sensors

MDPI

Article

0-Dimensional Persistent Homology Analysis Implementation in Resource-Scarce Embedded Systems

Sérgio Branco [1,2], João G. Carvalho [1,3], Marco S. Reis [4], Nuno V. Lopes [3] and Jorge Cabral [1,2,*]

[1] Algoritmi Center, University of Minho, 4800-058 Guimarães, Portugal; id8676@alunos.uminho.pt (S.B.); b7380@algoritmi.uminho.pt (J.G.C.)
[2] CEiiA—Centro de Engenharia, Av. D. Afonso Henriques 1825, 4450-017 Matosinhos, Portugal
[3] DTx—Digital Transformation CoLab, University of Minho, 4800-058 Guimarães, Portugal; nuno.lopes@dtx-colab.pt
[4] CIEPQPF, Department of Chemical Engineering, University of Coimbra, Rua Sílvio Lima, Pólo II—Pinhal de Marrocos, 3030-790 Coimbra, Portugal; marco@eq.uc.pt
* Correspondence: jcabral@dei.uminho.pt; Tel.: +351-22-016-4800

Abstract: Persistent Homology (PH) analysis is a powerful tool for understanding many relevant topological features from a given dataset. PH allows finding clusters, noise, and relevant connections in the dataset. Therefore, it can provide a better view of the problem and a way of perceiving if a given dataset is equal to another, if a given sample is relevant, and how the samples occupy the feature space. However, PH involves reducing the problem to its simplicial complex space, which is computationally expensive and implementing PH in such Resource-Scarce Embedded Systems (RSES) is an essential add-on for them. However, due to its complexity, implementing PH in such tiny devices is considerably complicated due to the lack of memory and processing power. The following paper shows the implementation of 0-Dimensional Persistent Homology Analysis in a set of well-known RSES, using a technique that reduces the memory footprint and processing power needs of the 0-Dimensional PH algorithm. The results are positive and show that RSES can be equipped with this real-time data analysis tool.

Keywords: persistent homology; topological data analysis; embedded intelligence; intelligent resource-scarce embedded systems; TinyML

check for updates

Citation: Branco, S.; Carvalho, J.G.; Reis, M.S.; Lopes, N.V.; Cabral, J. 0-Dimensional Persistent Homology Analysis Implementation in Resource-Scarce Embedded Systems. *Sensors* **2022**, *22*, 3657. https://doi.org/10.3390/s22103657

Academic Editor: Ramon Gonzalez Carvajal

Received: 18 April 2022
Accepted: 6 May 2022
Published: 11 May 2022

1. Introduction

The interest in implementing Machine Learning and Data Analysis algorithms closer to the end-user (edge/end-computing) has increased over the last few years [1–4]. The need for these algorithms closer to the end-user is due to the need to reduce data traffic and cloud dependency. Therefore, this implementation reduces the device's power consumption and addresses many security, privacy, and safety concerns. However, the problem with implementing such processing and memory demanding algorithms in Resource-Scarce Embedded Systems (RSES) is the lack of memory, processing capabilities, and arithmetical units. Therefore, to achieve the goal of running these algorithms in RSES, it is vital to re-invent and make optimizations in the algorithms to achieve the same result as in a high-end computer [5].

Persistent Homology (PH) analysis allows understanding of a problem using Topological Data Analysis (TDA). TDA reduces a problem to its topological features, reducing the problem to its simplicial complex space. A simplicial complex space is represented in terms of points, holes, tetrahedrons, and other geometrical shapes. PH takes snapshots of the topological space at each step. This allows finding how the dataset behaves, what can be considered noise during the sampling process to find clusters, and extracting important features. The PH and TDA tools are widely known because Euler used them to solve the Seven Bridges of Königsberg problem. These analysis techniques allow one to reduce the problems into their main features, which turn the increasingly used high-dimensional data into easy-to-see and easy-to-understand problems [6,7].

Persistent Homology analysis has helped solve problems across multiple fields, such as medicine [8–11], chemistry [12,13], image processing [14], physics [15], astronomy [16], and sensor networks [17,18]. Therefore, Persistent Homology Analysis has been implemented in a wide range of programming languages (e.g., Cpp [19–22], Python [19,23–26], Java [27], R [28], Matlab [27,29], and Julia [30]), and it is available in distinct packages that allow using this analysis method across multiple projects. Besides the wide range of software available, none is written in plain C, tested in resource-scarce embedded systems, and neither is designed to perform in such resource-constraint devices.

Besides the many advantages of PH, the tool is computational and memory expensive. The complexity and memory footprint increase as the dimension at which the analysis is made. Therefore, implementing PH in RSES requires looking for novel ways of solving the problem. However, as more and more devices are sampling data in distinct environments, it is necessary to provide RSES with tools for real-time data analysis. This can make the devices understand if the dataset being collected is not behaving as it should or reduce the quantity of traffic sent to the Cloud. Therefore, implementing it into these devices is a challenge but a powerful add-on to reduce cloud dependency and spread the computational work across the network.

The present works focus on the implementation of the data analysis tool, so-called 0 Dimensional PH analysis, proposing a methodology that does not comprise the memory and processing footprint of such resource-constrained devices by resorting to lightweight techniques, such as bitmask and Boolean logic. As shown, the proposed solution can provide outcoming similar PH images but is drawn from fewer samples than the main topological features. With this solution, RSES devices do not need to connect to a Cloud System, which opens the door to making RSES less cloud-dependent and spreading the data analysis task across the network. To the best of our knowledge, this is the first attempt to deploy 0 dimensional PH analysis tool in resource-constrained devices.

2. Persistent Homology Analysis

TDA is responsible for extracting the topological features of a given dataset. TDA uses a distance function (e.g., Euclidean Distance). However, instead of just looking at the distance between points, TDA wants to see how the points in a dataset connect. A connection between two points happens if they have a distance equal to or less than r. Therefore, one can think of an i-th dimensional sphere around each point with radius r and the center equal to the point's coordinates. If two i-th dimensional spheres touch each other, the points have a connection.

The TDA creates a simplicial space consisting of multiple simplicial complexes. A simplicial complex is a point, a line, a triangle, or any other shape formed by connecting one or multiple points. Therefore, for each possible r, there is a single simplicial space. As it is impossible to know which r provides the simplicial space that compresses the main topological features, a study is made using multiple rs.

PH is the method that takes snapshots of a simplicial space at a given r and then stores the information using a barcode. A barcode consists of a bar graph displaying the lifetime of an i-th dimensional hole. Because the main focus of this paper is 0-dimensional PH, one will focus on explaining the behavior of 0-dimensional barcodes and simplicial spaces [31–34].

In 0-dimensional PH, a hole dies when a connection between two points happens. Therefore, each point in 0-dimensional Homology is seen as a simplicial complex. Therefore, for 0-dimensional PH, one looks at the connection between points and not the empty space between them. One can find possible clusters by keeping track of the number of simplicial complexes at each r. Simplicial complexes that die at greater distances are possible point clusters. Simplicial complexes that die earlier are seen many times as noise, and the r is used as a filtration value.

For the 0-dimensional PH analysis, it is possible to withdraw two conclusions: The first simplicial complex will die at r equal to the minimum distance between points; The last two simplicial complexes join in the worst-case scenario in a r equal to the maximum distance between points.

Figure 1 shows how a 0-dimensional PH happens. The top image shows the dataset, with the orange representing the i-th dimensional sphere growing around each point. A connection (black line) appears if two orange circles touch. For each connection, a bar appears at the barcode on the bottom.

