Serverless architecture for data processing and detecting anomalies with the Mars Express MARSIS instrument

Pacios, D.; Vazquez-Poletti, J.L.; Sánchez-Cano, B.; Moreno-Vozmediano, R.; Schetakis, N.; Vazquez, L.; Titov, D.V.

The Astronomical Journal, 166:19 (11pp), 2023 July https://doi.org/10.3847/1538-3881/acd18d

Serverless Architecture for Data Processing and Detecting Anomalies with the Mars Express MARSIS Instrument

David Pacios[1], José Luis Vazquez-Poletti[1], Beatriz Sánchez-Cano[2], Rafael Moreno-Vozmediano[1], Nikolaos Schetakis[3,4], Luis Vazquez[1], and Dmitrij V. Titov[5]
[1] Universidad Complutense de Madrid, Facultad de Informática, C/ Profesor José García Santesmases 9, 28040 Madrid, Spain; jlvazquez@fdi.ucm.es
[2] The University of Leicester, University Road, Leicester, LE1 7RH, United Kingdom
[3] Quantum Innovation I.K.E., Aristotelous 84, Chania 73100, Greece
[4] Alma Sistemi Srl, Via Vittorio Iovino, 1, 80047 San Giuseppe NA, Italy
[5] University of Leiden, Niels Bohrweg 2, 2333 CA Leiden, The Netherlands
Received 2022 December 20; revised 2023 April 3; accepted 2023 April 27; published 2023 June 16

Abstract

The Mars Advanced Radar for Subsurface and Ionosphere Sounding (MARSIS) on board Mars Express has been sampling the topside ionosphere of Mars since mid-2005. The analysis of the main reflection (nadir) of the ionosphere through the ionograms provided by the MARSIS instrument is typically performed manually due to the high noise level in the lower frequencies. This task, which involves pattern recognition, turns out to be unfeasible for the >2 million ionograms available at the European Planetary Science Archive. In the present contribution, we propose a modular architecture based on serverless computing (a paradigm that stands on the cloud) for optimal processing of these ionograms. In particular, we apply serverless computing to detect oblique echoes in the ionosphere, which are nonnadir reflections produced when MARSIS is sounding regions above or nearby crustal magnetic fields, where the ionosphere loses the spherical symmetry. Oblique echoes are typically observed at similar frequencies to the nadir reflections but at different times delays, sometimes even overlaying the nadir reflection. Oblique echoes are difficult to analyze with the standard technique due to their nonconstant and highly variable appearance, but they harbor essential information on the state of the ionosphere over magnetized regions. In this work we compare the proposed serverless architecture with two local alternatives while processing a representative data subset and finally provide a study by means of cost and performance.

Unified Astronomy Thesaurus concepts: Mars (1007); Planetary ionospheres (2185)

1. Introduction

The Mars Advanced Radar for Subsurface and Ionosphere Sounding (MARSIS) on board Mars Express has been sounding the subsurface, surface, and ionosphere of Mars since mid-2005 (Orosei et al. 2015). The radar has two different operational modes, one dedicated to sound the surface and below with two simultaneous frequencies to choose between 1.8, 3, 4, and 5 MHz, and another one called Active Ionospheric Sounding (AIS) where the radar transmits a sweep of 250 frequencies between 0.1 and 5.5 MHz (Gurnett et al. 2005). In this mode, the ionosphere is sounded from the spacecraft location to the maximum ionization region, and the observations are recorded in the form of ionograms, which are plots of the echo reflections of each frequency plotted in the form of time of flight of the signal versus its frequency. These ionograms contain a large amount of information regarding the ionosphere, such as the main trace of the ionosphere formed after the different frequencies are reflected in the ionosphere when the carrier frequency is equal to the plasma frequency. This is a nadir reflection from which the topside electron density profile from the spacecraft altitude until the region of maximum ionization (typical at ∼130 km at the subsolar point; Sánchez-Cano et al. 2013) can be obtained. In addition, the local plasma electron density and magnetic field (magnitude) surrounding the spacecraft (Gurnett et al. 2008; Andrews et al. 2013) can also be obtained, as well as the surface reflection for frequencies larger than the plasma frequency and, sometimes, oblique echoes that come from nonnadir reflections (Andrews et al. 2014). These oblique echoes are produced when MARSIS-AIS sounds regions above or nearby crustal magnetic fields, where the ionosphere loses the spherical symmetry. These regions are mainly found in the southern hemisphere of Mars (Langlais et al. 2019), where the ionosphere is locally lifted breaking the general spherical symmetry. Oblique echoes are not always found in all ionograms, and when they are, they are typically associated with vertical crustal magnetic fields (Andrews et al. 2014). Despite many studies, it is still not clear whether these reflections are produced by high-density regions or by ionospheric upwelling (Němec et al. 2015; Fallows et al. 2019). However, they provide a unique view of the nonuniformity of the ionosphere and of the role of the crustal fields lifting or depressing plasma within local regions of the ionosphere. Their systematic study, therefore, has the potential to provide the best characterization of the lifetime of these ionospheric structures and their variability with local time and seasons.

