved results show a detection accuracy of approximately 99%, with appliance classification performance around 98%, with inference times ranging from 2.8 ms to 172.0 ms while executing the algorithm on an ARM Cortex-based board.

INDEX TERMS Electrical fault detection, Deep learning, Edge AI, Electrical safety.

SUPPLEMENTARY MATERIAL

The dataset and the code are available on Zenodo [1] at https://doi.org/10.5281/zenodo.15279701.

I. INTRODUCTION

ARC faults are unexpected electrical discharges that can occur in various areas of a home or in nearly any electrical fixture. These faults typically arise from long-term wear on wires due to normal load or overload, loose connections at junctions, or damage to insulation caused by external factors [2]. If an arc fault persists for an extended period, the energy generated can potentially ignite a fire accident. According to the Forum for European Electrical Safety (FEEDS), approximately 270000 domestic fires of electrical origin, representing about 20% to 30% of all household fires, are reported annually across the EU. These accidents result in an average of 1000 fatalities and 20000 injuries each year, along with property damage estimated at 6.25 billion euros [3]. Similarly, the National Fire Protection Association (NFPA) reports that, between 2012 and 2016, there were 44880 home fires attributed to electrical failures or malfunctions every year. These incidents led to approximately 440 civilian deaths, 1250 injuries, and an estimated annual direct property damage of 1.3 billion dollars. As a result, there is growing attention towards the development of Arc Fault Detection Devices (AFDDs) for AC domestic networks.

Arc faults typically exhibit a distinctive stochastic change in current magnitude which makes it challenging to detect this type of fault using conventional AFDDs. Furthermore, the literature highlights that the characteristics of arc faults would be distinct and complex, particularly in the presence of nonlinear or switching loads [4]. Because of this, arc fault detection and then the arc fault isolation system, having by the capability to catch and classify the appliance involved in the fault, require customized and advanced signal processing techniques capable of extracting arc fault signatures [5]. Current time series vary based on loads and the circuit state (normal or arcing). The waveform of the normal current

varies depending on the load type: some exhibit an almost sinusoidal shape, while others feature flat shoulders or abrupt changes. When an arc fault occurs, various abnormal behaviors can be observed, such as impulses or spikes, an increase in high-frequency harmonic components, a decrease in the conduction angle, abrupt changes in the AC signal, waveform distortion in the time domain, and amplitude reductions. The variability, complexity, and chaotic nature of current time series during arc faults, along with the multitude of possible scenarios, pose significant challenges in designing reliable arc fault detection methods [6].

Various arc fault detection and classification methods based on single or multi-criteria approaches have been considered in the literature. Initially, several approaches based on Neural Networks (NNs) have been proposed. In [7], the authors propose a method for residential AC series arc fault detection using sparse representation and a fully connected NN. This approach, tested on a personal computer, achieves a classification accuracy of 94.3% across ten classes.

The authors of [8] use time and frequency domain features and a fully connected NN to achieve 99% detection accuracy. The algorithm is also deployed on a MicroController Unit (MCU). In [9], gray image analysis and a CNN are adopted to obtain 99% detection and 98% classification accuracy. The algorithm is deployed on a Field Programmable Gate Array (FPGA).

Similar results are presented in [10] using image analysis and recurrence quantification. The authors of [11] use time and frequency indicators and a Back-Propagation (BP) NN. The algorithm, implemented on a PC, reaches up to 99% in detection and 100% in load classification. In [12], a Convolutional Neural Network (CNN) is introduced, reaching a maximum arc fault detection accuracy of 99.47% at a 10 kHz sampling rate. The model is implemented on a Raspberry Pi 3B.

In the last couple of years, further advancements have been presented in the literature, considering both the introduction of new features and the use of different classification methods. In [13], the authors propose a detection method based on an improved recurrence quantification analysis and a BP NN to detect series arc faults under different vibration frequencies and amplitude conditions. In [14], the authors propose a lightweight arc fault detection method that integrates load classification into fault detection. Load classification is developed using an event-based method, which classifies load into resistive, inductive, and switchable loads, according to the turn-on patterns. Then, the arc detection algorithm is developed for each load category, which is achieved through sequential forward floating selection. In [15], an Adam-optimized BP NN is deployed on a MCU, reaching 98% accuracy. In [16], a teacher-student distillation approach with a CNN is employed and then deployed on a MCU, obtaining 99% accuracy in both arc fault detection and device classification. In [17], the authors compare the total harmonic distortion with a threshold, achieving 98.8% accuracy. In [18], deep NNs are employed to detect faults using low sampling frequency (60 Hz) current data, achieving accuracy around 99%. In [19], time-frequency features are employed and redundancy is reduced using zero-phase component analysis. Finally, arc fault detection is performed by a broad learning system. In [20], an arc fault detection model based on L2/L1 norms and a classification algorithm is proposed. This method, tested on a personal computer, achieves 98.74% accuracy across six load types. In [21], the authors introduce a lightweight One-Dimensional (1D) CNNs for arc fault classification. The model, tested on a Jetson Nano GPU, reaches 98.05% accuracy with four classes and an inference time of 5.26 ms. In [22], the authors present an arc fault detection framework that utilizes the relative position matrix and deep convolutional network to detect alternating current series arc faults. This method, tested on a personal computer, achieves 98.74% accuracy across six load types. A transformer NN for AC series arc-fault detection is proposed in [23]. The transformer is tested in a personal computer, achieving high classification accuracy. In [24], the authors present a feature selection method combined with a light gradient boosting machine to optimize feature quality and classification performance for arc fault detection. The algorithm is tested on a Raspberry Pi 4B. In [25], the authors propose a detection method using signal-type enumeration and a zoom circular convolution algorithm. Since the coupling method, which is employed to capture high-frequency differential signals, confuses normal signals with arcing ones in dimmer loads, to address the issue, in [26] the authors also present a time series reconstruction method based on spectral features.

Most AI-based methods for arc fault detection in the literature rely on expensive computing architectures such as GPUs or CPUs. While promising, these advanced AI approaches are often too computationally demanding to be implemented on industrial chips typically used in Arc Fault Circuit Interrupters (AFCIs). In fact, any AFCI solution must operate in real-time, processing data and reacting within a predefined tripping time to be considered acceptable.

Additionally, despite advancements in detection methods, accurately identifying arc faults remains challenging due to the complex waveforms of arc fault currents caused by varying loads, and particularly in the presence of nonlinear or switching loads [4].

According to the experimental tests performed on commercial AFDDs in [27], their performance are poor, with less than half of the selected products passing standard tests and failing in more diverse appliance or complex circuit configuration scenarios. Therefore, their reliability still needs improvements.

Most proposed methods in the literature fail to account for the transient behaviors of appliances during transitions, such as from the off state to the on state, from startup to normal operation, and eventually to the arc fault state. These operational changes in common electrical appliances can lead to false alarms or missed detections.

To address the mentioned issues, we propose an AC

arc fault detection method, based on the HYDRA (HYbrid Dictionary-Rocket Architecture) algorithm [28]. This is a computationally simple, fast, and accurate dictionary method for time series classification using competing convolutional kernels, incorporating aspects of both Rocket and conventional dictionary methods. In [28], the authors showed that HYDRA is faster and more accurate than the most accurate existing dictionary methods and it gives comparable or higher accuracy of other dictionary methods which represent prominent approaches to time series classification.

In this work, the algorithm is developed in embedded board and the execution time is estimated on MCUs and MicroProcessor Units (MPUs). Its ability to extract diverse discriminative features for arc detection across various types of loads significantly enhances the generality and accuracy of arc detection under complex operating load conditions. The primary contributions of this work can be summarized as follows:

- a dictionary method using competing convolutional kernels is proposed for the classification of arc faults, yielding promising results;
- the algorithm is tailored for deployment on embedded boards and its computational performances are analyzed, proving its real time feasibility;
- the classification accuracy is also assessed during transient events by analyzing the False Positive (FP) and False Negative (FN) rates resulting from operational changes in common electrical appliances.

The paper is organized as follows. Section II describes the methodology, the HYDRA algorithm for feature generation, and the linear SVM employed for classification. The general experimental test bench used to generate a series arc faults in an electrical network, in order to record the current electrical signatures, is described in Section III. In Section IV, the proposed methodology is evaluated with real data of various load types and working conditions. The results are discussed in Section V, where FPs and FNs are analyzed on the entire time series. In Section VI, the method is tested on several MCUs and MPUs to prove its feasibility in real-time. The methodology is validated by the ablation study in Section VII, while Section VIII provides a comparison with state-of-the-art methods, also testing different datasets.

II. METHODOLOGY

The proposed solution consists of the HYDRA algorithm [28] to generate diagnostic features, followed by a simple linear classification strategy to perform arc fault detection or isolation. The block diagram in Fig. 1 illustrates the main steps. Current measurements are collected into a time series with a predefined size l, then the HYDRA algorithm extracts a set of features, which are then fed into binary classifiers.

In the following, we discuss the HYDRA algorithm, the choice of the classifiers, and the formulation of classification problem. Further details on the labeling of each time series, the choice of hyperparameters, and the pre-processing of current measurements are then provided in Section IV.

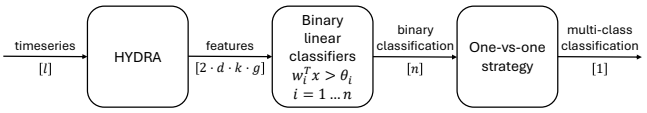

FIGURE 1. Block diagram illustrating the main steps of the proposed algorithm, with data dimensions indicated in square brackets.

A. FEATURE GENERATION: THE HYDRA ALGORITHM

HYDRA is a hybrid dictionary–rocket architecture for time series classification [28]. Random convolutional kernels are used to transform the input time series, as in Rocket. Randomness is mainly motivated by computational efficiency; nonetheless, it achieves a high accuracy. In contrast, data-driven and model-based alternatives for kernel generation require prior knowledge, additional tuning, or significantly increase computational effort, limiting HYDRA's general applicability.

The kernels are arranged into g groups with k kernels per group, as done in dictionary methods. Counting the kernels representing the closest match with the input time series for each group makes HYDRA similar to dictionary methods, while Rocket also applies the proportion of positive values pooling. Like other dictionary methods, HYDRA uses patterns which approximate the input, and produces features which represent the counts of these patterns. These counts can be treated as features, which are then used to train a classifier. However, unlike typical dictionary methods, HYDRA uses random patterns, represented by random convolutional kernels. HYDRA transforms the input time series using random convolutional kernels, which are organised into groups. Then, the algorithm counts the kernels in each group representing the closest match with the input at each time step. In other words, HYDRA treats each kernel as a pattern in a dictionary, and treats each group as a dictionary. The algorithm's hyperparameters are:

- The characteristics of the kernels:
  - the kernel length m;
  - the number of groups g per dilation;
  - the number of kernels k per group.
- The characteristics of the input sequence:
  - the sampling frequency f [Hz];
  - the time window t [s].
- The characteristics of the counting:
  - maximum and/or minimum response;
  - hard or soft counting.
- The inclusion of first-order difference.

Hard counting consists in counting the kernel with the maximum (and/or minimum) response at each timepoint; soft counting accumulates the maximum (and/or minimum) response for the kernel with the max (and/or min) response at each timepoint [28]. The weights are drawn from $\mathcal{N}(0,1)$ and then normalized (zero mean, unitary $L1$ norm). As for the input length, $l = f \cdot t$ holds.

