Cloud Native Arrhythmia Classification

Di Lullo Marco*
Department of Science and Technology
University of Naples Parthenope
Naples, Italy
marcodilullo2714@gmail.com

Gicchino Salvatore Alberto*
Department of Science and Technology
University of Naples Parthenope
Naples, Italy
g.salvatore93@gmail.com

Pagliarulo Gaetano*
Department of Science and Technology
University of Naples Parthenope
Naples, Italy
gaetano.pagliarulo96@outlook.it

Spoleto Antonio Junior*
Department of Science and Technology
University of Naples Parthenope
Naples, Italy
spoletoantoniojr@gmail.com

Vitale Maria Concetta*
Department of Science and Technology
University of Naples Parthenope
Naples, Italy
cvitale819@gmail.com

Abstract

Nowdays, cardiological tools are only capable of offering a statistical report which does not offer valuable help to recognize common arrhythmias. Furthermore, existing AI-driven solutions require in-house implementation and constant maintenance. This paper presents a cloud native application running on Google Cloud Platform which provides an intuitive web interface. In this way, continuous management is decoupled from the public structure altogether, making the cloud approach more appealing. Having an always available deep learning solution for heart diseases analysis can be useful for a quick and preemptive patient examination before the actual medical input. Our contribution is the development of a cloud native solution, making the service highly scalable and resilient, as opposed to in-house configurations. We propose a three stateless microservices architecture that hierarchically manages the data flow and the inference, the latter being carried on by a two stage neural network which firstly handles the noise separation and secondly the classification. We run benchmarks on both our cloud solution and a local hosted equivalent to assess scalability and performance differences.

Keywords: Cloud computing, Arrhythmia classification, Heart diseases, ECG, Kubernetes, Microservices, PaaS, Serverless

1 Introduction

1.1 Contextualization and Motivation

Heart diseases are one of the major death causes. The electrocardiogram (ECG) is a complex signal and contains crucial information regarding the functionality of cardiovascular system. In fact it is one of the most convenient and non-invasive tools for monitoring peoples' heart condition. ECG is recorded over a period of time by placing electrodes and leads on the human body. ECG can be used for diagnosing a wide range of heart diseases, including Cardiac Arrhythmia, Acute Coronary Syndrome, etc. Given this large amount of possible outcomes, cardiologists may have trouble figuring out what the patient's condition is in a short period of time. Moreover, usually hospital wards can be overcrowded and the only few available medics and medical resources may not be enough to handle patient demands right away. Most sophisticated electrocardiography devices can be cumbersome because they require medical feedback from a cardiologist, and a large amount of time is needed to properly place the electrodes on the patient. Finally, many computer-aided ECG disease detection exist, but they require extensive and onerous management from the hospital point of view. The cloud computing has gained more and more popularity in recent years. A cloud-based health care provider bears many important features. First, it can be accessed using any device such as a web browser or a mobile phone from anywhere. Users no longer need to install the software at their local machines. Second, the cost for cloud-computing is greatly reduced based on the pay-per-usage scheme. Third, computing resources are provided in an elastic way so that additional resources will be automatically provided only when they are needed, which is often referred to as the horizontal scalability. In addition, the cloud computing system is designed to be geographically scalable, which means the performance and usefulness of the system can be maintained regardless of the geographic pattern. We propose a solution that includes the usage of cloud computing in conjunction with artificial intelligence to address the discussed problems.

1.2 Related work

Existing commercial ECG disease detection methods show substantial rates of misdiagnosis due to the limitations of the abilities of extracted features, and the lack of generalization on different medical devices. For this reason, many works

Authors contributed equally.
https://github.com/ParthenopeCloudTeam/CloudNativeECG-Classification