MARSIS-AIS constitutes a rich and large data set for ionospheric studies that have revolutionized our knowledge of Mars' ionosphere in the 18 yr of continuous observations as it was the first instrument to systematically sample the topside ionosphere of Mars at all latitudes, longitudes, and local times. A summary of the main discoveries of this instrument can be found at Orosei et al. (2015). However, despite the large amount of ionospheric parameters that can be retrieved, some of them are easier to obtain than others. For example, a

parameter than can be easily retrieved with an algorithm is the peak density of the ionosphere (critical frequency; Withers et al. 2015), which is the last reflection in the main ionospheric trace. However, the analysis of the main trace of the ionosphere (main reflection) is typically done manually. This is because, although an automatic detection is possible for frequencies above ∼1.5 MHz, the lower frequencies are affected by a myriad of noise issues, including antenna optimization issues, and significant overlaying of different signatures, such as local plasma harmonics and electron cyclotron echoes near the spacecraft. For this reason, MARSIS-AIS ionograms are typically processed manually (Sánchez-Cano et al. 2012). Another example are the surface reflection echoes that can be easily identified and retrieved via analysis of the power signal reflection at the higher frequencies on the AIS ionograms (Němec et al. 2015), or their nonreflections that indicate the level of absorption present in the lower ionosphere (i.e., radio blackouts; Sánchez-Cano et al. 2019; Lester et al. 2022). Since the radar deployment in mid-2005, MARSIS has been in operation for more than 20,000 orbits. This is, therefore, a huge data set with a lot of scientific possibilities, for which alternative methods to the standard method could be very powerful.

In this study, we explore the possibility of using cloud computing in the form of serverless computing with MARSIS-AIS in order to estimate the feasibility of this technique for planetary science. This technique offers a large variety of advances, such as a quick computing process time, and quick identification of signatures in large data sets.

Cloud computing platforms, and especially IaaS (Infrastructure as a Service) clouds, have been used for years to deploy different kinds of scientific applications and scientific workflows (Malawski et al. 2012; Panneerselvam & Subbaraman 2018; Ahmad et al. 2021; Rajasekar & Palanichamy 2021). The IaaS cloud model enables the on-demand provision of processing, storage, networks, and other fundamental computing resources on a pay-per-use basis, offering the flexibility to scale IT resources up and down on demand. This model allows individual users and organizations to avoid the cost and complexity of purchasing, managing, and maintaining physical servers and data center infrastructures.

One of the main challenges when deploying a scientific workflow on an IaaS cloud, such as the one addressed in this paper, is to optimize the workflow scheduling, which can be divided into two different subproblems (Rodriguez & Buyya 2017): resource provisioning and task allocation. Resource provisioning consists of selecting and instantiating the cloud resources that will be used to run the workflow tasks (e.g., virtual machines or containers, storage, interconnection networks), trying to select the types of resources that are best suited to different tasks, and trying to optimize the amount of resources deployed to reduce the cost of the infrastructure. Task allocation consists of mapping each task onto the best-suited resource, trying to maximize the resource utilization and/or reduce the task execution time and the total make-span. This is an NP-complete problem, and many different heuristics and metaheuristics has been proposed to solve it (Houssein et al. 2021; Belgacem & Beghdad Bey 2022).

More recently, a new cloud computing service model has been delivered, the serverless model (Castro et al. 2019; Jonas et al. 2019), which aims to abstract infrastructure management from final users and application developers. In this model, the provider is responsible for allocating, deploying, and scaling the resources required to meet the needs of the user's applications. This model is very useful for the deployment and execution of scientific workflows as resource provisioning and tasks scheduling issues are the responsibility of the provider, while developers only have to worry about providing the application source code in the form of atomic functions (Vazquez-Poletti & Llorente 2018), and they are billed only for the time the code is running.

In this paper we propose a modular architecture based on serverless computing for optimal processing of MARSIS-AIS ionograms. In order to evaluate the feasibility of this method, we have selected oblique echoes as the key signatures to be detected. This is mainly because these features are not always present in the ionograms and when they are, they appear in multiple complex forms. Oblique echoes are typically observed at similar frequencies to the nadir reflections but at different times delays, sometimes even overlaying the nadir reflection. They are difficult to analyze with the standard technique due to their nonconstant and highly variable appearance but they harbor essential information on the state of the ionosphere over magnetized regions. Therefore, they are considered are a good type of signature with large scientific potential that can be easily detected with cloud computing. Although in this study we use serverless cloud computing for MARSIS-AIS oblique echoes, we note that this technique can be applied to identify other signatures in the images, such as the main trace of the ionospheric reflection, or the surface of the planet, or even be applied to other data sets.

We will demonstrate that the proposed serverless-based architecture will significantly reduce the processing time compared to other classic server-based solutions, with a reduced infrastructure cost.

2. MARSIS-AIS Data Set

We use the MARSIS-AIS data set in this work to analyze two years of ionograms focusing on the detection of oblique echoes. For that, we use the summary plots (i.e., images) that can be found on the European Space Agency (ESA) Planetary Science Archive (PSA) rather than the actual data files. The main reason to use the summary plots are that the actual data are stored in complex binary files that need some preprocessing before using them for scientific purposes, and these browse images are easier to manage, have less data volume, and so, are ideal for the cloud computing technique. All images are in .PNG format and were processed by University of Iowa (US) until 2019 December and by University of Leicester (UK) since then. All of them have the same color bar scale measuring the power return of the signal in $V^2\,m^{-2}\,Hz^{-1}$. In total, there are >2 million ionograms publicly available at the PSA. We choose a subset of data from 2013 and 2014 to run our experiment. This comprises 441,919 images, which correspond to 120 GB in size.