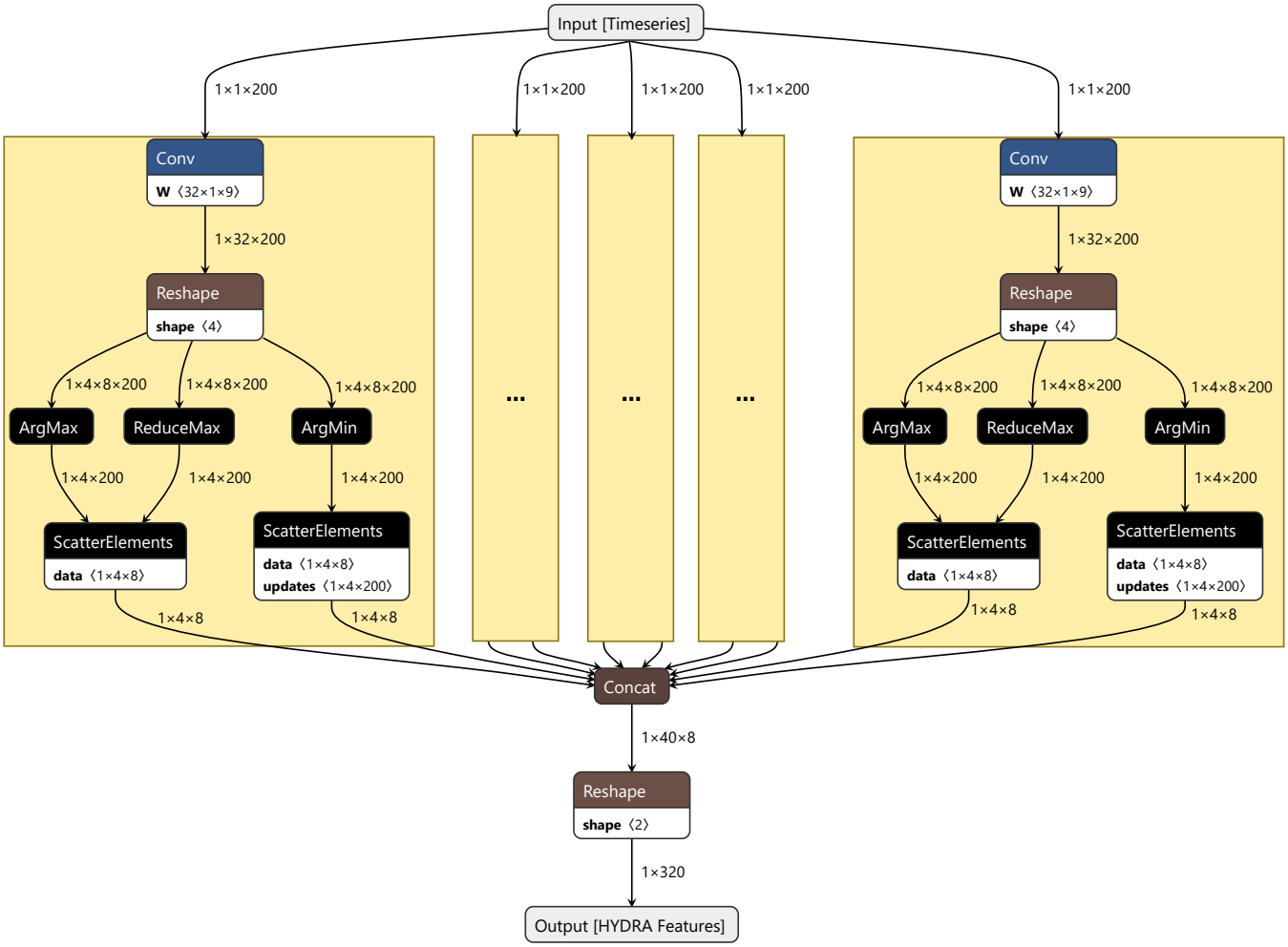

FIGURE 2. ONNX model of HYDRA. Each yellow rectangle represents a 1D convolution using a different dilation ($2^0, \ldots, 2^4$).

An exponential dilation $2^0, 2^1, ...2^{d-1}$ is employed by default [28], where

$$d = 1 + \left\lfloor log_2 \frac{l-1}{m-1} \right\rfloor . \tag{1}$$

Since HYDRA employs g groups per dilation [28], the total number of random kernels is $d \cdot k \cdot g$. As the number of kernels increases, both the number of features and the computational effort also increase. Conversely, memory occupancy depends on $k \cdot g$, while it is independent of d, as each dilation can be computed separately. Let us consider a fixed input and kernel length (l and m, respectively), so that the dilation d is also fixed according to (1). Hence, if $k \cdot g$ remains constant, both memory occupancy and complexity are, to a first approximation, the same, leading to a trade-off between k and g. If k increases, there are more competing kernels, which means a higher probability that the kernels closely match the input sequence. As a result, HYDRA behaves more similarly to a dictionary-based method. Conversely, if g increases, fewer kernels compete within the same group, leading to a higher degree of approximation (i.e., a lower likelihood of a close match). When $k = 1$, HYDRA essentially becomes a variant of Rocket. The input sequence is compared to a larger number of smaller groups, keeping the total number of features unchanged. This condition, however, enhances sparsity [28], meaning that an increasing number of features take a value of zero, and some kernels may never be counted at any time point.

The ONNX model of HYDRA is shown in Figure 2 in the case $m = 9$, $k = 8$, $g = 4$, $t = 20$ ms, and $f = 10$ kHz. Then, $l = f \cdot t = 200$ and, according to (1), the example considers $d = 5$ dilations.

Figure 2 illustrates the HYDRA algorithm pipeline. The entire sequence of length l is fed to several 1D convolution in parallel, i.e., one for each dilation ($d = 5$). Each dilation forms a branch in Figure 2 where the same operations are performed in each. The size of the weights for each convolution is $[kg \times 1 \times m]$, where kg is the number of kernels in that convolution (k competing kernels for each of the g kernel groups) and m is the kernel length. Each convolution returns a tensor with dimension $[1 \times kg \times l]$, representing the convolution of the signal of length l with each kernel. The reshape operator rearranges data to obtain

a dimension $[1 \times g \times k \times l]$. The ArgMax, ReduceMax, and ArgMin compare the results of the different competing kernels k (third dimension of the tensor) and returns three tensors of dimension $[1 \times g \times l]$. The ArgMax indicates the kernel k showing the maximum response, i.e., a value in $1, \ldots, k$ indicating the convolution with the largest magnitude, for each input sample $1, \ldots, l$. ReduceMax returns the actual convolution value with the largest magnitude, for each input sample $1, \ldots, l$. Finally, ArgMin indicates the kernel k showing the minimum response, i.e., a value in $1, \ldots, k$ indicating the convolution with the most negative magnitude, for each sample $1, \ldots, l$. Then, the *ScatterElements* function loops through the entire length l and accumulates the values. For the ArgMax and ReduceMax pair, the $[1 \times g \times k]$ output cumulates the largest correlations (soft counting, namely) for each kernel in each group. For the ArgMin, the $[1 \times g \times k]$ output counts the occurrence of most negative correlations (hard counting, namely) for each kernel in each group. The results are collected for each dilation, obtaining $2 \cdot d \cdot k \cdot g$ features in output (also see Figure 1).

B. ARC FAULT DETECTION AND ISOLATION: LINEAR CLASSIFIERS

In this work, we employ linear classifiers due to their low computational cost, fast inference time, and ease of implementation on resource-constrained hardware like MCUs and MPUs. We face three different classification problems.

- Fault Detection (FD): we define 2 classes, 1 representing healthy appliances, and 1 representing any appliance experiencing an arc fault conditions.
- Fault Detection and Isolation (FDI): we define 5 classes, 1 representing healthy appliances, and 4 representing the load class experiencing an arc fault conditions.
- Detailed Fault Detection and Isolation (DFDI): we define 14 classes, 1 representing healthy appliances, and 13 representing the appliance experiencing an arc fault conditions.

FD represents a binary classification problem, so a single binary classifier ($n = 1$) is used. For multi-class classification problems, instead, several strategies are available to decompose the task into a set of binary classification problems using $n > 1$ binary classifiers, such as one-vs-one and one-vs-rest. With a view to subsequent implementation on a MCU, a lasso regularization with a specified λ value is introduced in the training of the classifiers to induce sparsity, therefore reducing the number of actual input features and classification weights.

III. TEST SETUP

Arc faults are generated following the protocols outlined in the UL 1699 and IEC 62606 standards [29], [30]. Figure 3 illustrates the experimental test bench, designed to generate series arc faults in an electrical network, enabling the recording of current and voltage electrical signatures for database development.

In the following, we summarize the key aspects of the experimental setup. Further details about the dataset can be found in [10], [31] and the database of electrical signatures can be found in [31].

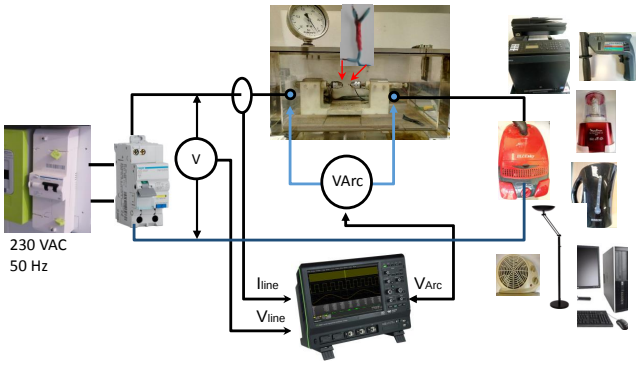

FIGURE 3. Experimental test bench.

The AC mains power source operates at 230 V and 50 Hz, supplying a load that may consist of a single appliance or multiple appliances connected in parallel. The household appliances used in this study include resistive devices, switching units, universal motors, and dimmer-based appliances. Table 1 reports the full list of appliances.

Measurements are taken using a Lecroy HDO 6104 oscilloscope with a bandwidth of 2 GHz. The sampling time is originally set to 1 MHz; if lower frequency analysis is required, sub-sampling is performed. The line current is measured at the output of the power source using a Lecroy AP 30 probe with a 100 MHz bandwidth. To ensure the fault remains active in the circuit, the arc voltage across the fault is also recorded using a TT-SI 9010 probe with a 70 MHz bandwidth. The collected data files are subsequently transferred to a computer for post processing and data analysis.

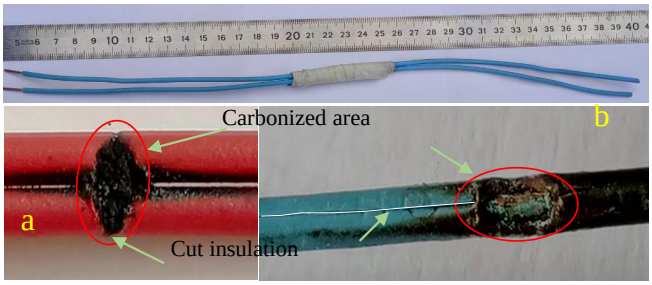

FIGURE 4. Cables preparation.

To create the arc fault, two copper cables, each 20 cm in length with a 5 cm slit in their insulation, are tightly bound together and wrapped. This setup creates a carbonized conductive path between the two conductors (as shown in Figure 4). The modified sample cable can be directly inserted into the circuit to replicate the arcing fault. The series arc fault is triggered at two different intervals using a relay switch.