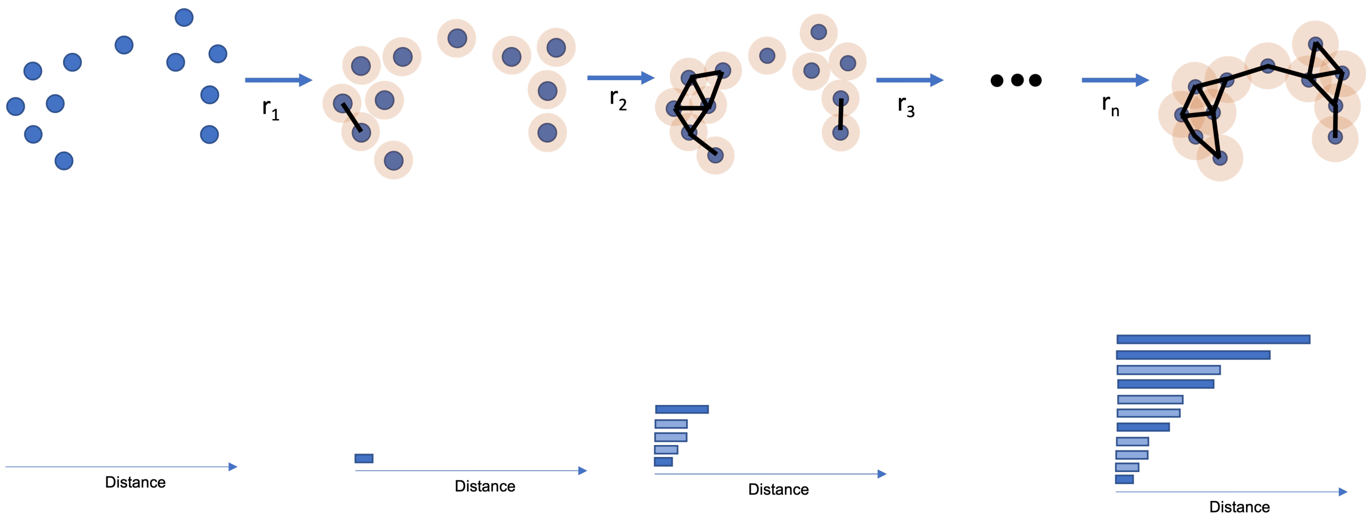

Figure 1. Example of a 0-dimensional Persistent Homology Analysis. At the top, the connections between points form as r increases. At the bottom, the barcode originated from the analysis. Each bar represents the distance at which a component has died.

3. Methodology

The research method followed for the implementation lies in five steps.

The first was to retrieve the barcode image, distance matrix, and simplices information from an existing TDA software. The chosen software was the Ripser (version 0.6.1) from the Scikit-TDA Package. Therefore, the authors verify if the distance matrix obtained from the third-party tool matches the distance matrix built using our method.

The second step focused on building the barcode from the entire dataset using the developed method. Therefore, a comparison with the barcode obtained from Ripser was made. It is verified if the death of the first simplice and the last one occur at the same distance in both tools. Afterward, the authors have withdrawn the unique values from each barcode array, verifying if the values are the same, without a significant gap or a difference higher than 5%. This step verifies that the tool works accordingly to what is expected. Due to the lack of resources in the selected platforms, this step should first be run on a personal computer.

The third step was to verify if the death of the last simplice is the same as the maximum distance found in the distance matrix. Therefore, proving that as soon as one obtains a single simplice, one no longer has to verify for distances higher than the maximum distance already obtained.

The fourth step was to build a visual representation of the dataset using a dimensional reduction tool, verifying if the number of relevant simplices (last three or four) are expected to have larger or smaller values in terms of distance. Therefore, if the given representation provides two clusters significantly distant in the euclidean space, the two bars representing at which distance only two simplices exist up until they merge into one should have a significant difference in size.

The fifth step is to reduce the number of samples through random selection and verify if a smaller batch will provide the same information as the entire dataset. Therefore, proving that the method should and could be implemented into RSES.

The main concern in implementing 0-dimensional homology analysis in a resource-scarce embedded system is the memory needed to hold the entire data structure. Most common implementations of homology use the euclidean distance between points (samples) to analyze, the device needs to store some samples. This set needs to be big enough to make assumptions but small enough not to fill the entire device's memory. Therefore, the sample matrix and the number of samples (N) are the first variables to consider.

Because 0-dimensional homology looks at the distance, the device must also store the distance between samples in a matrix for further analysis. Since the distance matrix has the main diagonal composed of 0 s, and the upper and lower triangles mirror each other, the entire matrix can be compressed into a small array. N is the number of samples, and for our method to work, N should always be given by 2^p, with p an integer higher than one, and $p \in \mathbb{N}$. The array's size can be withdrawn by re-writing the combination formula as the authors propose in Equation (1).

$$\binom{2^p}{2} = \frac{2^p!}{2!(2^p-2)!} \tag{1}$$

After the distance matrix is calculated, the method iterates through each distance (r) inside the distance matrix. The consideration of each r being a distance value in the matrix is because the first component will die at the minimum distance existing in the matrix, and two components can only exist until the maximum distance in the matrix. For each r, a connection matrix is built. This matrix is a bitmask matrix of row size equal to the number of samples, and each row has a bit size equal to the number of samples (Figure 2a). This is the reason for N to be given by 2^d. The bit j of row i takes the value 1 if sample i and j are at a distance equal to or smaller than r.

The connection matrix allows using bitwise logic to search for connected components. If the AND bitwise logic between rows i and j have any bit equal to 1, then i and j are connected. The OR bitwise logic between the two rows (after the AND) gives the entire list of connected points (the simplice). The OR mask is stored at each iteration to avoid repeating simplices, which shortens the processing time. Each simplice is stored and added to a linked list afterward (Figure 2b). Because the matrix has a bit-size of $(N_{samples})^2$, doubling the sampling size quadruples the memory needed. Notice that the connection matrix is a mirror matrix with the main diagonal composed of 1s. Therefore, it is possible to implement the same technique used for the distance matrix. However, it will take more processing but reduce the memory needed.

For distance r, the method builds the entire set of simplices that exist. Afterward, the method searches in the BarCode linked list for any node with the same number of simplices. If any match occurs, the method verifies if r is higher than the distance r_m stored in the node. If true, r_m is changed by r, otherwise, r_m continues, as the death value for simplices. If no node matches the simplices' number, then a new node is created (Figure 3).

Algorithms 1 and 2 explain the pseudo-code used to build the 0-Dimensional Persistent Homology tool.