In order to use cloud computing over the MARSIS-AIS ionogram images, we need to define the main feature that we would like to identify. In this study, we use the ionograms images already present in the BROWSER folders in the ESA PSA. Figure 1 shows an example of the images used in this work, where the main features are highlighted. In particular, we use data on the dayside of Mars where a clear reflection from the ionosphere is observed. In Figure 1, the main ionospheric reflection is seen between frequencies 2 and 3.5 MHz and time delays of 1.4 and 1.8 ms. In addition, other features are present,

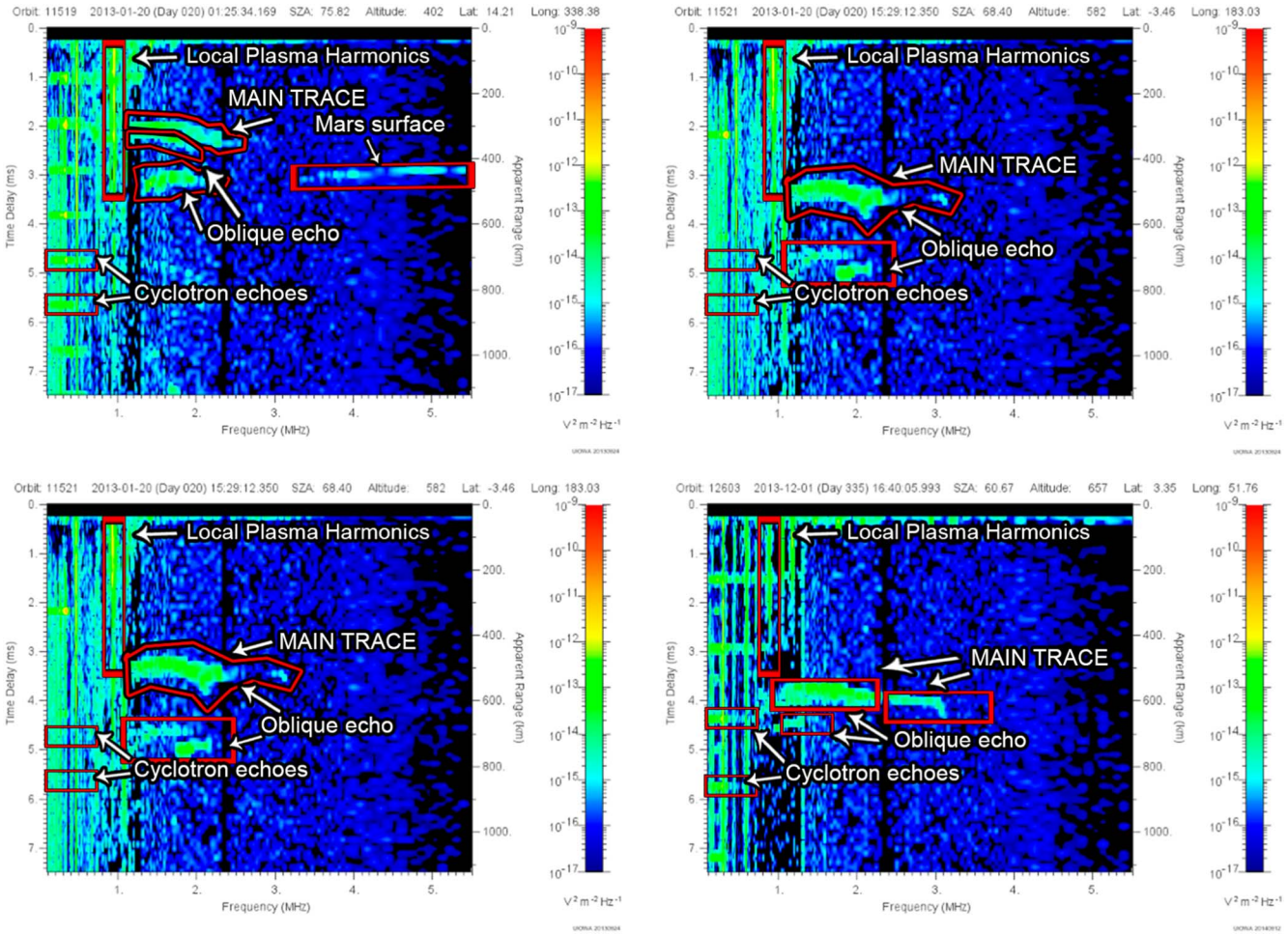

Figure 1. Examples of MARSIS-AIS ionograms with different visible features, such as oblique echoes, local plasma harmonics, Mars surface lines, and cyclotron echoes. In the bottom right and top right ionograms shown, the main trace and oblique echo are superimposed.

such as the local harmonics of plasma from which the local electron density around the spacecraft can be retrieved (vertical lines at frequencies lower than ∼1.5 MHz); electron cyclotron echoes from which the magnitude of the magnetic field surrounding the spacecraft can be obtained (horizontal lines at frequencies lower than ∼1 MHz); the surface reflection (straight line at frequencies higher than 4.5 MHz); and the oblique echoes (curve reflection between frequencies ∼1.2 and ∼2.6 MHz and time delays of 2 and 3 ms). In order to study the feasibility of cloud computing on this data set, we use the oblique reflections as a case study.

3. Computational Solution

3.1. Cloud Computing and Serverless for Data Science

Serverless computing is a cloud computing model that abstracts the underlying infrastructure, operating system, and server management from developers. In a serverless environment, the application logic is commonly implemented as a set of stateless functions that are triggered by events (e.g., API calls, message queues or scheduled tasks), and are executed by containerized or microVM based run time environments.