Figure 5 illustrates the non-stationary trends of the arc voltage and line current signatures of a vacuum cleaner

TABLE 1. List of appliances.

Appliance	Load class
Electric Pump	Universal motor
Dell Desktop Computer	Converter: switching power supply
Dolce Gusto Espresso Machine	Resistive
Titan Drill	Universal motor (710 W)
Peugeot Drill	Universal motor (550 W)
Mewal Halogen Lamp	Converter: single-phase rectifiers
Blow Heater	Resistive
Severin 2200 Kettle	Resistive
Moulinex Mini Food Processor	Universal motor
Multi-functional Printer	Universal motor
Sabre Electric Jigsaw	Universal motor
Bluesky Optimo Vacuum Cleaner	Universal motor (1200 W)
Philips Vacuum Cleaner	Universal motor (1250 W)

with a nominal power of 1250 W during an arc fault over time. Additionally, the figure emphasizes the non-stationary behavior of the appliance, particularly during startup phase, despite the absence of arc faults (i.e., consistently null arc voltage).

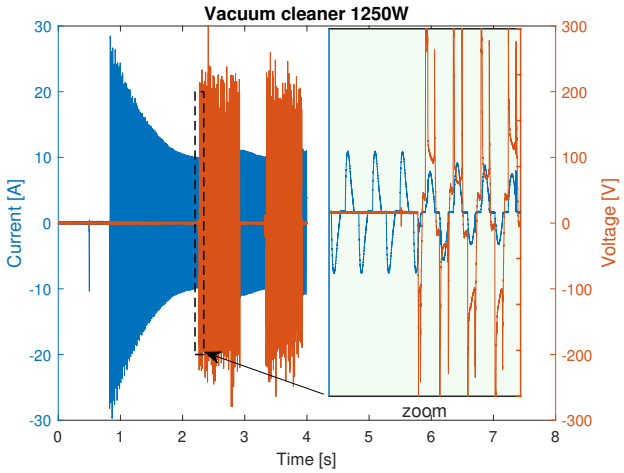

FIGURE 5. Arc voltage (orange line) and current signals (blue line) of vacuum cleaner 1250 W.

IV. EXPERIMENTAL RESULTS

The dataset consists of several time series, which represent the instantaneous current drawn by each load, to be split into training and testing sets. First, we pre-process data by computing the absolute value of the signal: by eliminating the sign, the positive and negative half-waves become indistinguishable. This adaptation led to improved classification performances, as the sign of the half-wave is irrelevant to the training process and contributes to the introduction of bias.

Then, we split each sequence into consecutive non-overlapping windows of length l. These windows were then alternately assigned to the training and testing sets: the first window is allocated to the training set, the second to the testing set, the third to the training set, and so on. The sequences are labeled based on the arc voltage envelope. If the envelope arc voltage exceeds 30 V, the segment is labeled as "ARC"; otherwise, it is labeled as "NOARC". Additionally, segments corresponding to the switch-off state, where the current is below 10 mA, are also labeled as "NOARC". Since the proposed algorithm takes a window of length l as input and returns a single classification value, we define the window as "ARC" if at least one "ARC" sample is present within it. Also note that, if the current drawn is low, arc faults do not represent a threat. Therefore, as in commercial AFDD devices, we do not perform arc FD and the device is considered to be switched off if the mean absolute value of the current is consistently below 10 mA.

The training set is finally randomized using a 20% holdout with stratification. We perform a repeated holdout validation, repeating the partitioning 5 times, and then averaging the results. We recall that, according to Figure 1, the HYDRA algorithm [28] is used to extract features; then, these features are used as input to binary classifiers, namely a set of linear learners with regularization and standardized inputs. We have explored through a grid search the 23040 possible combinations of the following hyperparameters:

- Segment length t: 5 ms, 10 ms, 20 ms, 50 ms.
- Sampling frequency f: 10 kHz, 20 kHz, 50 kHz, 100 kHz.
- Number of kernels per group k: 2, 4, 8, 16.
- Number of groups g: 2, 4, 8.
- Stride s: 1, 2, 4, 8.
- Kernel length m: 5, 9, 17.
- Lasso regularization λ: 10 logarithmically spaced values in $[0.001, 0.1]$.

These parameters influence the number of features generated by HYDRA and employed by the classifiers. Exploring their combinations, we evaluate the trade off between small number of features, that is desirable for deploying the code in MCUs and MPUs, and accuracy. Both the minimum response (in hard counting mode) and the maximum response (in soft counting mode) were used, as already detailed in Section II-A (see Figure 2). First-order differences, which are optional in HYDRA, were not included to reduce computational complexity.

The diagnostic features from HYDRA are employed to face the classification problems detailed in Section II-B, namely, FD, FDI, and DFDI. To extend the binary classifier to multiclass problems (i.e., FDI and DFDI), a one-vs-one approach is adopted, requiring $n = 10$ linear learners for FDI and $n = 91$ linear learners for DFDI. FD, instead, requires a single linear learner ($n = 1$). Training each classifier takes only a fraction of a second, making the training time practically negligible for this application.

A. FAULT DETECTION

Table 2 shows the distribution of the classes and the corresponding labels for FD.

TABLE 2. FD: class distribution and corresponding labels.

Class	%	Label
NOARC - Any appliance	63	0
ARC - Any appliance	37	1

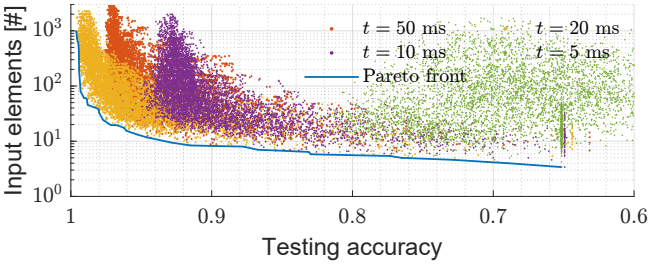

FIGURE 6. FD: objective space (semi-logarithmic scale).

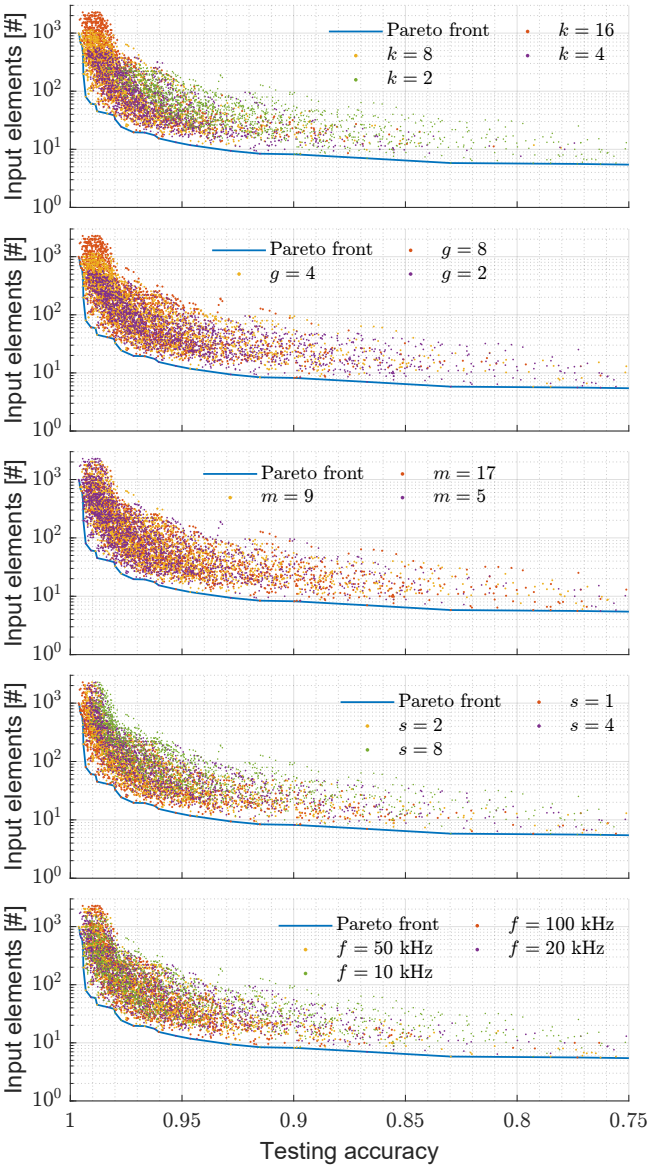

FIGURE 7. FD: objective space (20 ms, semi-logarithmic scale).

Figure 6 shows the average accuracy in the test set and the number of input features for each combination of the hyperparameters. The blue line represents the Pareto front, i.e., the classifiers that are not dominated by any other classifier in terms of lower number of input features and higher accuracy. The color of the dot represents the length of the window. The list of classifiers constituting the Pareto front, sorted by decreasing accuracy, is reported in Table 13 in the Appendix. The maximum accuracy shows to be related to the window length, which represents the most important hyperparameter. With a window length set to 5 ms, 10 ms, 20 ms, and 50 ms, the highest accuracies are 0.8244, 0.9561, 0.9963, 0.9782, and the average (over 5 random partitionings) number of input features is 49, 37, 995, 341, respectively. Please note that some of these results do not belong to the Pareto front. Also note that large values of λ increase sparsity, thereby reducing the number of features and potentially decreasing accuracy.

According to Table 13, a 20 ms window length is the best choice in the accuracy range above 0.8997. A smaller window length is clearly preferrable, as it reduces the detection delay, and $t = 20$ ms returns an acceptable trade-off between accuracy and detection delay.

To minimize the occurrence of false positives and false negatives, which, in turn, entail unnecessary tripping and missed arc FD, let us focus on the case of 20 ms window length as it returns the most accurate results. Figure 7 shows the univariate sensitivity analysis, illustrating the influence of each hyperparameter on accuracy and the number of features required as input. As the number of kernels per group k increases, both the maximum accuracy and the maximum number of input features increase. The Pareto front includes points across the entire tested range of k, however, $k = 2$ leads to a low accuracy and $k = 8$ is often the optimal value. The effect of g is similar; a larger g generally increases the number of input features and, potentially, the accuracy. However, the objective space is less clearly partitioned with respect to g. As for the kernel length m and the sampling frequency f, no separation in the objective space is evident. Remarkably, in the top rows of Table 13, a 50 kHz sampling frequency gives the best result in terms of accuracy. Moreover, a 10 kHz sampling frequency proves to be sufficient for achieving a 0.9888 accuracy. Conversely, the larger the stride s, the worse the performances in general (see Figure 7).

B. FAULT DETECTION AND ISOLATION

Table 3 shows the distribution of the classes and the corresponding labels for FDI.

TABLE 3. FDI: class distribution and corresponding labels.

Class	%	Label
NOARC - Any appliance	63	0
ARC - Universal motor	22	1
ARC - Converter: switching power supply	2	2
ARC - Resistive	10	3
ARC - Converter: single-phase rectifiers	3	4

Figure 8 shows the average accuracy in testing and the number of input features for each combination of the hyperparameters. The list of classifiers constituting the Pareto front, sorted by decreasing accuracy, is reported in Table 14 in the Appendix. Notably, the maximum accuracy (0.9952) is comparable to FD (0.9963). However, the number of input

features is approximately twice as large at the same level of accuracy. In fact, FDI is a multiclass classification problem, which requires 10 linear learners, therefore more features are necessary. This is reasonable, because the FDI task is more challenging, also requiring to define the load class of the faulty load. Note that $t = 20$ ms dominates in the accuracy range above 0.9112. The accuracy in the Pareto front positively correlates with k and g. The parameter values $t = 50$ ms and every $s > 2$ are absent in the Pareto front.