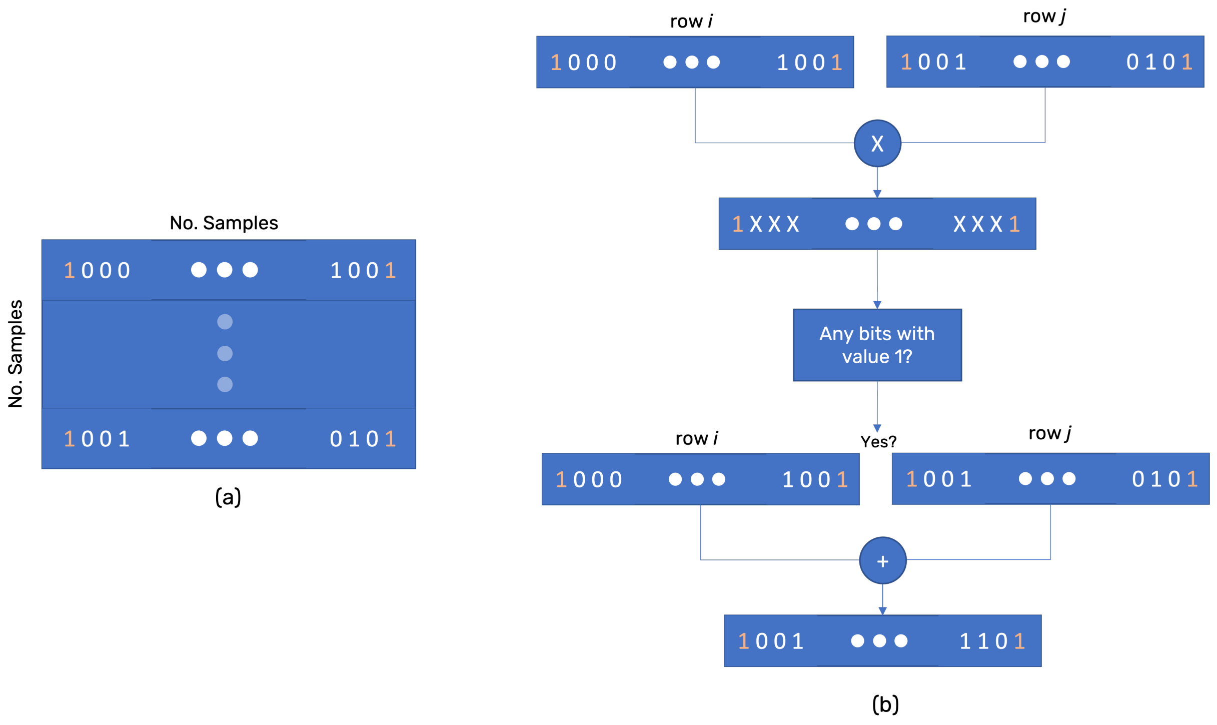

Figure 2. (**a**) The connection matrix has a row size equal to the number of samples. Each row has a bit size equal to the number of samples, implemented using 32-bit unsigned integers. If sample i and j are connected, then bit j in row i has value 1 and bit i in row j has value 1. (**b**) If the bitwise AND result between row i and j have any bit with value 1, the two points have a connection path. If they have a connection path, then the bitwise OR gives us the entire connected points set.

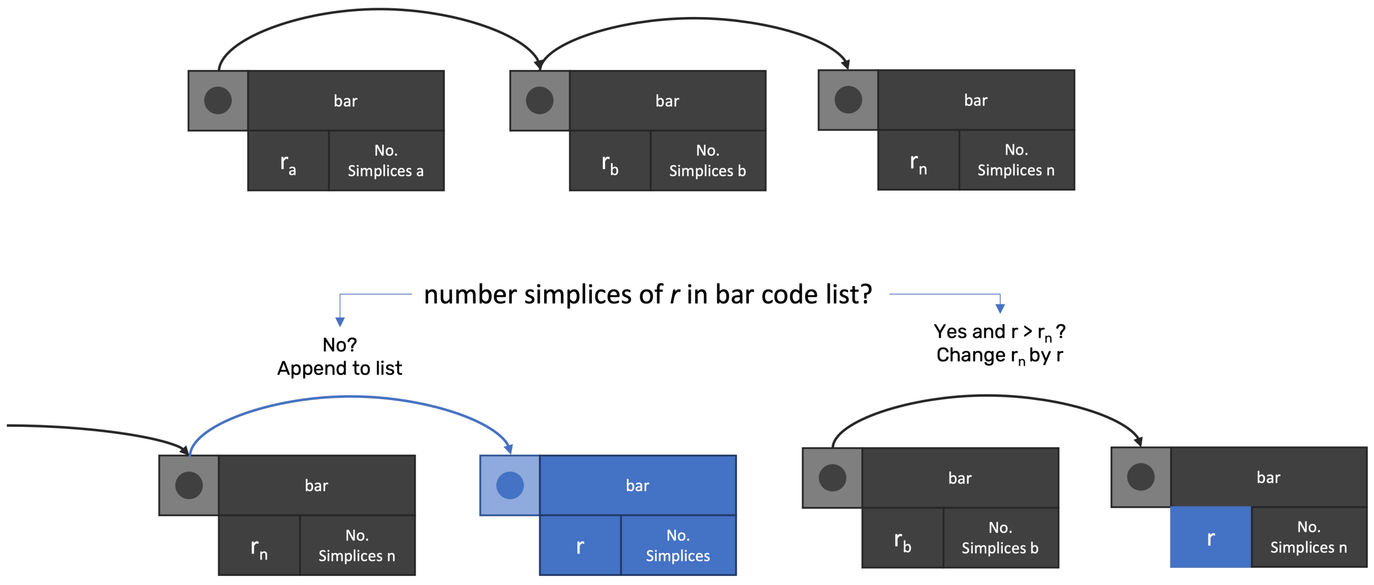

Figure 3. After building the simplices for distance *r*, the method searches any node in the Bar Code list with the same number of simplices. If no node matches the search, a new node is created, storing distance *r* and the number of simplices. If a match happens, it checks if the distance is higher than *r* and changes it if true.

Algorithm 1 Build BarCode.

```
build_distance_matrix()
for r in distance_matrix do
    if r > r_max or r == 0 then
        ignore
    else
        no_simplices = build_simplices_for_r()
    end if
end for
for element in simplices_linked_list do
    if element.no_simplices == no_simplices and element.r < r then
        element.r = r
        break
    end if
end for
if r not in simplices_linked_list then
    add_to_linked_list()
end if
```

Algorithm 2 Build Simplices for distance r.

```
for i in range(no_samples) do
    for j in range(no_samples) do
        if distance_matrix[i][j] < r then
            connection_matrix[i] |= (1«j)
        end if
    end for
end for
for i in range(no_samples) do
    if i not in seen then
        logical_or_for_all_1_bits()
        np_simplices += 1
    end if
end for
return no_simplices
```

4. Experimental Design

The authors have selected a wide range of development platforms that vary in processing power, memory, and hardware to test the hypothesis. The selected platforms and their main specs are present in Table 1. To avoid developing specific code for each platform, the authors have used a generic tool platformio and the same framework (Arduino) to program all boards in equal terms.

Table 1. Experimental tests platform comparison.

Platform	MCU	Cortex	Clock (MHz)	Flash (kB)	RAM (kB)	FPU
Arduino M0+ Pro	ATSAMD21G18	M0	48	256	32	No
Arduino Due	AT91SAM3X8E	M3	84	512	96	No
EK-TM4C123GXL	TM4C123GH6PMI	M4F	80	256	32	Yes
NodeMCU ESP8266	ESP8266	-	80	1024	128	No
STM32F767ZIT6	STM32F767ZI	M7	216	2048	512	Yes

The IRIS dataset was selected to check if the results are equal on all platforms and quickly verify the method's integrity. The dataset is simple in terms of features and classes. However, tests were realized in other datasets to ensure that the results were equally verified.

4.1. PCA Representation of the IRIS Dataset

The IRIS dataset represents a classification problem that intends to use four features to classify iris flowers into three species. Therefore, the dataset occupies a four-dimensional Euclidean space. To verify if the results obtained for the 0-dimensional Homology analysis are correct and representative of the dataset behavior, one must try to understand the expected result from such analysis. As the dataset is represented in four dimensions, a Principal Component Analysis (PCA) must be done.