The main benefit of the serverless model (Baldini et al. 2017; Eivy & Weinman 2017) is that developers can focus solely on writing code and deploying their applications, without worrying about the infrastructure or scaling requirements. The cloud provider takes care of the heavy lifting of server management, capacity planning, and scaling, allowing developers to build and deploy applications more quickly and efficiently. In addition, serverless computing reduces the cost of computing by charging only for the resources consumed during the execution of the function.

Platforms implementing this serverless model are categorized as Function as a Service (FaaS). FaaS platforms enables developers to upload their code to the cloud provider, define the trigger that will activate it, and specify the resources required to run the function. When a trigger occurs, the cloud provider automatically spins up the required resources, runs the code, and then releases the resources when the function has completed execution.

Some examples of commercial FaaS platforms are Amazon AWS Lambda,[6] Google Cloud Functions,[7] and Microsoft Azure Functions.[8] There are also some open source serverless initiatives, such as Apache OpenWhisk[9] (Quevedo et al. 2019), OpenLambda[10] (Hendrickson et al. 2016), and Knative.[11]

[6] https://aws.amazon.com/lambda
[7] https://cloud.google.com/functions
[8] https://azure.microsoft.com/services/functions
[9] https://openwhisk.apache.org
[10] https://github.com/open-lambda/open-lambda
[11] https://knative.dev

Serverless computing or FaaS platforms are suitable for the deployment and execution of scientific workloads (John et al. 2019; Burkat et al. 2021; Eismann et al. 2020; Malawski et al. 2020) for two main reasons. On the first hand, serverless computing has revealed itself as a valuable paradigm for complex high-throughput applications environments that demand optimal resource provision (Raman et al. 1998). This is because dynamic load peaks can be effectively attended via automatic resource management. With serverless, the cloud provider automatically handles the scaling of resources based on the needs of the application. This can lead to improved performance and availability for the end user. On the second hand, serverless platforms offer the possibility of function chaining (Daw et al. 2020; Balla et al. 2021; Sabbioni et al. 2021), where the completion of a function can trigger the invocation of another function. This allows for the creation of complex workflows by linking together multiple simple functions. For example, each task in the workflow can be implemented by a different function, and when a task is completed, its output is placed in an intermediate storage that triggers the execution of the next function. In this work, we propose the use of this serverless function chaining scenario to deploy our scientific application in an efficient way.

In particular, we have focused on Amazon Web Services (AWS), one of the leading public cloud providers, which pioneered the provision of serverless computing services. Its FaaS Lambda[12] has been used in our paper because it allows an easy code execution without managing infrastructure elements while defining events that trigger the functions (Villamizar et al. 2016).

AWS Lambda operates by executing each of its functions within a dedicated container, created when the function is first created. These containers are then executed on a multitenant cluster of machines managed by AWS. To ensure that each function is allocated the necessary resources, each container is allocated the required RAM and CPU capacity before execution. When a function completes, the RAM allocation is multiplied by the function's run time, and customers are charged accordingly based on the allocated memory and run time. AWS Lambda allows many instances of the same or different functions to run concurrently, making it possible to deploy highly scalable applications. To implement function chaining in AWS Lambda, developers can use a variety of methods, including AWS Step Functions, AWS EventBridge, or the Simple Storage System (S3), among others. The scientific workflow implemented in this work is based on S3 buckets that are configured to trigger a lambda function when new objects are added or updated, so that a sequence of lambda functions are executed, with each function processing the data in a specific way before passing it on to the next function in the workflow.

The total function running time in AWS Lambda comprises the execution of the code itself and the initialization performed by Lambda (Fouladi et al. 2017). The code is executed inside a container that is deployed to this end. This container will be reused if the function is triggered again in the next 15 minutes, reducing the initialization time (Vazquez-Poletti & Llorente 2018).

[12] https://aws.amazon.com/es/lambda/

3.2. Serverless Architecture

In order to understand the proposed architecture, its building blocks are explained in depth:

1. S3 buckets are a type of cloud storage system with some special characteristics. In particular, we take advantage of the triggering produced by new files appearing in the bucket. This way, each new MARSIS ionogram uploaded to S3 will force the execution of an operation we have previously defined on it. Multiple triggerings will produce multiple executions that will run in parallel.
2. A lambda function is a cloud computing component that allows to execute a certain code (different languages are supported[13]) in a set of given conditions (execution deadline and maximum memory). As it can be deduced from the previous S3 explanation, lambda functions are invoked through defined triggers. The main advantage is its high level of parallelism, allowing up to 1000 instances of the same function to be executed at the same time. In our study, we conducted tests on the architecture with a limit of 1000 parallelization functions; however, it is worth noting that a significantly higher number of simultaneous executions can be requested on-demand through Amazon Web Services (AWS).

There are different programming languages for developing lambda functions. In the present work, Python was chosen for its versatility when performing mathematical calculations and working with advanced color patterns. We have been using both 3.8 and 3.9 run times because not all lambda layers work with the same version.

Additionally, lambda developers do not interact with any operating system. However, the function execution takes place on a container running Amazon Linux.

The proposed architecture is decomposed in several chained lambda functions that will use S3 as swap memory, as the output files of a function will trigger one or more other functions. These functions and buckets are organized in modules, as it can be seen in Figure 2.

The first module (`COMMON PRE-PROCESSING`) is common to all MARSIS input files, and it prepares the data to be processed by the following modules.