The maximum accuracy is clearly obtained with a 20 ms window. Therefore, we detail the results using a 20 ms window length for brevity. Figure 9 shows the influence of each hyperparameter on the accuracy and the number features required in input. As k increases, both the maximum accuracy and the number of input features increase as well. The optimal g is generally 2, including a wide part of the Pareto front except for the first couple of rows in Table 14. Again, m does not show any evident effect, while large values for s are clearly suboptimal.

C. DETAILED FAULT DETECTION AND ISOLATION

Table 4 shows the distribution of the classes and the corresponding labels for DFDI. Figure 10 shows the average

TABLE 4. DFDI: class distribution and corresponding labels.

Class	%	Label
NOARC - Any appliance	63	0
ARC - Electric Pump	2	1
ARC - Dell Desktop Computer	2	2
ARC - Dolce Gusto Espresso	5	3
ARC - Titan Drill	2	4
ARC - Peugeot Drill	3	5
ARC - Mewal Halogen Lamp	3	6
ARC - Blow Heater	3	7
ARC - Severin 2200 Kettle	2	8
ARC - Moulinex Mini Food Processor	2	9
ARC - Multi-functional Printer	4	10
ARC - Sabre Electric Jigsaw	2	11
ARC - Bluesky Optimo Vacuum Cleaner	5	12
ARC - Philips Vacuum Cleaner	2	13

accuracy in testing for each combination of the hyperparameters. The list of classifiers constituting the Pareto front is reported in Table 15 in the Appendix. The maximum accuracy (0.9885) is marginally lower than FD (0.9963) and FDI (0.9952), and the total number of input features for DFDI is generally larger. Again, we focus on the 20 ms window length for brevity, as it yields the highest accuracy, outperforming the remaining window length values.

Figure 11 shows the influence of each hyperparameter on the accuracy and the number features required in input. The maximum accuracy and the number of input features increase as k increases. The Pareto front includes points across the entire tested range of k, while the optimal value for g is 2, except for the two topmost rows of Table 15.

V. DISCUSSION

In this section, we present the results from the complete time series, which represent the current drawn by the loads in both faulty and faultless conditions. At this stage, the transients

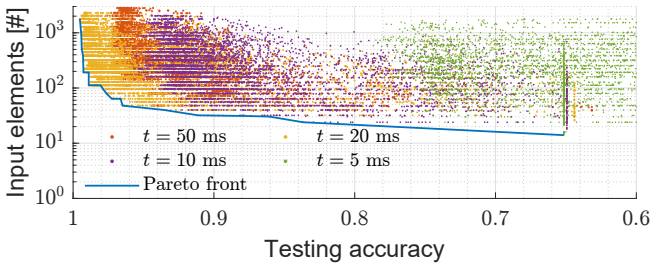

FIGURE 8. FDI: objective space (semi-logarithmic scale).

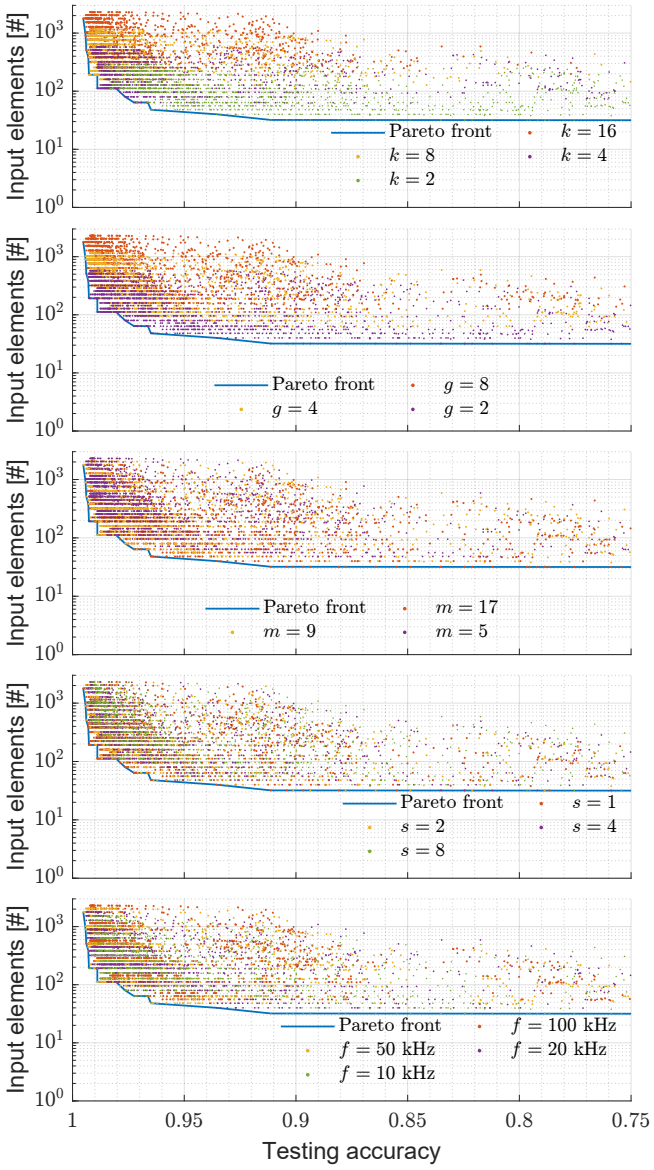

FIGURE 9. FDI: objective space (20 ms, semi-logarithmic scale).

between arcing and non-arcing, as well as between different operating conditions, are kept.

A. TIME SERIES ANALYSIS

Figure 12 and Figure 13 illustrate arc FD for various appliances, including a Dell desktop computer, a Dolce Gusto

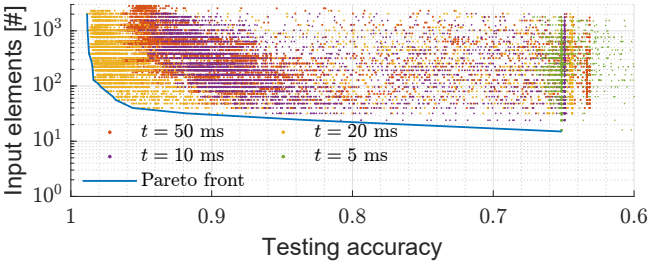

FIGURE 10. DFDI: objective space (semi-logarithmic scale).

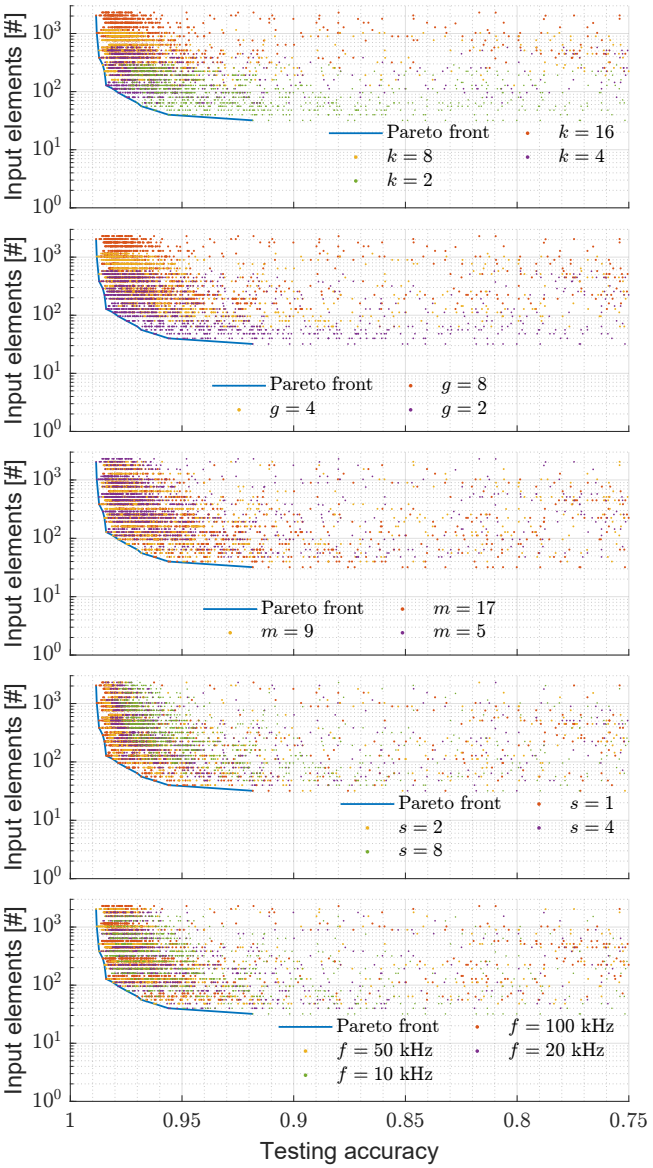

FIGURE 11. DFDI: objective space (20 ms, semi-logarithmic scale).

espresso machine, a Mewal halogen lamp, and a Philips 1250 W vacuum cleaner. The blue line represents the current in case of non-arcing device ("NOARC" label), while the red line represents the current in case of an arc fault ("ARC" label). The outcome of the HYDRA based classification is reported with a green background, when the classification is accurate, and a red background otherwise. Then, a red background together with a blue current line highlights a FP, i.e., the test incorrectly indicates a fault or issue when there is none. A red background together with a red current line, instead, highlights a FN, i.e., the test fails to detect an actual arc fault. Analyzing misclassification in detail, we note that it is caused by changes in the operating states of the appliances: transitioning from off to on, from normal mode to arcing mode, or from arcing mode back to normal mode.

A single FN occurs at the onset of the arc fault in both the halogen lamp and the vacuum cleaner (thin red rectangle in Figure 12). This behaviour occurs because the algorithm operates in a windowed mode. Therefore, if the data segment consists of data from a faultless device in the first part and from a faulty device in the second part, the classifier's response is not well defined. However, a single FN does not represent a critical issue. Practically, as the FN is just a single one, it means that the arc fault is detected in the following data segment (i.e., after additional 20 ms). Therefore, the HYDRA algorithm demonstrates its reliability even in the presence of operational mode changes.

The HYDRA algorithm exhibits the worst classification performance with the desktop computer (top part of Figure 13), where multiple FNs are present. However, it is important to note that, in this case, the current is low, a condition in which commercial AFDDs do not trigger as it does not represent an actual threat.

The Dolce Gusto espresso machine (bottom part of Figure 13) exhibits multiple operating states, including off, standby, heating, and arcing, yet the algorithm effectively classify well these transitions.

Single FNs are triggered when the arc fault is injected, as in the case of Figure 12, and the same considerations hold. It is important to highlight that most FN events are concentrated in the desktop computer appliance. Most of the literature does not include this type of analysis and often reports classification accuracy under steady-state conditions. However, we highlight that transients can lead to FP or FN events in real applications. In this context, reference [14] presents a similar analysis, making the results more solid and reliable for real-world applications.