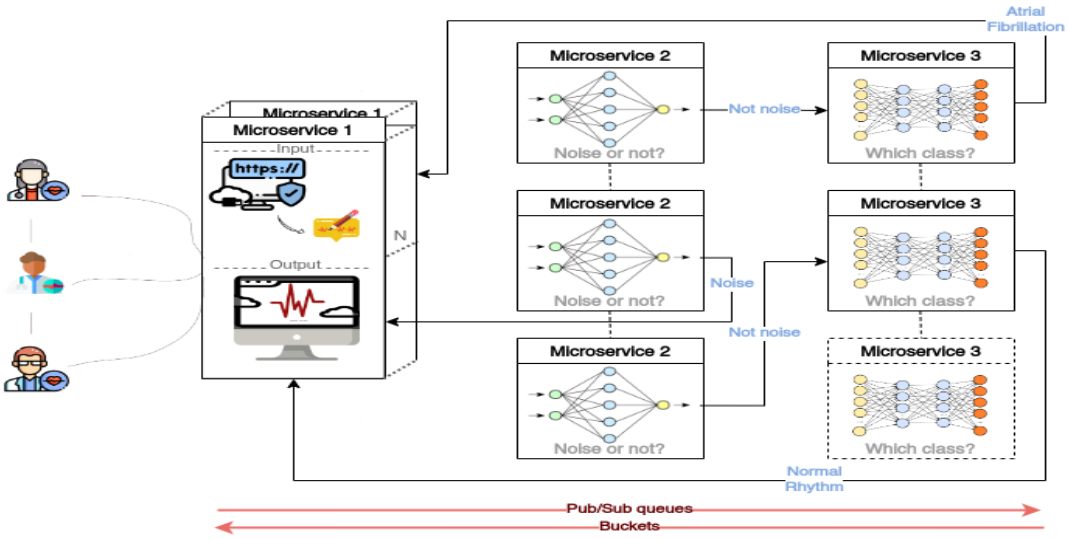

Figure 1. Architecture schema and data flow. A dotted box represents an idle container.

in this field take advantage of deep learning methods which have shown great potential in healthcare and medical area. Specifically, there are some pioneer works that show successes of deep learning methods on ECG disease detection as pointed out in this survey [2]. Deep learning solutions such as CNNs, excel in recognizing the distinct ECG wave types and discern the complex relationships between them over time. This is a difficult task due to the variability in wave morphology between patients as well as the presence of noise. However, these methods are still far away from practical applications because none of these solutions have been deployed for providing publicly available ECG disease detection services. In fact most of them only highlight advancements in either the pre/postprocessing step adopted for the data, or the machine learning model architectures and accuracy [4]. Contrary to this trend, the work in [1] shifts the focus on building a publicly available out-of-the-box cloud deep learning service that can be used for cardiac disease detection from ECG. In this case the "cloud component" consists of the ability to invoke an API running on a cloud server in order to obtain a prediction response, but does not pay attention to or evaluate the scalability a cloud can offer. Other researches in the cloud computing field address data flow issues. Pandeya et al. [3] designed a scalable and economical cloud-based system for hosting ECG distribution services. It was designed to collect ECG data from mobile devices and then to transmit the data to remote servers for simple analyses. On a similar trend the aim of the article [5] is to address the issues regarding the usefulness of the ECG data collected from patients themselves through mobile devices. In our work we find a common ground between these research topics by working both on improving the machine learning pipeline and offering a way to deploy it in a cloud infrastructure as scalable and independent microservices.

1.3 Novel contribution

Our contribution in terms of software architecture is shown in Figure 1. The workflow is as follows. First of all, cardiologists connect to the website, hosted in our first microservice, and upload the ECG signal generated by their electrocardiograph. The signal is then forwarded to the second microservices, which run a neural network to infer the presence of noise in the signal. If noise is detected a result is made available to the corresponding user; vice versa, the third microservice continues the classification process of the signal in three possible arrhythmia classes, namely, "Atrial Fibrillation", "Normal Rhythm" and "Other Rhythm". Finally, the result will be available for the users.

2 Materials and Methods

2.1 ECG Dataset

We used the dataset in [4], consisting of 64,121 ECG records from 29,163 patients. The ECG data is sampled at a frequency of 200 Hz and is collected from a single-lead, noninvasive and continuous monitoring device called the Zio Patch. Each ECG record in the training set is 30 seconds long and can contain more than one rhythm type. Each record is annotated by a clinical ECG expert: the expert highlights segments of the signal and marks it as corresponding to one of the 4 rhythm classes. We then split the dataset in the two following partially overlapped subsets:

- Noise vs non-Noise data
- Atrial Fibrillation, Normal Rhythm and Other Rhythm

The first subset was heavily biased toward non noise class. To address this issue, we opted to balance out this dataset by injecting noise in the data. Finally, in order to assess the

performance of our models we evaluated them on a test set comprised of 336 additional records.

2.2 Machine Learning Models

The ECG arrhythmia detection task is a sequence-to-sequence task which takes as input an ECG signal $x = [x_1 \dots x_k]$ and outputs a sequence of labels $r = [r_1 \dots r_k]$, such that each r_i can take on one of m different rhythm classes. Each output label corresponds to a segment of the input. Together the output labels cover the full sequence. However, this phase is executed after a preliminary classification step, which aims to look for noise presence in the signal.

We use a convolutional neural network for both the binary and the sequence-to-sequence learning task. The networks take as input a time-series of raw ECG signal, and output a binary and a sequence of labels predictions, respectively. The models are structured as follows:

- 9 layers of convolution followed by a fully connected layer and a softmax
- 33 layers of convolution followed by a fully connected layer and a softmax

Note that the shallowness of the first model mirrors the simplicity of the corresponding classification task.