The next classification module (`CUSTOM MEASURE MODULE`) launches simultaneously the initial functions of different detection modules and on the same input data. In this work we have focused on the `GREEN DETECTION` function, which moves the process to the module for anomaly in the green spectrum detection, and finally to the module for the oblique echoes detection. Returning to the `CUSTOM MEASURE MODULE`, the first function of additional detection algorithms besides the oblique echoes one would be placed here.

The final result of the proposed architecture are the ionograms classified in different folders on an S3 bucket, according to what has been detected on them.

3.2.1. Common Preprocessing Module

MARSIS ionograms are uploaded to the `S3_INIT` input bucket. Each file upload triggers the `INIT_MODULE_MARS` function, which classifies the ionogram according to the latitude, longitude, and solar zenith angle. This process is aided by a

[13] https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html

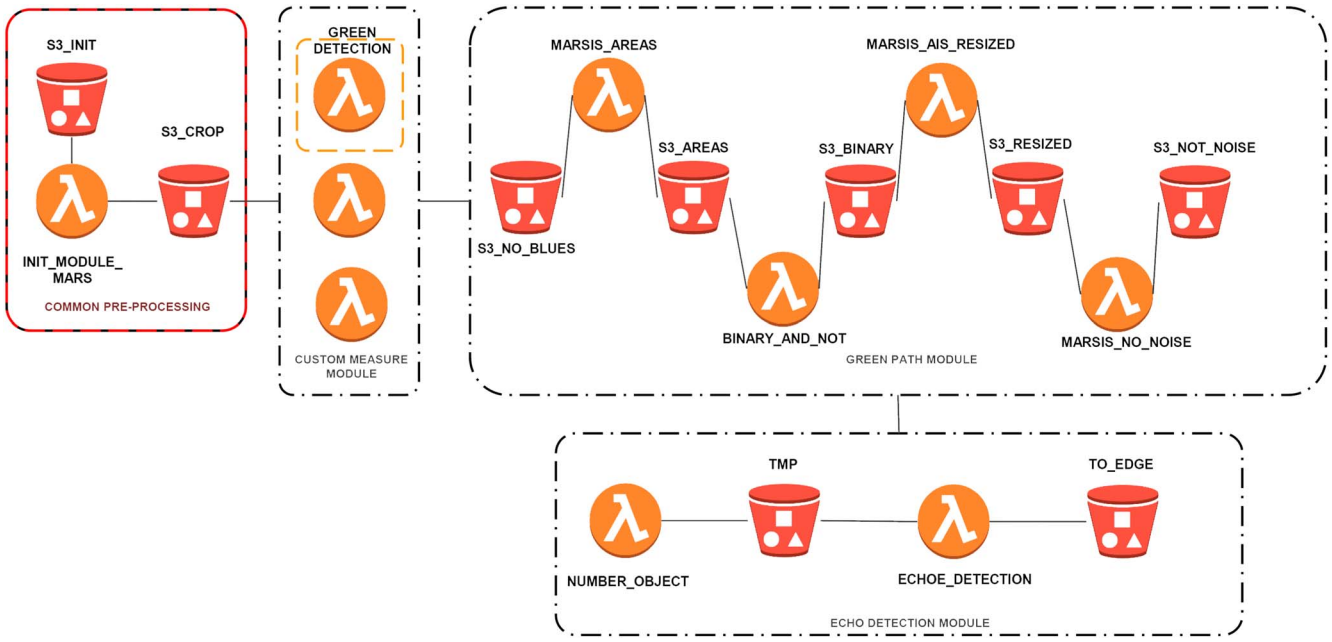

Figure 2. Serverless modular distributed architecture for anomalies detection in MARSIS ionograms. This architecture has two common modules: PRE-PROCESSING and MEASURE (from where the specific oblique echo detection modules are called).

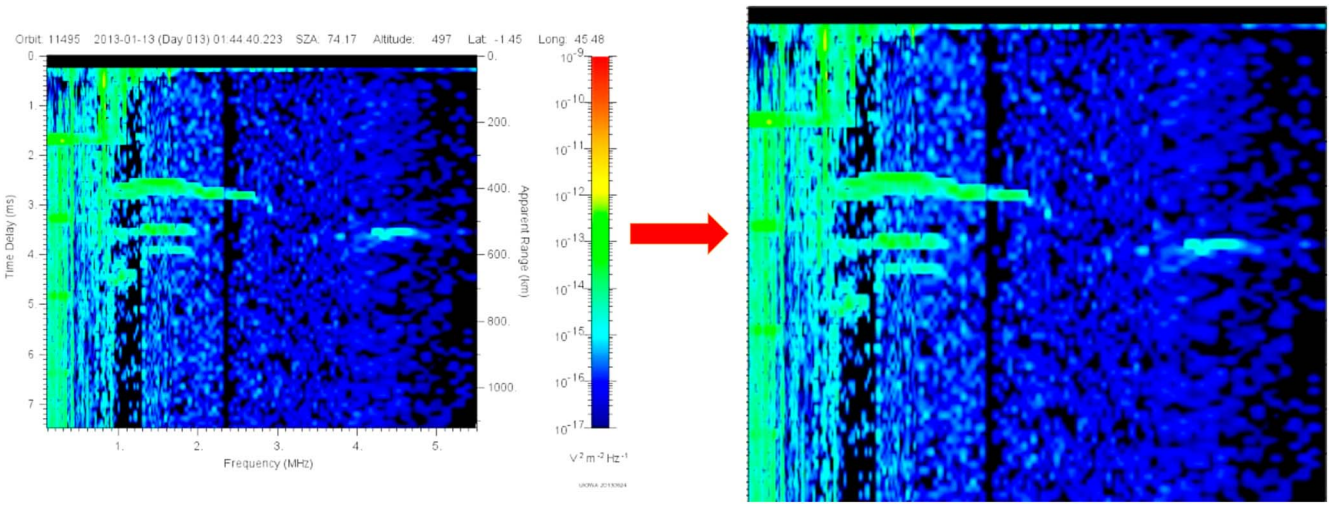

Figure 3. Original ionogram (left) and cropped image where the color area is isolated (right).

custom layer named `Tesseract` that parses the previous mentioned parameters directly from the image (as they are written on it).