PCA is a tool used to lower the dimension of any dataset. It makes an orthogonal linear reduction that projects the feature space into a subspace generated by a few eigenvectors. So, PCA does not necessarily discard features; it combines them.

Therefore, by making a PCA analysis of the IRIS dataset, it was possible to reduce the dataset into a two-dimensional space (Principal Components 1 and 2), which compresses much of the dataset's relevant information and allows for a graphical representation of the feature space. Figure 4 shows the PCA representation of the dataset for the first two principal components. As the image shows, the expectation is to see four bars live longer because of the four clusters (three are well visible, and the fourth is composed of the two top-right green dots).

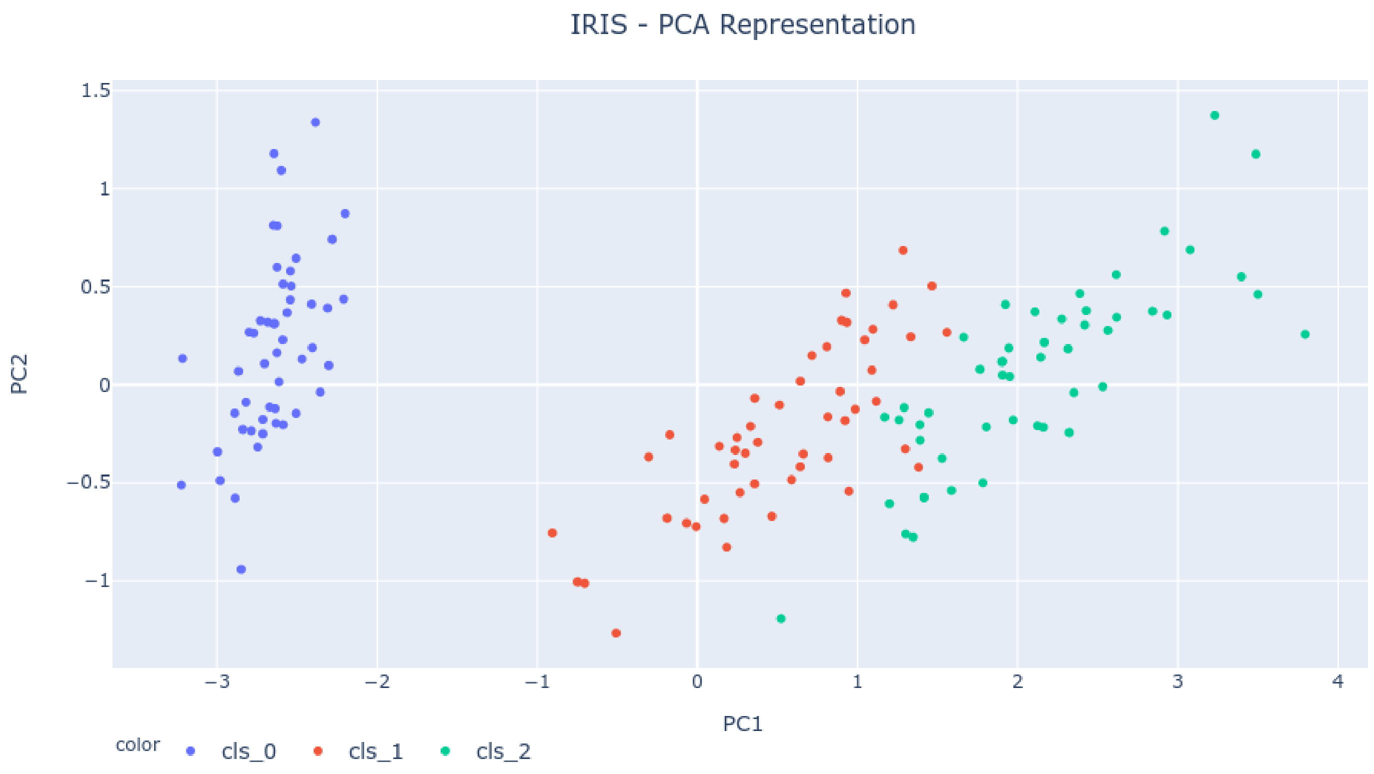

Figure 4. IRIS Principal Components Analysis representation.

4.2. Comparison of Proposed Method against Third-Party Tool Scikit-TDA Risper

Before running any tests in an MCU, it was essential to test if the code running in a standard computer could achieve the same results as a third-party generic tool available. The selected third-party tool was the Ripser available in the Python package Scikit-TDA. Figures 5 and 6 show the bar codes obtained by analyzing the dataset in both tools. The results are very similar, and we can see three long bars form. The developed method has one more bar at the end because the authors have decided not to use the infinity value for the one component value but the death value of the second component.

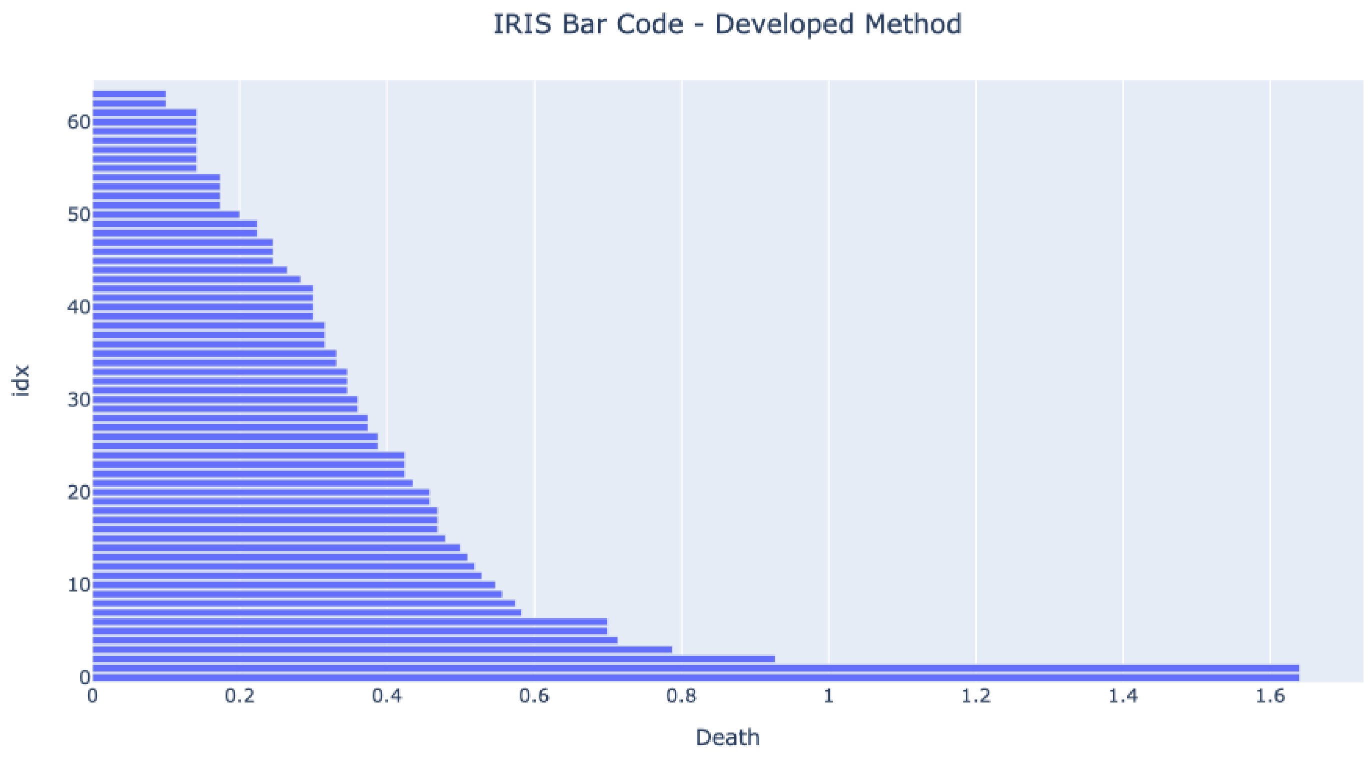

Figure 5. Bar Code obtained from running the developed 0-dimensional homology analysis on the IRIS dataset. The X-axis represents the death of each component. The Y-axis is the array index.