B. MISCLASSIFICATION ANALYSIS

In the following, we present an analysis of consecutive misclassifications (FPs and FNs) for a selection of hyperparameters that ensure online feasibility using affordable MCUs/MPUs. Their feasibility, in terms of computation time and memory usage, is discussed in Section VI. If present, the initial phase where the device is switched off (e.g., mean absolute value of the current consistently below 10 mA) is not considered, while transients are kept. Table 5 reports the duration of the largest segment of data consistently leading to misclassification. Table 5 shows that no FPs are present in the entire testing set, with any of the proposed sets of HYDRA parameters. This is crucial in practical applications, where

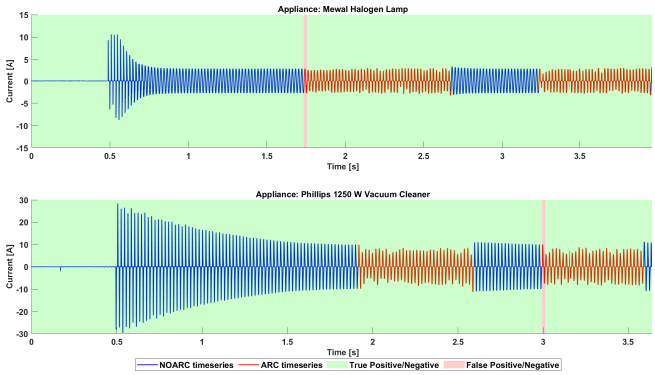

FIGURE 12. Arc FD on Mewal Halogen Lamp and Philips 1250 W Vacuum Cleaner.

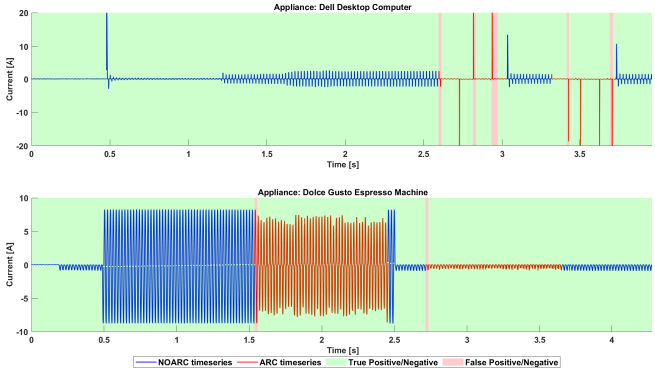

FIGURE 13. Arc FD on Dell Desktop Computer and Dolce Gusto Expresso Machine.

any FP represents an undesired tripping in AFDDs, reducing their usability and user acceptability.

Also, Table 5 shows that the maximum sequence length of consecutive FNs is between 20 ms and 60 ms (1 and 3 consecutive faulty data segments returning a FN, respectively). This estimates the worst case delay in detecting an arc fault due to a missed FD. This delay is taken into account in the following Section to discuss the compliance with international standards for AFDDs.

TABLE 5. Analysis of misclassification (FPs and FNs) in consecutive windows.

						Max error length	
t [ms]	f [kHz]	k	g	m	s	FP [ms]	FN [ms]
20	10	8	4	9	1	0	20
20	10	8	2	5	1	0	40
20	10	8	4	17	1	0	60
20	20	8	8	5	1	0	20
20	20	8	4	5	1	0	60
20	50	8	4	5	2	0	20
20	20	8	4	9	1	0	20
20	50	8	4	5	1	0	20
20	20	8	8	9	1	0	60
20	50	8	2	17	1	0	40
20	50	8	4	9	1	0	40

VI. REAL-TIME FEASIBILITY

The algorithm is tested on a series of STMicroelectronics boards through the ST Edge AI Developer Cloud service, which provides tools for model creation, optimization, and benchmarking. It leverages ST Edge AI Core technology for compilation on various STMicroelectronics devices and supports hardware AI acceleration on compatible devices. The main advantage of the ST Edge AI Developer Cloud service is the ability to compare the algorithm's performance on various devices with actual implementation on a remote board, without the need for prior purchase.

The model is first uploaded in a standard format (ONNX) and the target MCU or MPU is selected. The default single precision is employed. Then the code is optimized to balance latency and memory usage. The C code is generated by using the CMSIS-NN library [32]. Next, a benchmark is executed to evaluate the performance of the optimized model, focusing on speed and memory usage. The results are analyzed to compare devices and identify the most suitable platform. Finally, once the testing and analysis are complete, C code is generated for integration on the selected device.

A. MCU/MPU SELECTION: MEMORY AND INFERENCE TIME ACROSS HYPERPARAMETERS

To show the feasibility of the proposed method in real-time, we have tested the method on three different MCUs and three different MPUs, using the same hyperparameters shown in Table 5. We focus on ARM-based MCUs as they provide an optimal balance of low cost, low power consumption, and ease of development compared to alternatives such as DSPs, FPGAs, and single-board computers. As will become clear in the following, their computational capabilities are sufficient to effectively implement the proposed AFDD algorithm, making them ideal for cost effective applications. Table 6 resumes the memory usage and the time to extract features using HYDRA together with subsequent linear classification. The corresponding choice of the hyperparamers and the number of features actually used by the classifier are reported as well.

We note that both the inference time and the memory usage generally increase as the number of output features increase. Figure 14 shows the fitting curve from inference time in ms and the RAM occupancy in kB. The curve is approximated as a cubic function with an R^2 value of 0.92.

Figure 14 relates memory occupancy and inference time: increasing the number of kernels, both the number of weights and multiplications grow. Empirically, to keep the inference time below 50 ms, RAM usage should be below 300 kB.

According to Table 6, the inference time on an MPU architecture is lower than on an MCU architecture, making the implementation on an MPU more scalable when RAM usage increases. We remark that the inference time of the linear classifier alone is negligible (approximately 0.1 ms when deployed on a Cortex-M7 MCU) with respect to the HYDRA inference time for feature generation.

TABLE 6. Execution time and memory usage of HYDRA feature extraction and classification on different Arm Cortex MCUs/MPUs.

							Inference time [ms]						Memory [kB]	
t [ms]	f [kHz]	k	g	m	s	Features [#]	M7 600 MHz	M7 550 MHz	M7 480 MHz	A35+M33 1.5 GHz (2 cores)	A7 1 GHz (1 core)	A7 800 MHz (1 core)	RAM usage	FLASH usage
20	10	8	2	5	1	64	6.6	7.1	8.2	2.8	6.7	6.9	63.7	57.6
20	10	8	4	9	1	59	6.7	7.3	8.3	3.1	7.6	7.8	86.3	61.9
20	10	8	4	17	1	72	11.6	12.4	14.2	3.2	7.7	7.7	67.5	57.2
20	10	8	2	5	1	160	12.9	14.1	16.2	3.3	7.9	8.4	78.6	58.3
20	10	8	4	17	1	192	19.3	20.6	23.5	3.8	9.1	9.3	87.8	60.5
20	20	8	4	5	1	192	28.3	31.3	36.4	9.1	21.3	23.3	194.0	77.1
20	20	8	8	5	1	112	28.5	32.0	36.8	8.2	22.6	23.9	367.0	103.2
20	20	8	4	5	1	212	41.4	45.3	52.7	9.8	23.1	25.7	202.6	86.1
20	50	8	4	5	2	197	50.2	55.4	63.8	13.8	32.0	35.2	298.3	94.8
20	20	8	4	9	1	384	52.6	57.2	66.1	10.5	24.6	25.9	224.3	79.4
20	50	8	4	5	1	80	52.8	57.4	66.0	11.9	37.1	38.0	475.8	83.9
20	20	8	8	9	1	643	85.2	96.9	107.7	19.2	44.0	49.3	437.9	118.2
20	50	8	2	17	1	188	91.0	99.1	114.6	14.2	37.1	37.2	284.0	67.8
20	50	8	4	9	1	384	138.6	149.6	172.0	25.8	64.0	67.0	538.3	85.5

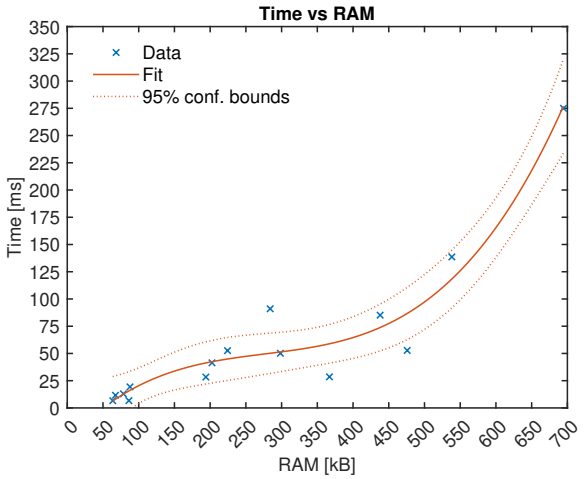

FIGURE 14. Fitting of Execution time versus memory usage.

Please note that the choice of the hyperparameters is crucial for performance. Increasing f and/or t can improve accuracy but dramatically increases memory usage and inference time, as they increase $l = f \cdot t$ (and, in turn, d); therefore, l should be limited to the smallest acceptable value. Increasing k and/or g can also improve accuracy, but at the cost of increased memory usage and inference time. Moreover, there is a threshold beyond which further increasing k and/or g provides negligible benefits, as shown in Section IV. Increasing m increases the number of weights and, consequently, the computational load of the 1D convolution. However, it also reduces the dilation d, which helps limit the overall inference time. Regarding m, there is no clear strategy to improve accuracy: short kernels are problematic because they are more likely to be highly correlated, whereas long random kernels are less likely to align with patterns in the input, effectively acting as random noise. Experimentally, when increasing m on MCUs/MPUs, the HYDRA extraction time increases due to the additional computational load of the 1D convolution (as more weights and multiplications are required for a longer kernel), despite the reduction of d. Increasing s can slightly reduce inference time due to a faster convolution but also reduces accuracy, so its optimal value is typically 1. Finally, increasing λ can reduce memory usage and inference time due to increased sparsity but may also reduce accuracy.

B. COMPLIANCE OF TRIPPING TIME WITH IEC STANDARDS

To determine the compliance with international standards, we refer to the IEC 62606 standard. It specifies a maximum *tripping time*, namely, the maximum time that may elapse between the onset of the arc fault and the load disconnection. According to the IEC 62606 standard, the maximum tripping times for domestic AFDDs range from 0.12 s (at 63 A_{rms}) to 1 s (at 2.5 A_{rms}). The tripping time is composed of the time to:

1) acquire and buffer the data (*buffering time*),
2) extract the features,
3) make the decision on the presence of the fault,
4) repeat the previous steps in case of false negatives,
5) interrupt the electrical circuit (*disconnection time*).

We define *inference time* as the time taken to elaborate data, i.e, for steps 2 to 3. The first step (data buffering) takes 20 ms, therefore prolonging the tripping time by up to 20 ms in the worst case (i.e., when an arc fault occurs at the beginning of the window). The time to perform the second step (feature extraction) is the most significant. It ranges from 2.8 ms to 172.0 ms, depending on the HYDRA hyperparameters and the available MCU/MPU, as reported in Table 6. The time to perform the third step (classification) is in the order of 0.1 ms, therefore it is negligible. The time lost due to false negatives (step 4) coincides, in the worst case, with the maximum error length reported in Table 5, which should be taken conservatively into account. Step 5 (disconnection time) depends on the delatching mechanism, however, it can be conservatively overestimated at 20 ms.