Once the images are classified, their white borders are removed. This way each image is properly centered and deposited in the `S3_CROP` bucket (see the example shown in Figure 3). This action triggers the next modules.

As explained before, in this contribution we have focused on the measured module (`CUSTOM MEASURE MODULE`).

3.2.2. Custom Measure Module

Upon detecting a new file in the `S3_CROP` bucket, different functions are simultaneously triggered (GREEN DETECTION, BLUE DETECTION, HARMONICS DETECTION). The present work covers the `GREEN DETECTION` function, which removes the blue spectrum in the ionogram. This procedure isolates its green zones using color masks converts the color space of the image from the RGB color space to the HSV color space using the `cv2.inRange` function; this range falls between 60° and 120° on the HSV color space. This function returns a binary mask where pixels that fall within the specified range of colors are set to white, while the rest are set to black.

Then the `cv2.bitwise_and` function applies the mask to the original image, resulting in a new one where only the pixels that fall within the specified color range are visible, resulting in a cleaner version that is stored in the `S3_NO_BLUES` bucket (see the example shown in Figure 4). This triggers the `GREEN PATH MODULE`.

3.2.3. Green Path Module

This module is the most complex in the architecture, because it involves several functions to prepare the ionogram for oblique echoe detection.

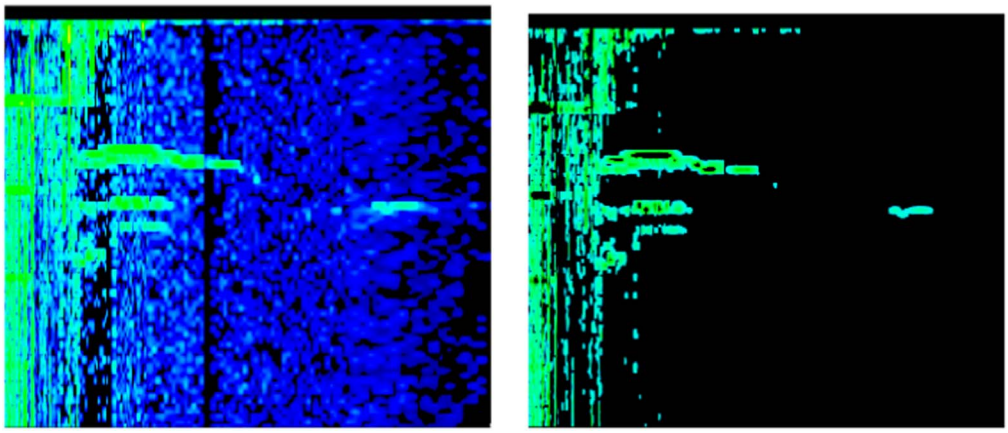

Figure 4. Cropped ionogram from the previous step (left) and result of processing with no blues using the GREEN DETECTION function (right).

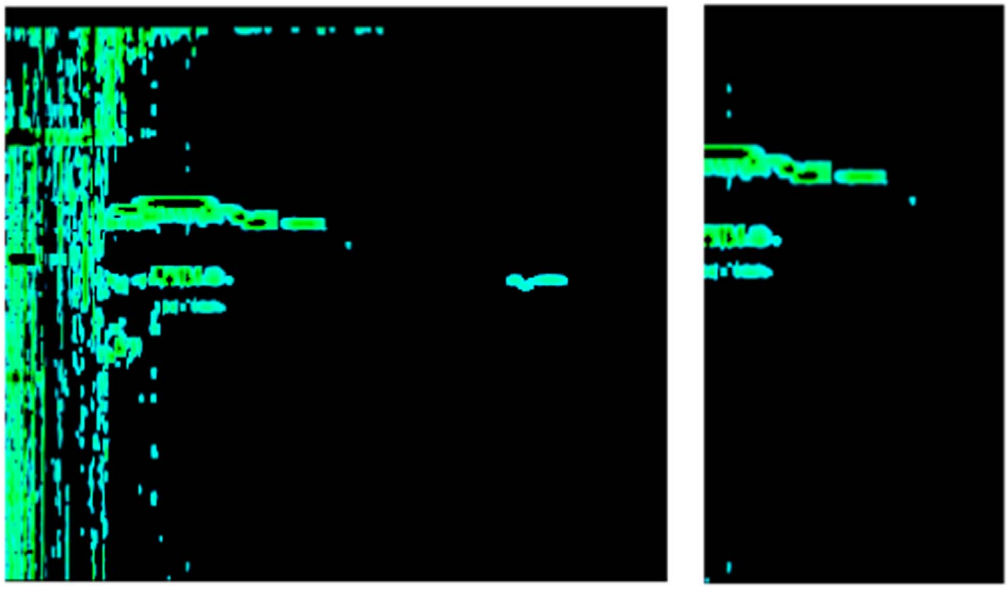

Figure 5. Green ionogram (left) and result cropped according to the parameters passed to the MARSIS_AREAS function (right).