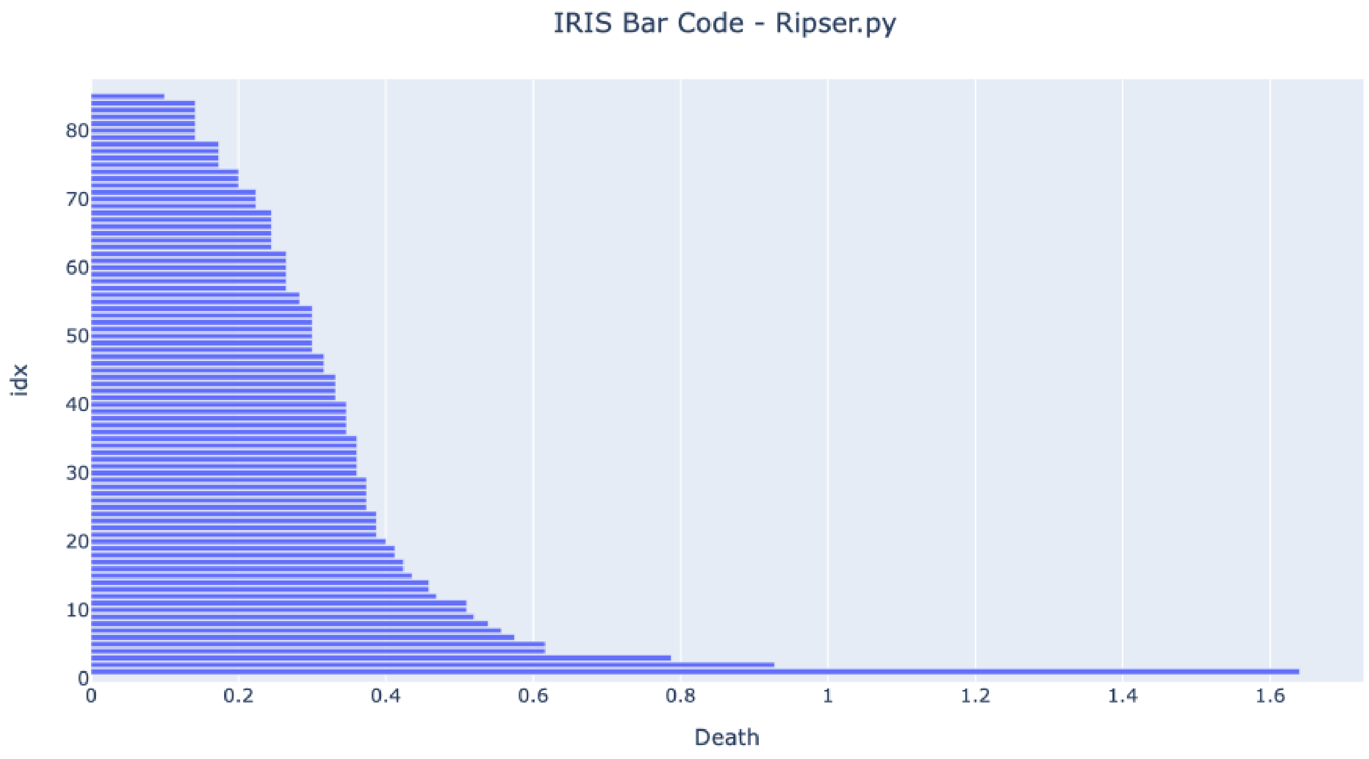

Figure 6. Bar Code obtained from running the scikit-tda ripser 0-dimensional homology analysis on the IRIS dataset. The X-axis represents the death of each component. The Y-axis is the array index.

The barcodes are similar, meaning that the developed method provides the same results as other proven methods. The figures show that the expected bar code behavior happens once the three bars that live longer than any other represent the four clusters.

5. Results

Tables 2 and 3 displays the profiling results obtained from running the 0-Dimensional Homology Analysis in each MCU for 32-Samples and 64-Samples, respectively. The 64-Samples analysis was only possible for the STM32F767 because it had enough memory to allocate (100 KB) for the iteration that spent more memory.

As the tables show, every platform could perform the 32-Samples analysis. From the tests, it is possible to understand that an MCU needs to have at least 14KB free for dynamic allocation to perform such analysis. Only the NodeMCU ESP8266 has allocated a different amount of memory, which is not understandable by the bare eye. A possibility is that the NodeMCU implements any memory protection mechanism or alignment to avoid errors or increase performance.

The execution time seems to be mainly linked to the clock speed, but other variables need to be considered. The EK-TM4C123GXL has a slower clock than the Arduino Due, but it runs the analysis slightly faster. One possibility is because the EK-TM4C123GXL has an FPU.

The EK-TM4C123GXL has the same clock speed as the NodeMCU ESP8266, but the last one spends less time to achieve a result. However, as Figure 7 shows, the result differs from all the other ones, which indicates that it handles the arithmetics in a distinct way than the others.

The Bar Codes in Figures 7–9 present the results obtained from running the IRIS dataset 0-dimensional analysis on 32 samples. The barcodes are equal in all platforms (with small differences for the NodeMCU 8266). Therefore, the method is platform-independent. The barcode is also similar to the one obtained from the third-party tool. The important thing to notice is that the death of the last two simplices (to merge into one) happens at a greater distance than all others. This is the expected behavior by looking at the PCA Analysis.

The 64-sample analysis was possible only for the STM32F767 platform (Figure 10). The barcode presents more bars and is closer to the computer analysis, but the most important features exist in all bar codes.

Therefore, the following study shows that reducing the number of samples is possible without losing the persistent homology dataset's fundamentals. However, this may not be true for all datasets.

Table 2. Profiling results for 32 samples 0-Dimensional Homology Analysis. The memory is in bytes. The Dynamic Memory field represents the maximum amount of memory used by an iteration. The table is ordered by execution time.

Platform	Flash Memory	Ram Memory	Dynamic Memory	Execution Time (ms)
Arduino M0+ Pro	23,208	7348	13,828	959
Arduino Due	18,396	6764	13,828	505
EK-TM4C123GXL	14,797	8293	13,828	422
NodeMCU ESP8266	270,101	33,188	15,856	267
STM32F767ZIT6	21,596	6292	13,828	70

Table 3. Profiling results for 64 samples 0-Dimensional Homology Analysis. The memory is in bytes. The Dynamic Memory field represents the maximum amount of memory used by an iteration. The table is ordered by execution time.