For each combination of the HYDRA hyperparameters, Table 7 defines the minimal MCU/MPU requirements to

TABLE 7. Analysis of MCU/MPU configurations ensuring compliance with IEC 62606 safety constraints.

t [ms]	f [kHz]	k	g	m	s	Max trip time [ms]	Feasible MCU/MPU
20	10	8	4	9	1	68.4	M7 480 MHz
20	10	8	2	5	1	96.3	M7 480 MHz
20	10	8	4	17	1	119.4	M7 600 MHz
20	20	8	8	5	1	96.9	M7 480 MHz
20	20	8	4	5	1	109.9	A35+M33
20	50	8	4	5	2	115.5	M7 550 MHz
20	20	8	4	9	1	117.3	M7 550 MHz
20	50	8	4	5	1	117.5	M7 550 MHz
20	20	8	8	9	1	119.3	A35+M33
20	50	8	2	17	1	117.3	A7 800 MHz
20	50	8	4	9	1	105.9	A35+M33

comply with the tripping time prescribed by the IEC 62606 standard in the most challenging scenario (0.12 s at 63 A_{rms}). The tripping time is conservatively calculated as the sum of the data buffering worst case delay (20 ms), the feature extraction time using the chosen MCU/MPU (see Table 6), the decision time (0.1 ms), the delay due to false negatives in the worst case (see Table 5), and the disconnection time (20 ms).

The Arm Cortex-M7 480 MHz, which exhibits the lowest processing performance in the group, provides sufficient computational power to implement lightweight arc FD strategies (top rows of Table 7). Their affordability enables potential integration into domestic applications as well as individual household appliances, representing a shift in approach compared to traditional AFDDs, which are typically installed at the electrical panel level.

VII. ABLATION STUDY

The proposed method for arc FD, FDI, and DFDI is based on a two step process (see Figure 1), namely, feature extraction using HYDRA and classification using linear SVMs. The objective of this section is to analyze the contribution of individual components and justify their choice with respect to possible alternatives. Each test is performed under the same conditions detailed in Section IV.

A. ABLATING HYDRA: RAW CURRENT DATA

The performances using the same classifiers without prior feature extraction, i.e, performing direct classification of the raw data, returns unsatisfactory results. Figure 15 shows that the accuracy reaches, in the best case, 74.36% in FD, 83.60% in FDI, and 88.20% in DFDI.

The reported results are drastically worse than HYDRA, that shows up to 99.63% in detection (Table 13), 99.52% in isolation (Table 14), and 98.85% in detailed isolation (Table 15). The performances are worse because linear kernel SVMs are fast and efficient but require linearly separable data. Therefore, the SVM benefits from high dimensional data that is linearly separable, making it suitable with the HYDRA algorithm for feature generation.

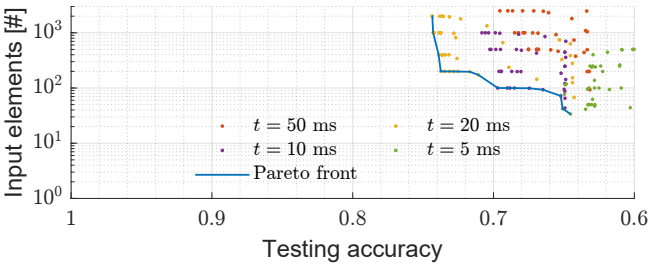

FIGURE 15. FD using raw current data: objective space.

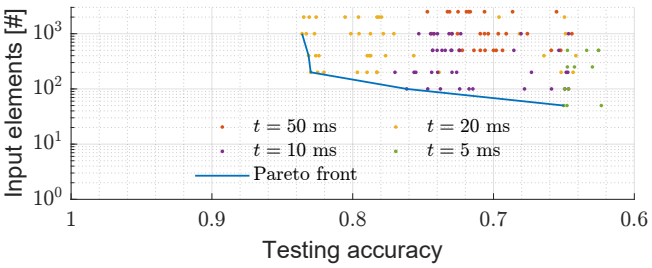

FIGURE 16. FDI using raw current data: objective space.

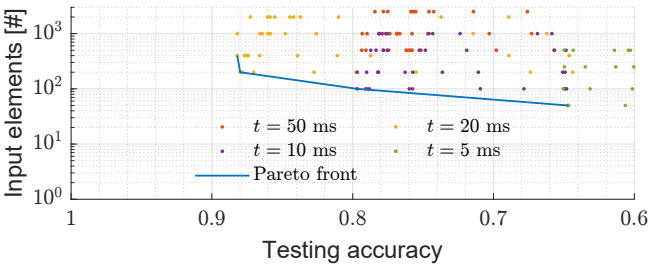

FIGURE 17. DFDI using raw current data: objective space.

B. ABLATING HYDRA: DIAGNOSTIC FEATURE DESIGNER

We compare the features generated by HYDRA with those obtained on the same dataset using the Diagnostic Feature Designer (DFD) in MATLAB® 2024b. The DFD is a graphical user interface tool designed to assist with the automatic feature extraction of time series data for detection and diagnostic purposes. The DFD-based detection approach primarily relies on the feature extraction step provided by the DFD tool. A total of 66 features were computed, encompassing both time-domain and frequency-domain characteristics. Specifically, we extracted 56 time-domain features and 10 frequency-domain features. These features were calculated for both the raw signals and their first-order differences, resulting in a total of 132 features. Given the relatively low number of features, a SVM classifier with linear kernel is used for binary classification. The hyperparameters are optimized using classical Bayesian optimization with a 5-fold cross-validation. The results are summarized in Table 8, which reports the FD accuracy. Table 8 presents results for two scenarios: one considering only the 66 features (without the first-order differences) and another incorporating all 132 features.

The accuracy reported in Table 8 is poor if compared to

TABLE 8. FD: Diagnostic Feature Designer

t [ms]	f [kHz]	DFD detection accuracy	
		66 features	132 features
20	20	94.4%	97.2%
20	10	90.2%	94.2%

Pareto optimal choices for the HYDRA method (Table 13). Therefore, it does not justify using the DFD features instead of HYDRA features to increase the accuracy. Moreover, calculating the DFD features is computationally demanding. Setting $t = 20$ ms and $f = 20$ kHz, the feature extraction process requires 0.09 s for 132 features and 0.045 s for 66 features on a laptop equipped with an Intel 7700HQ CPU, 16GB RAM, which has a computational power far superior to the previously tested MCUs and MPUs. These results together reveal that a DFD based approach is not reliable nor suitable for the implementation in MCUs and MPUs.

C. ABLATING SVM: ALTERNATIVE CLASSIFIERS

To motivate the choice of the SVM, we propose a thorough comparison between linear SVMs and every classifier proposed by the Classification Learner app in MATLAB® 2024b.

Table 9 reports the results for the FD case, where the linear SVM is replaced by alternative classifiers. To limit computation time and the amount of results, we have limited the analysis to the best hyperparameters for HYDRA in FD, namely, $t = 20$ ms, $f = 50$ kHz, $k = 8$, $g = 8$, $m = 5$, and $s = 1$ (see Table 13). The results of the best classifiers are comparable to those in Table 13. Note that the occupied space is only an upper-bound estimate, as it represents the size in MATLAB and not the optimized size for deployment on a MCU. Similarly, the prediction speed refers to a high-end PC, and the results on a MCU may differ. The Efficient Linear SVM, proposed by the Classification Learner, provides a high prediction speed and a very low model size, which is consistent with the findings on MCUs and MPUs reported in Section VI.

Repeating the same analysis for the FDI and the DFDI gives similar results and are omitted for brevity. As the number of classes increases, the space efficiency of the SVM is challenged by NNs, which provide similar results and less memory occupancy. Therefore, in the case of DFDI, NNs represent a promising alternative in case of space limitations.

TABLE 9. Alternatives to SVMs in FD, sorted by accuracy.

Classifier	Accuracy [%]	Prediction Speed [obs/sec]	Model Size
Fine KNN	99.700	1994.3	4.54 MB
Subspace KNN	99.700	496.1	68.4 MB
Medium NN	99.700	3478.3	260.0 kB
Cubic SVM	99.625	2445.1	883.0 kB
Quadratic SVM	99.550	2460.7	796.0 kB
SVM Kernel	99.475	1923.8	291.0 kB
Efficient Linear SVM	99.400	3934.2	167.0 kB
Medium Gaussian SVM	99.400	2771.1	1.1 MB
Bilayered NN	99.400	2144.4	200.0 kB
Narrow NN	99.250	3113.6	198.0 kB
Wide NN	99.250	2897.6	568.0 kB
Trilayered NN	99.250	2823.0	201.0 kB
Linear SVM	99.175	2164.2	907.0 kB
Subspace Discriminant	99.100	753.7	66.0 MB
Linear Discriminant	98.876	3886.5	8.6 MB
Weighted KNN	98.801	1529.1	4.5 MB
RUSBoosted Trees	98.726	2299.6	1.8 MB
Logistic Regression Kernel	98.726	1546.0	291.0 kB
Cosine KNN	98.651	1164.9	4.5 MB
Bagged Trees	98.501	1868.6	4.7 MB
Medium KNN	98.426	1951.8	4.5 MB
Cubic KNN	98.051	332.8	4.5 MB
Efficient Logistic Regr.	97.976	4147.4	163.0 kB
Fine Gaussian SVM	97.976	2474.4	3.8 MB
Fine Tree	97.901	4824.3	150.0 kB
Medium Tree	97.901	3873.4	150.0 kB
Coarse Gaussian SVM	96.702	3064.7	2.2 MB
Coarse Tree	95.127	3512.6	148.0 kB
Binary GLM Logistic Regr.	92.129	1836.3	22.4 MB
Kernel Naive Bayes	89.355	61.5	21.5 MB
Coarse KNN	85.682	2010.4	4.5 MB
Gaussian Naive Bayes	79.760	906.5	553.0 kB
Boosted Trees	64.543	3659.0	147.0 kB

VIII. COMPARISONS

In this section, the results for arc FD are compared with various methods from the literature and summarized in Table 10. This comparison is based on the results reported in recent works, therefore the dataset and the operating conditions may vary. Specifically, since the presence of load transients in different datasets is not guaranteed, the accuracy in a real environment may be lower. Also, we note that only part of the works address both arc FD and load classification simultaneously. In [11] and [8], frequency analysis provides input data for classification via a NN. In [11], high classification accuracy is achieved, but with a limited number of categories. The analysis requires both current and voltage measurements. Moreover, note that the algorithm is executed on a PC; therefore, the execution time on MCUs and MPUs is expected to be significantly higher. In [8], a genetic algorithm is required to optimize the network's initial parameters. The accuracy and the execution time are remarkable, however, this approach has been tested on a limited range of appliances. The method proposed in [9], which is tested on a wider variety of appliances, is based on current analysis using an HCF sensor with a relatively high sampling frequency of 1 MHz. The algorithm is tested on an FPGA, which can significantly accelerate code execution. Consequently, the inference time cannot be directly compared to that of MPUs and MCUs. Both the recent works [15] and [16] rely on NN to detect arc faults, testing it on a variety of loads. In [15], the detection performance is slightly lower than the other works and no classification is performed. However, the algorithm runs on a low-end MCU, so the inference time could be further reduced. In [16], knowledge distillation significantly reduces the inference time, achieving excellent inference time while maintaining high accuracy in detection (99.31%) and classification (98.85%). The author's final validation is performed on the same dataset used in this work [31], which includes challenging transient behaviors: in this case, the FD accuracy drops to 97.88%. In the proposed work, we achieve improved detection performances on the same dataset and we provide classification results. The inference time on a