The first function of this module is MARSIS_AREAS, which works with a dictionary that provides possible detection locations within the image. This way the ionogram is cropped depending on the frequency or area where potential anomalies should arise. For the specific case of detecting oblique echoes, we perform an image cropping in the frequency range of 2.5–5.5 MHz to isolate the areas where these phenomena may occur. This technique is commonly used in ultrasound imaging to enhance the visibility of oblique structures and improve diagnostic accuracy. By selectively removing unwanted noise and artifacts from the image, the cropped frequency range can provide a clearer and more detailed representation of the targeted area of interest. The result is stored in the S3_AREAS bucket (see the example shown in Figure 5), which triggers the BINARY_AND_NOT function.

Finally, the resulting image is written to a file at a given path using OpenCV's imwrite function. The resulting image is then saved to an S3 container.

In this particular case, the purpose of this image processing is to isolate the green areas of the image and filter out the blue color values. This is intended to identify the anomalies, which may appear in green.

The process then focuses in identifying the relative position of the objects in the figure. This is accomplished by the BINARY_AND_NOT function, which produces a binary image with black background and white objects, followed by a color invertion (white background and black objects) and file upload to S3_BINARY.

The next function triggered is MARSIS_AIS_RESIZED, which isolates the image according to the 800×600 px standard adopted by ESA in this public database. The function rescales the image, leaving the detection area in the middle. This prevents each loose pixel from being treated as a separate object from the whole image. The result is loaded to S3_RESIZED (see the example shown in Figure 6), triggering the MARSIS_NO_NOISE function.

This function removes the abovementioned loose pixels by detecting their proximity with lines. It uses OpenCV's imread function at the beginning and then converts the color space of the image from RGB to grayscale using the cvtColor function.

Then it applies a binary threshold to the grayscale image using OpenCV's threshold function. This creates a binary image where pixels with intensity values greater than 127 are

Figure 6. Full processing, original cropped ionogram (left) with binary mode (middle), and resized image using ESA 800 × 600 px standard (right). As it can be seen in the binary mode image, the loose pixels have been removed, producing shapes without gaps. In the image, in a red box, the red areas that disappear with the OpenCV operation have been marked. It is done to avoid detecting several objects through those "holes."

set to 255 (white), and pixels with intensity values less than or equal to 127 are set to 0 (black).

Next, the function finds contours in the binary image using OpenCV's `findContours` function. The contours are retrieved using the `RETR_TREE` mode, which retrieves all the hierarchical contours, and the `CHAIN_APPROX_SIMPLE` method, which compresses horizontal, vertical, and diagonal segments and leaves only their end points.

Then, a mask with the same shape as the original image is created using NumPy's zeros function. The code then loops through the contours and checks if they have no child contours and if their area is less than 70. If these conditions are met, the contour is drawn on the mask with a white color.

Finally, OpenCV's `bitwise_not` function is used to invert the colors of the original image using the mask. The resulting image is then written to a file at a given path using OpenCV's `imwrite` function. The resulting image is then saved to an S3 container.

In this particular case, the purpose of this image processing is to detect and isolate small objects in the image that have an area less than 70 pixels. The detected objects are then converted to a mask and used to remove those objects from the original image. The result of this last function is stored in the `S3_NOT_NOISE` bucket, triggering the `ECHO DETECTION MODULE`.

3.2.4. Echo Detection Module

The primary objective of this module is to identify the number of objects and their relative position. At least two of these objects arranged in parallel are needed for detecting oblique echoes, as it will be explained later.

The process starts with the `NUMER_OBJECT` function, which counts the objects in the image. It first reads the ionogram and converts it to grayscale using the `cv2.cvtColor()` function. Then, it applies a threshold to the grayscale image using the `cv2.threshold()` function, with a minimum threshold value of 225, a maximum threshold value of 255, and an inverted binary threshold type (cv2.THRESH_BINARY_INV).

Next, it creates a 2 × 2 kernel using `np.ones()` function and performs dilation on the thresholded image using the `cv2.dilate()` function with the created kernel and 10 iterations. The resulting image is used to find contours using `cv2.findContours()` function with an external retrieval mode (cv2.RETR_EXTERNAL) and simple approximation method (cv2.CHAIN_APPROX_SIMPLE).

As said before, oblique echo detection needs at least two objects: an original and a replica underneath. When these (or more) are detected by the architecture, the image is loaded into the `TMP` bucket, which triggers the `ECHO_DETECTION` function. This function takes the relative positions of the objects into account and, if the detection is positive, it stores the original image along with the detection proof (see the example shown in Figure 7) in the final bucket (`TO_EDGE`).

4. Experiments and Results

The chosen MARSIS ionogram data set (AIS mode) for our experiments corresponds to 2013 and 2014. This data set contains 441,919 ionograms and is 120 GB in size.

4.1. Experimental Environment

We have considered three different computing platforms to run our experiments:

1. *Computer_1*: an Intel Core i9 10980XE (18 cores, 36 threads) computer with 64 GB RAM and SSD (M2) hard disks. Its estimated price is €3500.
2. *Computer_2*: an Intel Core i7 vPro i7-10875H (8 cores, 16 threads) computer with 64 GB RAM and SSD (M2) hard disks. Its estimated price is €3250.
3. *Serverless*: the AWS Lambda system described in this contribution with a 128 MB memory limit.