Platform	Flash Memory	Ram Memory	Dynamic Memory	Execution Time (ms)
STM32F767ZIT6	23,070	19,399	98,344	1055

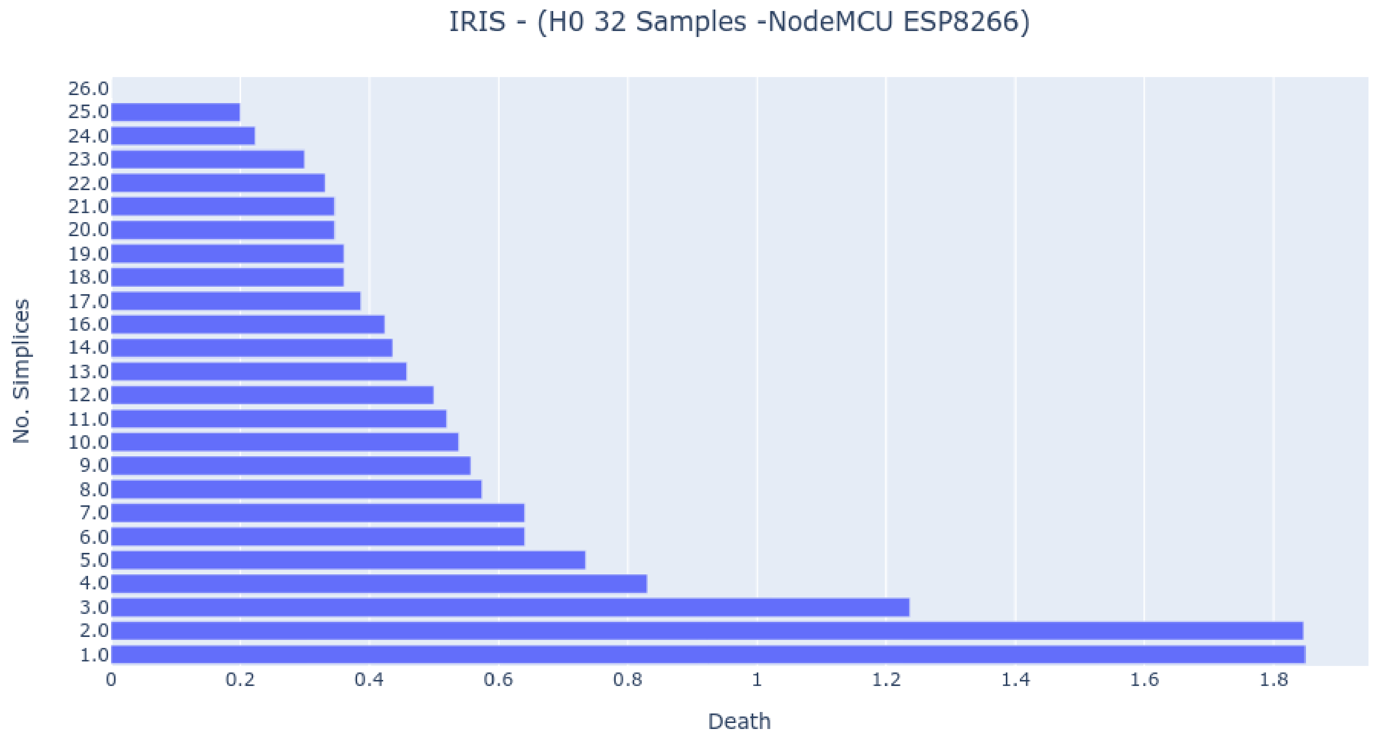

Figure 7. Bar Code result from 0-dimensional Homology analysis in NodeMCU 8266 platform. The analysis was obtained by randomly sampling 32 samples from the IRIS dataset.

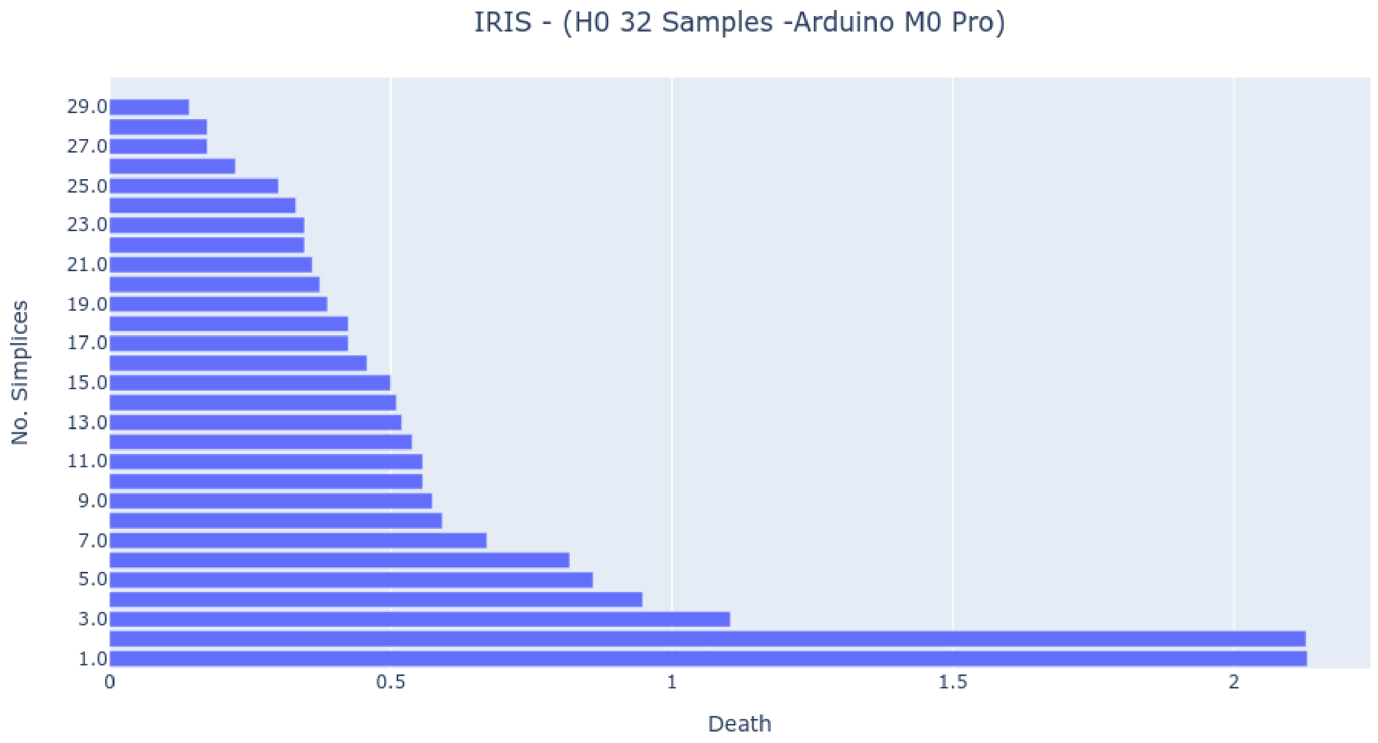

Figure 8. Bar Code result from 0-dimensional Homology analysis in Arduino M0+ Pro platform. The analysis was obtained by randomly sampling 32 samples from the IRIS dataset.