2.3 Cloud Technologies

When building our application, we were looking for a solution that could offer a reliable, flexible and robust system. We opted to use a Platform-as-a-Service (PaaS) as it offers a ready-to-use cost efficient solution with the caveat of having less fine tuned control. The PaaS of our choice is Google Cloud Platform, as it offers a plethora of services for developers. The most crucial one being Google Cloud Run, a serverless model that can run and automatically manage containers thanks to the under the hood Kubernetes layer. This setup perfectly fits our needs of an autoscaling and on-demand server infrastructure. Furthermore, Cloud Run can be configured to support multiple concurrent requests on a single container instance which allows us to save time and cost. We built three containers for our app through the usage of Docker. Each container holds a microservice which is then deployed to a GCR service. Microservices are an architectural style for developing applications: they allow a large application to be decomposed into independent constituent parts, with each part having its own realm of responsibility. This leads to simpler scalability and optimization, which is exactly what we are looking for. The microservices communicate with each other in a feed-forward fashion thanks to the pub/sub protocol in conjunction with the usage of Eventarc triggers. The aforementioned approach decouples clients from the server (they won't need to wait for the receiver to be available), and this separation is further enhanced by triggers' way of message delivery: the push method. Clients will receive messages as soon as they are ready, without the necessity of a synchronous wait. Figure 1 shows our platform's architecture. It is split in the three following microservices:

- M1 handles user inputs through a web interface and then waits for M2's or M3's output. Once the result is ready, it will be presented on the webpage, along with a plot of the ECG and the option to download the excel file for further analysis. During this process, a cardiologist enters his tax code and uploads the ECG file. This data is then published to the "ecg" topic, which will trigger M2's activation. Then, M1 will wait for the output to be uploaded on a bucket. M1 is set up with the following options:
  - 2 CPUs and 2 GBs of RAM
  - Number of maximum requests per container: 6
  - Allow all traffic
  - Authentication: Allow unauthenticated invocations
- M2 is activated every time M1 publishes to the "ecg" topic. This service hosts a CNN that's capable of detecting the presence of noise in the signal. If the uploaded ECG shows an irregularity, M2 uploads it to a bucket. Otherwise, M2 publishes this signal to another topic,
  "nonNoise-ecg", which triggers M3's activation. Finally, M2 starts waiting for a new ECG to be published on the "ecg" topic.
  - 4 CPUs and 4 GBs of RAM
  - Number of maximum requests per container: 3
  - Allow internal traffic and traffic from Cloud Load Balancing
  - Authentication: Require authentication
- M3 is waiting for a regular signal published by M2. Like M2, this service hosts a CNN capable of classifying the ECG among various types of arrhythmia classes; the result is then uploaded into the bucket. Finally, M3 starts waiting for a new ECG to be published on the "nonNoise-ecg" topic.
  - 4 CPUs and 4 GBs of RAM
  - Number of maximum requests per container: 3
  - Allow internal traffic and traffic from Cloud Load Balancing
  - Authentication: Require authentication

The microservices above all share the following configurations:

- CPU is only allocated during request processing.
- The number of minimum available instances is 2, while the maximum number is 100.

ECG data is stored inside a bucket whose access requires an authentication, further protecting the data from the outside. Data triggers are imposed in such a way that data migrates to a lower end colder bucket after 24 hours. We keep this data stored due to legal issues and also to periodically

improve neural networks performances with retraining as pointed out in 4.

3 Results

We now show how the cloud native performs under a large amount of workload. We created a script capable of simulating multiple users sending requests to the service. Specifically, we gradually scale the number of concurrent users in the range [1, 5000]; such upper bound was chosen as it is sufficiently appropriate for the context, in terms of expected load on a website. We opted to use the number of served requests per unit of time as a metric for benchmarking our application. Additionally, we monitored the following:

- The number of requests per second
- The number of allocated instances per container

These parameters are strictly related to each other: the number of allocated instances per container is directly proportional to the number of requests per second. This ensures, as expected, great scalability as the task is properly distributed between instances. Figure 3 also emphasizes the improvements of a microservice architecture over the traditional monolithic counterpart. This is due to each microservice scaling independently when needed. We can observe that the workload is hierarchically distributed between instances: M1 being flooded by HTTP requests and M3 being the less requested due to inherently fulfilling a more narrower set of requests. The trends shown in Figure 2 and Figure 3 prove our assertion.