TABLE 10. Comparison with prior methods

Reference	Method	Category	Loads	Sensor	Platform	Inference Time	Accuracy
Ref. [11]	Wavelet transform (db3); $f = 200$ kHz; Category recognition: shoulder band phase analysis; Arcing: time and frequency indicators + BP NN	Resistive, Resistive-Inductive, Rectifying circuit with a capacitive filter	Resistance, Fluorescent lamp, Halogen lamp, Drill, Air conditioner, Computer, Tungsten lamp	I/V	PC	10 ms	Detection≈99%; Classification 100% (3 classes); 630 time series
Ref. [8]	Frequency analysis (1st-3rd-5th harmonics) for category recognition and time domain features + fully connected NN	Resistive, Capacitive-inductive, Switching	Resistive load, Inductive load, Capacitive load, Switching load	I	Arm Cortex-M4 168MHz	3 ms ($f = 20$ kHz)	Detection≈99% (4 classes); 3950 time series
Ref. [9]	HCF sensor; $f = 1$ MHz; Gray image analysis + convolution NN	7 load classes (no fault) associated to 7 classes (arc fault)	Electric heater, Vacuum cleaner, Computer, Dimmer Fluorescent lamp, Induction cooker, Drill	I	Zynq-7020 (FPGA)	11 ms	Detection≈99%; Classification≈98% (14 classes); 12600 time series
Ref. [15]	Adam-optimized BP NN	Resistive, Motor, Switching Load, Rectifier load	Vacuum cleaner, Fluorescent lamp, Electronic lamp dimmer, Air compressor, Switching power supply, Electric hand-held drill	I	Arm Cortex-M4 84MHz	45 ms ($f = 12.8$ kHz)	Detection≈98%; 300 time series
Ref. [16]	Teacher-student knowledge distillation approach with a CNN architecture	Resistive, Motor, Switching Load, Gas discharge lamp	Electrical heater, electric iron, incandescent lamps, electric kettle, Capacitor start motor, vacuum cleaner, electric hand tool, Switch-mode power supply loads, dimmer, Halogen lamps, fluorescent lamps	I	Arm Cortex-M7 480MHz	3 ms ($f = 10$ kHz)	Detection≈99%; Classification≈99% (8 classes); 4500 time series
Ref. [10]	Recurrence quantification and Image analysis (gray level co-occurrence matrix)	Resistive, Motor, Switching Load, Rectifier load	Drill, Expresso machine, Halogen Lamp, Blow heater, Vacuum cleaner, Electric jigsaw, Food processor, Kettle, Printer	I	PC	50 ms ($f = 4$ kHz)	Detection≈99%; Classification≈98% (24 classes); 5000 time series
Proposed	HYDRA	Resistive, Motor, Switching Load, Rectifier load	Drill, Expresso machine, Halogen Lamp, Blow heater, Vacuum cleaner, Electric jigsaw, Food processor, Kettle, Printer	I	Arm Cortex M7 600MHz	20 − 50 ms ($f = 10 - 20$ kHz)	Detection≈99%; Classification≈99% (14 classes); 1500 time series

comparable MCU is higher with respect to [16], but still feasible to be implemented in real-time while respecting the prescribed maximum tripping time. The results of the proposed approach outperform those from the previous work [10].

A. COMPARISON WITH STATE-OF-THE-ART METHODS

Most of the works do not share the code nor the dataset to compare the results. In the following, we compare the results with two recent state-of-the-art algorithms [17], [20], which have been implemented from scratch to perform this test. The authors of [17] simply propose to compare three features with a fixed threshold. If any of the features overcome the threshold, an arc fault is detected. According to [17], the sampling is set to $f = 250$ kHz and a $t = 200$ ms window starting from a zero crossing is used. As detailed in [17], we have extracted three features for each data segment: Total Harmonic Distortion (THD), sum of harmonic components, and average rate of change of the currents. However, the approach in [17] fails in the dataset employed in this work. First of all, we note that the presence of an arc fault does not entail an increase of the selected features. For example, in presence of an arc fault using the Dell Desktop Computer and Philips Vacuum Cleaner, the three features decrease. Therefore, the founding principles of [17] do not apply for every kind of load. Also, fixed thresholds do not suit the complexity of the problem. For example, in [17], the THD threshold is set to 0.85. However, only one load (Sabre electric Jigsaw) overcomes this threshold in case of a fault. Using the predefined thresholds suggested in [17], the outcome is a 100% false positive rate. Finally, we remark that the authors of [17] require collecting current measurements for 200 ms. This makes the method inapplicable according to the IEC 62606 standard, where the maximum tripping time is 120 ms in the most challenging case.

TABLE 11. Classification accuracy using the algorithm proposed in [20].

Random Partitioning [#]	Accuracy (test set) FD [%]	FDI [%]	DFDI [%]
1	98.66	97.89	91.76
2	97.70	97.89	90.61
3	98.47	97.70	91.76
4	98.28	97.70	90.42
5	98.08	97.51	90.61
Avg.	98.24	97.74	91.03

The method proposed in [20] is based on the extraction of 6 time- and frequency-based features (kurtosis, crest factor, shape factor, clearance factor, L2/L1 norm, spectral centroid), followed by a random forest classifier. The sampling frequency $f = 100$ kHz and window length $t = 20$ ms are chosen according to [20]. Training and testing are performed under the same conditions detailed for the proposed method:

we exclude low current data, randomize the training set using a 20% holdout with stratification, and perform a repeated holdout validation, repeating the partitioning 5 times. Table 11 shows the accuracy of algorithm [20] on the same dataset used for the proposed method. Let us focus on the test set to fairly compare the results. The algorithm [20] shows lower accuracy in FD (98.24% vs. up to 99.63%, respectively, see Table 12 in the paper) and FDI (97.74% vs. up to 99.52%, respectively, see Table 13 in the paper). Regarding DFDI, algorithm [20] shows drastically lower accuracy (91.03% vs. up to 98.85%, respectively, see Table 14 in the paper). In summary, despite using only 6 features, the method proposed in [20] returns relatively high accuracy; however, it is still insufficient to make it applicable in practice.

B. COMPARISON USING A MULTIPLE-LOAD DATASET

The proposed method has been previously tested on a dataset comprising single loads. Among the various tests mandated by the IEC 62606 standard, procedures with inhibition loads are required to verify arc detection capability, including tests involving multiple loads arranged in specified configurations. To validate the algorithm in the presence of multiple loads, we have also considered the dataset [33] to compare the proposed algorithm with the state-of-the-art algorithm presented in [20].

TABLE 12. FD (multiple loads): average accuracy on the test set [33].

Random partitioning [#]	Algorithm Accuracy	
	Proposed [%]	Algorithm [20] [%]
1	98.21	98.52
2	97.78	98.21
3	98.08	98.30
4	98.91	98.34
5	99.04	98.08
Avg.	98.41	98.29

We employed a single HYDRA ($k = 8$, $g = 8$, $m = 9$, $s = 1$) and a linear classifier to compare the results with [20]. The same sampling frequency $f = 100\,\text{kHz}$ and window length $t = 20\,\text{ms}$ are chosen according to [20]. For a fair comparison, both methods are tested on the same single- and multiple-load data from [33]. The training set is randomized using a 20% holdout with stratification. We perform a repeated holdout validation, repeating the partitioning 5 times, and then averaging the results. Table 12 shows that the proposed method achieves a slightly higher performance in arc FD even in the case of multiple loads: the average accuracy in the entire data sequence increases from 98.29% in [20] to 98.41% using the proposed algorithm.

IX. CONCLUSION

This paper presents a method for detecting and classifying series arc faults in domestic AC electrical circuits from current time series data, based on fast dictionary method for time series classification employing competing convolutional kernels. The results demonstrate that detection accuracy can reaches 99%, while appliance detailed classification achieves around 98%, with inference times ranging from 2.8 ms to 172.0 ms on the ARM Cortex-M7 architecture. Also, we have shown that misclassification usually occurs during transients, and the number of consecutive misclassifications is small.

The main limitation of the proposed approach consists in the high RAM usage of HYDRA, which increases with the number of kernels k, groups g, and the length l of the time series. This can negatively impact inference time, as shown in Figure 14, and the choice of the hyperparameters is critical. Essentially, one should start by finding the minimal t to describe an arc fault, and then choose the minimal f to capture its behavior. This allows for a minimal sequence length l, which is superlinearly correlated with both space and time complexity. Moreover, as t increases, more time is needed to buffer data, further increasing the delay between the occurrence of an arc fault and its detection. Then, g and k should be chosen as a trade-off between accuracy, memory usage, and inference time. The optimal value for s is usually 1, so there is little interest in investigating it. On the contrary, finding the optimal kernel length m is not straightforward; however, the value $m = 9$ represents a good starting point. The issue of limited computational power and memory can be addressed in future works by implementing quantization for the 1D convolution stage or by using half-precision in the algorithm.

From a practical standpoint, AFDDs are installed to protect final circuits, where multiple appliances may be connected. Although the results obtained on the multiple-load dataset [33] are satisfactory, further validation of the proposed method is necessary, particularly to meet the requirements of the IEC 62606 standard. To this end, we are conducting an extensive investigation into the effectiveness of the proposed approach across a broader range of multiple-load scenarios. In our future work, we are integrating new devices into the dataset, particularly switching power supplies, as they represent one of the most challenging cases. This also enables leave-one-appliance-out cross-validation to simulate the behavior of previously unseen appliances, providing insights into the model's generalization capabilities. The current findings also show that arc fault protection could be embedded either within individual appliances or at the level of electrical outlets, as the proposed method performs well in single-appliance scenarios and can be implemented at minimal cost using a microcontroller. This latter approach represents a significant departure from conventional practice, where AFDDs are generally installed at the electrical panel level. While centralized AFDDs offer the advantage of lower overall cost, we argue that the affordability of the proposed device architecture makes per-appliance installation a feasible and promising alternative.

REFERENCES

[1] F. Ferracuti, R. Felicetti, L. Cavanini, P. Schweitzer, and A. Monteriù, "Real-time series arc fault detection and appliances classification in AC networks based on competing convolutional kernels," Apr. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15279701

TABLE 13. Fault detection: Pareto-optimal parameters.