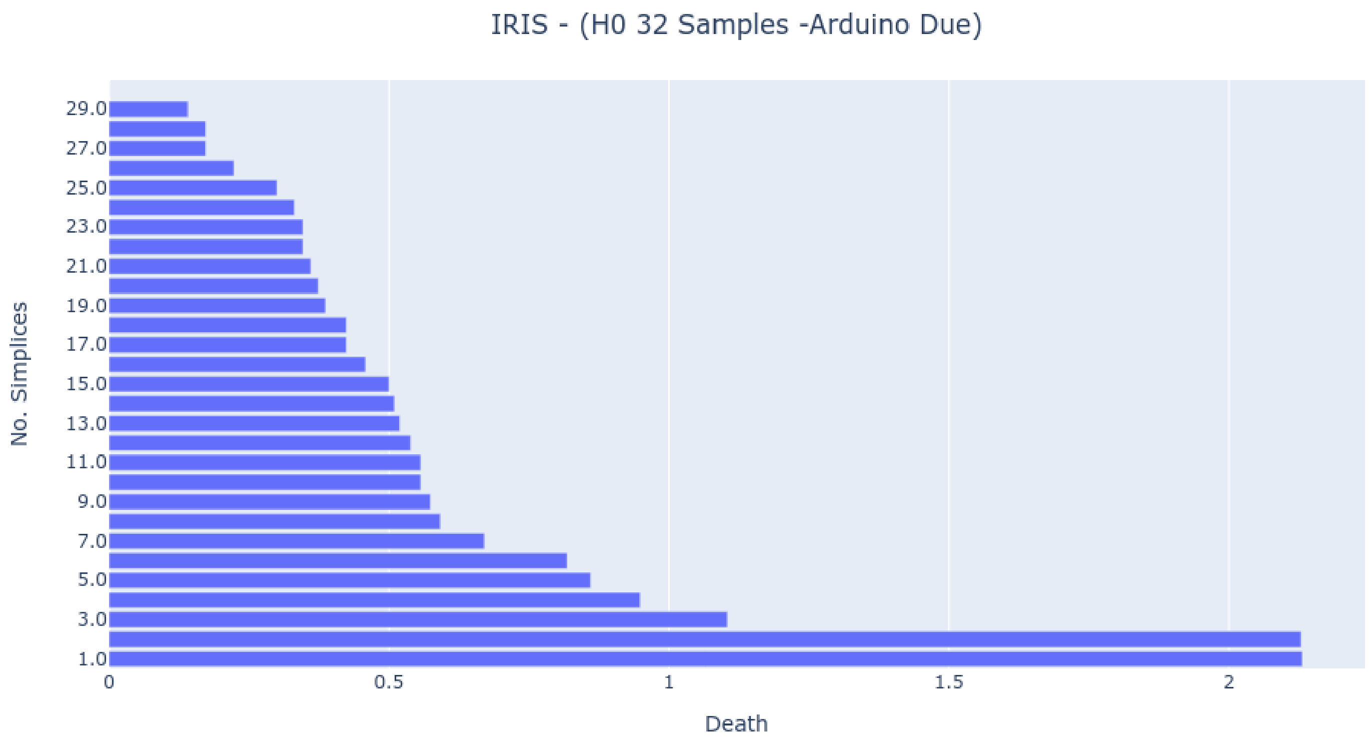

Figure 9. Bar Code result from 0-dimensional Homology analysis in Arduino Due platform. The analysis was obtained by randomly sampling 32 samples from the IRIS dataset.

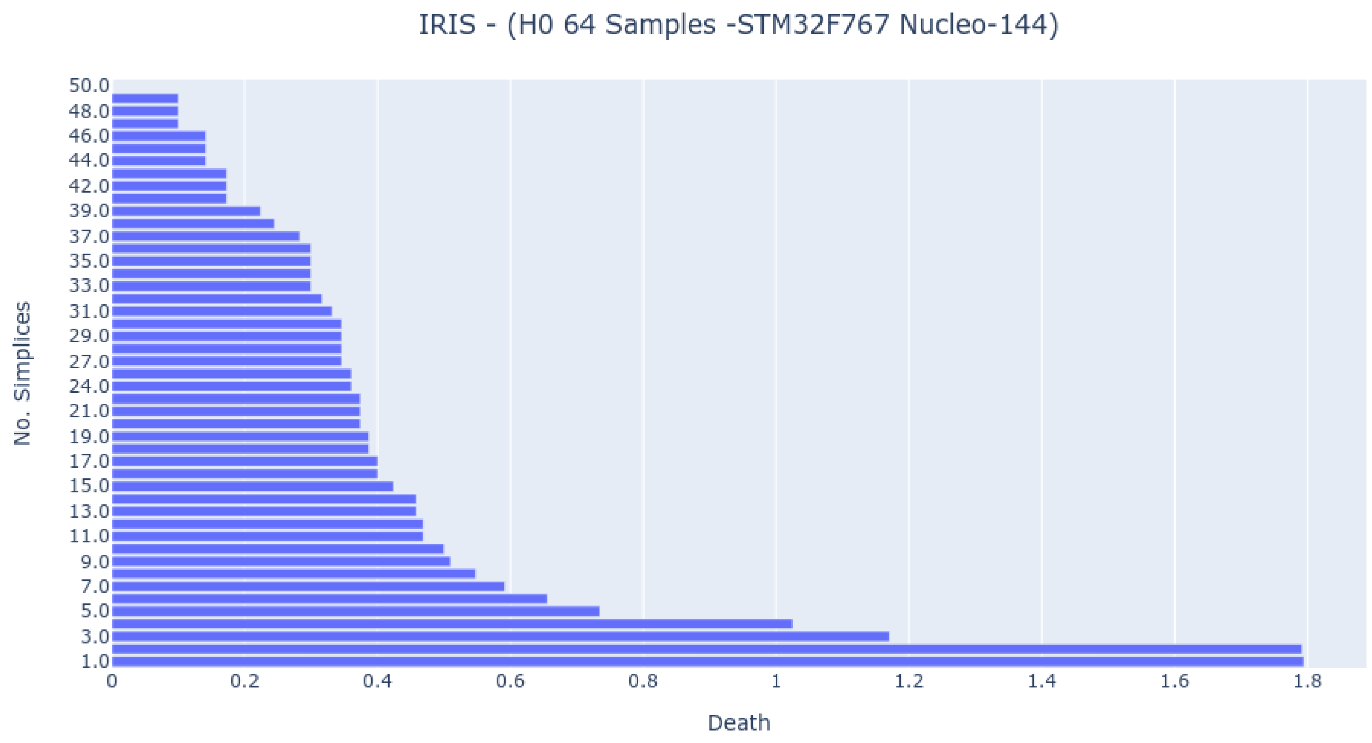

Figure 10. Bar Code result from 0-dimensional Homology analysis in STM32F767 Nucleo-144 platform. The analysis was obtained by randomly sampling 64 samples from the IRIS dataset.

6. Conclusions

The presented work has shown the implementation of 0-dimensional Persistent Homology analysis in a wide range of Resource-Scarce Embedded Systems. The devices select encompass a variety of MCU architectures, families, and memory/processing capabilities.

The work has shown the amount of memory and processing time it takes to achieve the expected result according to the available hardware. Moreover, it is possible to see that the usage of 32/64 samples can achieve the same results as the usage of the entire dataset. Therefore, PH is also a stable and practical method that requires only a variety of samples to present a good image of the entire dataset, at least for a lower-dimensional dataset.

7. Future Work

The authors consider that the following future directions of the researcher should be:

- Implementation of higher level PH in RSES for very high dimensional datasets.
- Creation of an algorithm to compare barcodes to verify if the collected dataset represents data obtained previously.
- Develop a dynamic algorithm to re-draw the barcode as new samples arrive.
- Implementation of a federated system to compare multiple barcodes as they are collected.