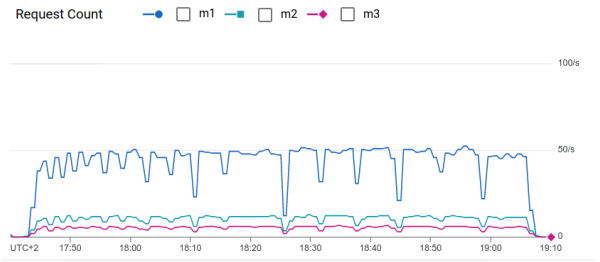

Figure 2. Request Count for Container

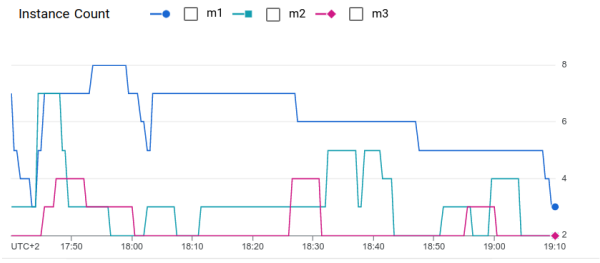

Figure 3. Instance Count for Container

We ran a similar benchmark on a local machine, namely the HPC cluster "PurpleJeans" 4, by adapting the code to a monolithical style to prove the importance of scalability in a cloud native environment. We also made sure that the hardware were somehow comparable and so we opted to use one CPU with 16 Xeon 5218 cores (operating at 2,3Ghz). The results of these benchmarks are shown in Figure 4 and Figure 5. We can confidently assess that the cloud benchmark's trend is downwards, albeit with some inconsistencies due to the creation of new instances over time; the local machine's benchmark, on the other hand, does exceptionally well at first. However, this trend tends to go upwards when the number of requests becomes too large. This happens due to lack of scalability in a local environment. Finally, we can observe that our cloud native solution is consistently faster than the same application running on a local machine.

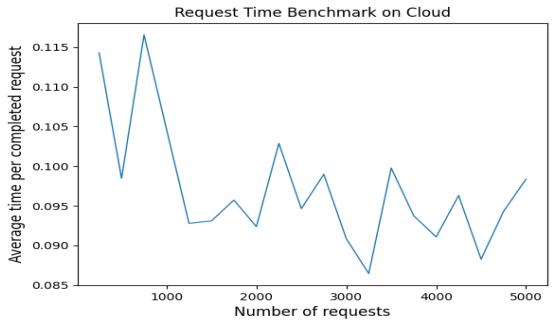

Figure 4. Cloud Benchmark

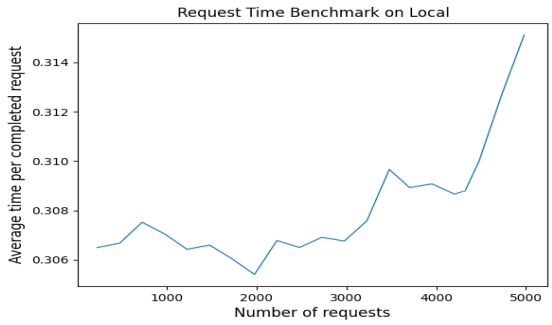

Figure 5. Local Benchmark

4 Conclusion

We develop a cloud native solution which detects a wide range of heart arrhythmias from single-lead ECG records. The main contribution is the design of a robust and scalable SaaS that can be accessed with ease by cardiologists, as a support for diagnosis. This also helps with digesting hospital queues more efficiently as doctors' load is alleviated. Future

works may concern adding a new microservice for cyclically retraining the neural networks as new data is stored into the buckets. We wish everyone can easily and early detect potential cardiac diseases anytime and anywhere.

Acknowledgments

We are grateful for the support of the University of Naples "Parthenope", Department of Science and Technologies, Research Computing Facilities (https://rcf.uniparthenope.it) for assistance with the calculations carried out in this work.

References

[1] Shenda Hong, Zhaoji Fu, Rongbo Zhou, Jie Yu, Yongkui Li, Kai Wang, and Guanlin Cheng. Cardiolearn: A cloud deep learning service for cardiac disease detection from electrocardiogram, 2020.

[2] Shenda Hong, Yuxi Zhou, Junyuan Shang, Cao Xiao, and Jimeng Sun. Opportunities and challenges of deep learning methods for electrocardiogram data: A systematic review, 2020.

[3] Suraj Pandey, William Voorsluys, Sheng Niu, Ahsan Khandoker, and Rajkumar Buyya. An autonomic cloud environment for hosting ecg data analysis services. *Future Generation Computer Systems*, 28(1):147–154, 2012.

[4] Pranav Rajpurkar, Awni Y. Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y. Ng. Cardiologist-level arrhythmia detection with convolutional neural networks, 2017.

[5] Henian Xia, Irfan Asif, and Xiaopeng Zhao. Cloud-ecg for real time ecg monitoring and analysis. *Computer Methods and Programs in Biomedicine*, 110(3):253–259, 2013.