Accuracy [%]	Features [#]	t [ms]	f [kHz]	k	g	m	s	λ
99.63	995	20	50	8	8	5	1	0.0010
99.60	932	20	50	8	8	5	1	0.0017
99.57	737	20	20	8	8	9	1	0.0017
99.52	643	20	20	8	8	9	1	0.0028
99.48	630	20	20	8	8	17	1	0.0010
99.43	424	20	50	8	4	5	2	0.0028
99.42	197	20	50	8	4	5	2	0.0077
99.31	80	20	50	8	4	5	1	0.0129
99.07	61	20	20	8	8	5	1	0.0359
98.88	59	20	10	8	4	9	1	0.0215
98.80	45	20	50	8	4	5	1	0.0215
98.34	41	20	20	8	8	5	1	0.0599
98.04	38	20	50	16	2	5	2	0.0359
97.99	33	20	10	4	4	9	1	0.0359
97.72	24	20	10	4	4	9	1	0.0599
97.17	20	20	100	16	2	5	1	0.0599
96.67	19	20	50	4	8	5	1	0.1000
96.22	17	20	20	4	2	17	1	0.0359
96.04	15	20	100	16	2	5	1	0.1000
95.28	13	20	20	4	2	17	1	0.0599
94.63	12	20	20	4	4	9	1	0.1000
92.82	9	20	50	4	2	5	1	0.1000
91.51	8	20	10	4	4	17	1	0.1000
89.97	8	20	50	2	8	17	4	0.1000
87.77	8	10	50	4	2	5	1	0.1000
86.72	7	20	20	2	2	17	1	0.1000
85.45	7	50	50	2	2	17	1	0.1000
83.11	6	10	100	4	2	5	1	0.1000
82.97	6	20	50	2	2	17	8	0.1000
77.34	5	10	10	4	2	9	2	0.1000
76.51	5	10	50	2	2	17	2	0.1000
72.81	5	10	100	2	2	17	8	0.1000
70.39	4	10	20	2	2	5	1	0.1000
68.97	4	10	20	2	2	5	2	0.1000
65.15	3	5	10	8	2	17	8	0.1000

TABLE 14. Fault isolation: Pareto-optimal parameters.

Accuracy [%]	Features [#]	t [ms]	f [kHz]	k	g	m	s	λ
99.52	1792	20	50	16	8	9	1	0.0010
99.42	896	20	50	16	4	9	2	0.0046
99.39	512	20	100	16	2	9	2	0.0046
99.30	384	20	50	16	2	17	1	0.0017
99.27	192	20	10	8	2	5	1	0.0017
98.89	191	20	10	8	2	5	1	0.0077
98.89	112	20	50	4	2	9	1	0.0010
98.05	112	20	20	4	2	5	1	0.0077
97.87	96	20	50	4	2	17	1	0.0017
97.65	80	20	10	4	2	9	1	0.0017
97.26	64	20	10	4	2	17	1	0.0017
96.63	64	20	100	2	2	9	1	0.0028
96.49	48	20	50	2	2	17	1	0.0010
93.54	40	20	20	2	2	17	1	0.0010
91.12	32	20	10	2	2	17	1	0.0010
87.38	31	10	10	2	2	9	1	0.0046
86.09	30	10	10	2	2	9	1	0.0077
83.68	24	10	10	2	2	17	1	0.0010
65.15	14	5	10	2	2	17	2	0.1000

TABLE 15. Detailed fault isolation: Pareto-optimal parameters.

Accuracy [%]	Features [#]	t [ms]	f [kHz]	k	g	m	s	λ
98.85	2046	20	50	16	8	5	1	0.0028
98.82	1022	20	50	16	4	5	1	0.0046
98.71	384	20	50	16	2	17	1	0.0129
98.55	288	20	100	8	2	5	1	0.0017
98.47	224	20	50	8	2	9	1	0.0017
98.40	128	20	100	4	2	9	1	0.0017
98.10	112	20	50	4	2	9	1	0.0028
97.86	96	20	20	4	2	9	1	0.0017
97.48	80	20	10	4	2	9	1	0.0010
97.00	64	20	100	2	2	9	1	0.0010
96.82	56	20	50	2	2	9	1	0.0010
96.25	48	20	50	2	2	17	1	0.0028
95.61	40	20	20	2	2	17	1	0.0010
91.83	32	20	10	2	2	17	1	0.0010
83.02	24	10	10	2	2	17	1	0.0010
65.15	15	5	10	2	2	17	8	0.1000

P. Joyeux, "An embedded system for AC series arc detection by inter-period correlations of current," Electric Power Systems Research, vol. 129, pp. 227 – 234, 2015.

[2] M. Rabla, E. Tisserand, P. Schweitzer, and J. Lezama, "Arc Fault Analysis and Localisation by Cross-Correlation in 270 V DC," in 2013 IEEE 59th Holm Conference on Electrical Contacts (Holm 2013), pp. 1–6, 2013.

[3] G. Concas, B. i. Dôme, L. Lhardit, and O. Tissot, "WHITE PAPER–Residential electrical safety How to insure progress," Tech. Rep., 2020. [Online]. Available: https://www.feedsnet.org/

[4] J. Lezama, P. Schweitzer, E. Tisserand, J.-B. Humbert, S. Weber, and

[5] J. J. Shea, "Identifying causes for certain types of electrically initiated fires in residential circuits," Fire and Materials, vol. 35, no. 1, pp. 19–42, 2011.

[6] P. Qi, S. Jovanovic, J. Lezama, and P. Schweitzer, "Discrete wavelet transform optimal parameters estimation for arc fault detection in low-voltage residential power networks," Electric Power Systems Research, vol. 143, pp. 130 – 139, 2017.

[7] F. Z. Y. Wang and S. Zhang, "A New Methodology for Identifying Arc Fault by Sparse Representation and Neural Network," IEEE Transactions on Instrumentation and Measurement, vol. 67, pp. 2526–2537, 2018.

[8] Y. Wang, F. Zhang, X. Zhang, and S. Zhang, "Series AC Arc Fault Detection Method Based on Hybrid Time and Frequency Analysis and Fully Connected Neural Network," IEEE Transactions on Industrial Informatics, vol. 15, no. 12, pp. 6210–6219, 2019.

[9] P. S. R. Chu and R. Zhang, "Series AC Arc Fault Detection Method Based on High-Frequency Coupling Sensor and Convolution Neural Network," Sensors, vol. 20, no. 17, p. 4910, Aug. 2020.

[10] F. Ferracuti, P. Schweitzer, and A. Monteriù, "Arc fault detection and appliances classification in ac home electrical networks using recurrence quantification plots and image analysis," Electric Power Systems Research, vol. 201, p. 107503, 2021.

[11] X. Han, D. Li, L. Huang, H. Huang, J. Yang, Y. Zhang, X. Wu, and Q. Lu, "Series arc fault detection method based on category recognition and artificial neural network," Electronics (Switzerland), vol. 9, no. 9, pp. 1–21, 2020.

[12] Y. Wang, L. Hou, K. C. Paul, Y. Ban, C. Chen, and T. Zhao, "ArcNet: Series AC Arc Fault Detection Based on Raw Current and Convolutional Neural Network," IEEE Transactions on Industrial Informatics, vol. 18, no. 1, pp. 77–86, 2022.

[13] Y. Liu, Z. Lv, S. Zhang, L. Zhang, and F. Guo, "Feature Extraction and Detection Method of Series Arc Faults in a Motor With Inverter Circuits Under Vibration Conditions," IEEE Transactions on Industrial Electronics, vol. 71, no. 6, pp. 6294–6303, 2024.

[14] Y. Zhang, H. C. Chen, Z. Li, C. Jia, Y. Du, K. Zhang, and H. Wei, "Lightweight AC Arc Fault Detection Method by Integration of Event-Based Load Classification," IEEE Transactions on Industrial Electronics, vol. 71, no. 4, pp. 4130–4140, 2024.

[15] W. Chen, Y. Han, J. Zhao, C. Chen, B. Zhang, Z. Wu, and Z. Lin, "Lightweight Arc Fault Detection Method Based on Adam-Optimized Neural Network and Hardware Feature Algorithm," Energies, vol. 17, no. 6, p. 1412, 2024.

[16] K. C. Paul, C. Chen, Y. Wang, and T. Zhao, "Larcnet: Lightweight Neural Network for Real-Time Series Ac Arc Fault Detection," IEEE Open Journal of Industry Applications, pp. 1–14, 2024.

[17] G. Reji, K. Selvajyothi, and B. Raja, "A robust online series arc fault detection for single-phase systems," IEEE Transactions on Instrumentation and Measurement, 2025.

[18] M. Tan, Y. Gong, L. Wang, K. Li, J. Tong, and Y. Su, "Lfarc-pfe: A series arc fault detection method based on low-frequency current data and

perturbation feature extraction," IEEE Transactions on Instrumentation and Measurement, 2025.

[19] J. Duan, L. Zeng, Y. Wu, L. Chen, and C. P. Chen, "Broadnet: A novel broad learning system based series ac arc fault detection approach with zero phase component analysis whitening transformation," IEEE Sensors Journal, 2024.

[20] W. Dai, X. Zhou, Z. Sun, Q. Miao, and G. Zhai, "Series AC Arc Fault Detection Method Based on L2/L1 Norm and Classification Algorithm," IEEE Sensors Journal, vol. 24, no. 10, pp. 16 661–16 672, 2024.

[21] A. Tang, Z. Wang, S. Tian, H. Gao, Y. Gao, and F. Guo, "Series Arc Fault Identification Method Based on Lightweight Convolutional Neural Network," IEEE Access, vol. 12, pp. 5851–5863, 2024.

[22] W. Dai, X. Zhou, Z. Sun, and G. Zhai, "Series alternating current arc fault detection method based on relative position matrix and deep convolutional neural network," Engineering Applications of Artificial Intelligence, vol. 136, p. 108874, 2024.

[23] A. Chabert, M. C. Bakkay, P. Schweitzer, S. Weber, and J. Andrea, "A transformer neural network for ac series arc-fault detection," Engineering Applications of Artificial Intelligence, vol. 125, p. 106651, 2023.

[24] L. Du, Z. Xu, H. Chen, and D. Chen, "Feature Selection-Based Low-Voltage AC Arc Fault Diagnosis Method," IEEE Transactions on Instrumentation and Measurement, vol. 72, pp. 1–12, 2023.

[25] R. Jiang and G. Bao, "Series Arc Fault Detection Method Based on Signal-Type Enumeration and Zoom Circular Convolution Algorithm," IEEE Transactions on Industrial Electronics, vol. 70, no. 10, pp. 10 607–10 617, 2023.

[26] R. Jiang and Y. Zheng, "Series Arc Fault Detection Using Regular Signals and Time-Series Reconstruction," IEEE Transactions on Industrial Electronics, vol. 70, no. 2, pp. 2026–2036, 2023.

[27] C. Jia, Z. Li, Y. Zhang, J. Cao, H. Chen, X. Zhao, Y. Du, Q. S. Cheng, N. Ma, and F. Qiu, "Serial arc-fault restriction survey with standard-listed products in the chinese market," IEEE Transactions on Industry Applications, vol. 59, no. 6, pp. 7453 – 7461, 2023.

[28] A. Dempster, D. F. Schmidt, and G. I. Webb, "Hydra: Competing convolutional kernels for fast and accurate time series classification," Data Mining and Knowledge Discovery, vol. 37, no. 5, pp. 1779–1805, 2023.

[29] Underwriters Laboratories Inc, "UL standard for arc-fault circuit-interrupters," 2011.

[30] IEC62606-2, General requirements for arc fault detection and protection devices (AFDDs), IEC International Standard 62 606-2, 2022.

[31] P. Schweitzer and F. Ferracuti, "Arc fault detection and appliances classification in AC home electrical networks using Recurrence Quantification Plots and Image Analysis [Data set]," DOI 10.5281/zenodo.3931688, 2020. [Online]. Available: https://zenodo.org/record/3931688

[32] ARM-Software, "CMSIS-NN," 2022, https://github.com/ARM-software/CMSIS-NN [Accessed: (05/02/2025)].

[33] N. Wu, "Arc fault detection data under different loads," 2024. [Online]. Available: https://dx.doi.org/10.21227/h2za-sg98