Author Contributions: Conceptualization, S.B.; methodology, S.B.; formal analysis, J.G.C.; writing—original draft, S.B.; writing—review and editing, J.G.C.; supervision, J.C., M.S.R. and N.V.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by FCT—Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

Data Availability Statement: The complete code implementation is available at https://github.com/asergiobranco/mcu_homology (accessed on 15 January 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Guo, B.; Zhang, D.; Wang, Z. Living with internet of things: The emergence of embedded intelligence. In Proceedings of the 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing, Dalian, China, 19–22 October 2011; pp. 297–304.
2. Guo, B.; Zhang, D.; Yu, Z.; Liang, Y.; Wang, Z.; Zhou, X. From the internet of things to embedded intelligence. *World Wide Web* **2013**, *16*, 399–420. [CrossRef]
3. Dai, W.; Nishi, H.; Vyatkin, V.; Huang, V.; Shi, Y.; Guan, X. Industrial edge computing: Enabling embedded intelligence. *IEEE Ind. Electron. Mag.* **2019**, *13*, 48–56. [CrossRef]
4. Silva, A.; Fernandes, D.; Névoa, R.; Monteiro, J.; Novais, P.; Girão, P.; Afonso, T.; Melo-Pinto, P. Resource-Constrained Onboard Inference of 3D Object Detection and Localisation in Point Clouds Targeting Self-Driving Applications. *Sensors* **2021**, *21*, 7933. [CrossRef]
5. Branco, S.; Ferreira, A.G.; Cabral, J. Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey. *Electronics* **2019**, *8*, 1289. [CrossRef]
6. Carlsson, G. Topology and data. *Bull. Am. Math. Soc.* **2009**, *46*, 255–308. [CrossRef]
7. Gowdridge, T.; Dervilis, N.; Worden, K. On Topological Data Analysis for SHM: An Introduction to Persistent Homology. In *Data Science in Engineering*; Springer: Berlin/Heidelberg, Germany, 2022; Volume 9, pp. 169–184.
8. Chung, M.K.; Bubenik, P.; Kim, P.T. Persistence diagrams of cortical surface data. In Proceedings of the International Conference on Information Processing in Medical Imaging, Williamsburg, VA, USA, 5–10 July 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 386–397.
9. Garside, K.; Henderson, R.; Makarenko, I.; Masoller, C. Topological data analysis of high resolution diabetic retinopathy images. *PLoS ONE* **2019**, *14*, e0217413. [CrossRef]
10. Lawson, P.; Sholl, A.B.; Brown, J.Q.; Fasy, B.T.; Wenk, C. Persistent homology for the quantitative evaluation of architectural features in prostate cancer histology. *Sci. Rep.* **2019**, *9*, 1139. [CrossRef]
11. Caputi, L.; Pidnebesna, A.; Hlinka, J. Promises and pitfalls of Topological Data Analysis for brain connectivity analysis. *NeuroImage* **2021**, *238*, 118245. [CrossRef]
12. Motta, F.C.; Neville, R.; Shipman, P.D.; Pearson, D.A.; Bradley, R.M. Measures of order for nearly hexagonal lattices. *Phys. D Nonlinear Phenom.* **2018**, *380*, 17–30. [CrossRef]
13. Townsend, J.; Micucci, C.P.; Hymel, J.H.; Maroulas, V.; Vogiatzis, K.D. Representation of molecular structures with persistent homology for machine learning applications in chemistry. *Nat. Commun.* **2020**, *11*, 3230. [CrossRef]
14. Bendich, P.; Edelsbrunner, H.; Kerber, M. Computing robustness and persistence for images. *IEEE Trans. Vis. Comput. Graph.* **2010**, *16*, 1251–1260. [CrossRef] [PubMed]

15. Suzuki, A.; Miyazawa, M.; Minto, J.M.; Tsuji, T.; Obayashi, I.; Hiraoka, Y.; Ito, T. Flow estimation solely from image data through persistent homology analysis. *Sci. Rep.* **2021**, *11*, 17948. [CrossRef] [PubMed]
16. Heydenreich, S.; Brück, B.; Harnois-Déraps, J. Persistent homology in cosmic shear: Constraining parameters with topological data analysis. *Astron. Astrophys.* **2021**, *648*, A74. [CrossRef]
17. De Silva, V.; Ghrist, R. Coverage in sensor networks via persistent homology. *Algebr. Geom. Topol.* **2007**, *7*, 339–358. [CrossRef]
18. De Silva, V.; Ghrist, R. Homological sensor networks. *Not. Am. Math. Soc.* **2007**, *54*, 10–17.
19. Morozov, D. Dionysus Software. 2012. Available online: https://mrzv.org/software/dionysus2/ (accessed on 30 November 2021).
20. Bauer, U.; Kerber, M.; Reininghaus, J. Distributed computation of persistent homology. In Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX), Portland, OR, USA, 5 January 2014; pp. 31–38.
21. Bauer, U.; Kerber, M.; Reininghaus, J.; Wagner, H. Phat–persistent homology algorithms toolbox. *J. Symb. Comput.* **2017**, *78*, 76–90. [CrossRef]
22. Lewis, R. CTL. Available online: https://github.com/appliedtopology/ctl (accessed on 30 November 2021)
23. Maria, C.; Boissonnat, J.-D.; Glisse, M.; Yvinec, M. The gudhi library: Simplicial complexes and persistent homology. In Proceedings of the International Congress on Mathematical Software, Seoul, Korea, 5–9 August 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 167–174.
24. Tralie, C.; Saul, N.; Bar-On, R. Ripser. py: A lean persistent homology library for python. *J. Open Source Softw.* **2018**, *3*, 925. [CrossRef]
25. Tierny, J.; Favelier, G.; Levine, J.A.; Gueunet, C.; Michaux, M. The topology toolkit. *IEEE Trans. Vis. Comput. Graph.* **2017**, *24*, 832–842. [CrossRef]
26. Tauzin, G.; Lupo, U.; Tunstall, L.; Pérez, J.B.; Caorsi, M.; Medina-Mardones, A.M.; Dassatti, A.; Hess, K. giotto-tda:: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration. *J. Mach. Learn. Res.* **2021**, *22*, 1–6.
27. Adams, H.; Tausz, A.; Vejdemo-Johansson, M. JavaPlex: A research software package for persistent (co) homology. In Proceedings of the International Congress on Mathematical Software, Seoul, Korea, 5–9 August 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 129–136.
28. Fasy, B.T.; Kim, J.; Lecci, F.; Maria, C. Introduction to the R package TDA. *arXiv* **2014**, arXiv:1411.1830.
29. Mendoza-Smith, R.; Tanner, J. Parallel multi-scale reduction of persistent homology filtrations. *arXiv* **2017**, arXiv:1708.04710.
30. Henselman, G. Eirene. Available online: https://github.com/Eetion/Eirene.jl (accessed on 30 November 2021)
31. Björner, A. Topological methods. In *Handbook of Combinatorics*; North Holland: Amsterdam, The Netherlands, 1995; Volume 2, pp. 1819–1872.
32. Hirzebruch, F.; Borel, A.; Schwarzenberger, R.L.E. *Topological Methods in Algebraic Geometry*; Springer: Berlin/Heidelberg, Germany, 1966; Volume 175.
33. Otter, N.; Porter, M.A.; Tillmann, U.; Grindrod, P.; Harrington, H.A. A roadmap for the computation of persistent homology. *EPJ Data Sci.* **2017**, *6*, 1–38. [CrossRef] [PubMed]
34. Carter, N. *Data Science for Mathematicians*; CRC Press: Boca Raton, FL, USA, 2020.