In order to compare these three platforms, the code of the serverless architecture shown in Figure 2 has been adapted to run locally in Computer_1 and Computer_2. During the initial experiments we considered the following:

1. An execution deadline of 1 hr was established for all experiments, so local resources were not busy for too long. Due to the low time variations in the same

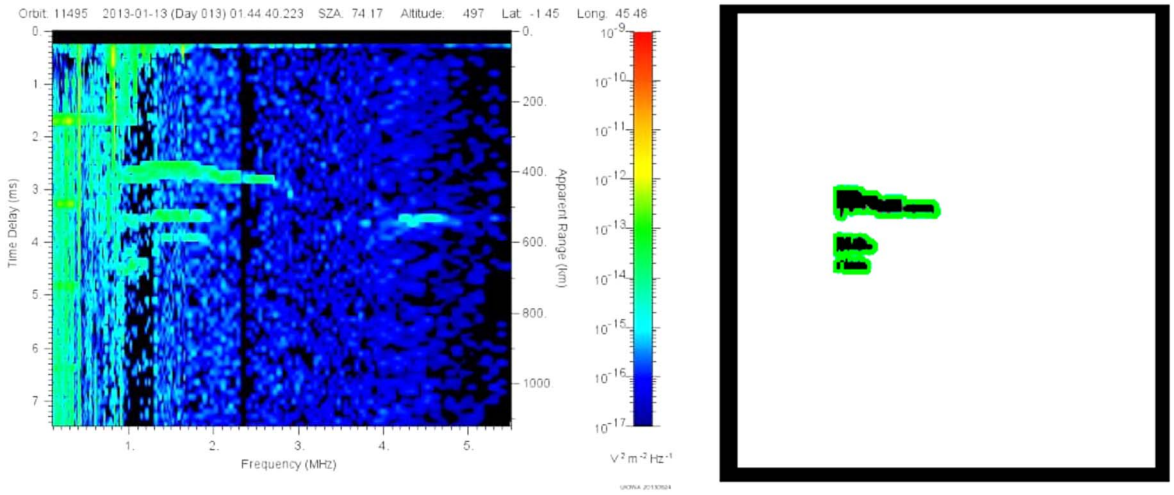

Figure 7. Original ionogram with an oblique echo (left) and its final detection proof, produced by the ECHO DETECTION MODULE (right).

execution, this deadline was considered enough for result extrapolation.

2. Due to the restrictions imposed by the *Tesseract* custom layer each execution will have to take place in a separate environment and assigned to an available processor core.

The reason for the last consideration is that text recognition may not be accurate, especially when the image quality is poor. Therefore, it is crucial to optimize the image before using this layer to enhance the recognition accuracy.

Using the *Tesseract* in a Python function requires an additional compiled layer. Unfortunately, the size limit of the lambda function layers is 250 MB, which can be a problem when more layers are added. This limitation prevents the inclusion of other required dependencies and resources in the same function, which can impact the overall performance (Section 3.2.1).

4.2. Platform Comparison

The serverless solution processed the entire data set in 20 minutes, Computer_1 managed to process 4320 (0.009%) and Computer_2 1440 ionograms (0.003%). With this in mind, Table 1 shows the execution time for the data set on each considered platform.

At this point it is important to indicate that using the *Tesseract* in Python has been deemed as a complex undertaking, particularly when attempting local execution. Despite the creation of scripts aiming to automate the process, manual execution remains a necessity. Throughout the testing phase, maintaining consistent code has been a priority in order to ensure accurate comparisons.

Parallelization on a desktop environment presents its own set of challenges, as it would need a distinct code base from that utilized in AWS Lambda. Moreover, threading in Python is a notoriously intricate and error-prone endeavor. While alternative solutions may be achievable through the employment of various tools in the C programming language, the advantages of cloud-based parallelization would be undeniable.

By leveraging the power of cloud computing, the achieved parallelization far surpasses what would be possible using all available threads on a local machine. As a result, the decision has been made to compare the same unmodified code in both environments, avoiding the falsification of outcomes and ensuring a fair comparison between the two distinct implementations.

Table 1
Process Time for the Entire Data Set (441, 919 Ionograms) on each Platform

Platform	Processing Time
Serverless	20 minutes
Computer_1	4 days, 6 hr, 18 minutes
Computer_2	12 days, 18 hr, 53 minutes

Table 2
Average Execution Times for each AWS Lambda Function when Processing 1000 Ionograms

Lambda Function	Time (ms per 1000 ionograms)
INIT_MODULE_MARS	482.33
GREEN DETECTION	454.59
MARSIS_AREAS	499.94
BINARY_AND_NOT	353.31
MARSIS_AIS_RESIZED	542.21
MARSIS_NO_NOISE	1.377
NUMBER_OBJECT	810.24
ECHO_DETECTION	767.17

4.3. Serverless Architecture Experimental Results

The main advantage of the AWS Lambda approach is its extreme parallelization capability, which clearly exceeds those of the local resources that were used during the experiments. This added to its autoscaling mechanisms result in the performance and cost numbers shown below.

Data for each function's performance have been gathered through the Amazon CloudWatch tool.[14] These data include execution times and costs. Table 2 shows the average execution times for each AWS Lambda function depicted in Figure 2.

Considering that AWS limits the simultaneous execution of lambda functions to 1000, we have estimated 3.9 s as the average execution time for each batch of 1000 ionograms.

The AWS Lambda cost is determined by the memory used by functions (128 MB in the proposed architecture), the maximum execution time per function (5 s in the proposed

[14] https://aws.amazon.com/cloudwatch/

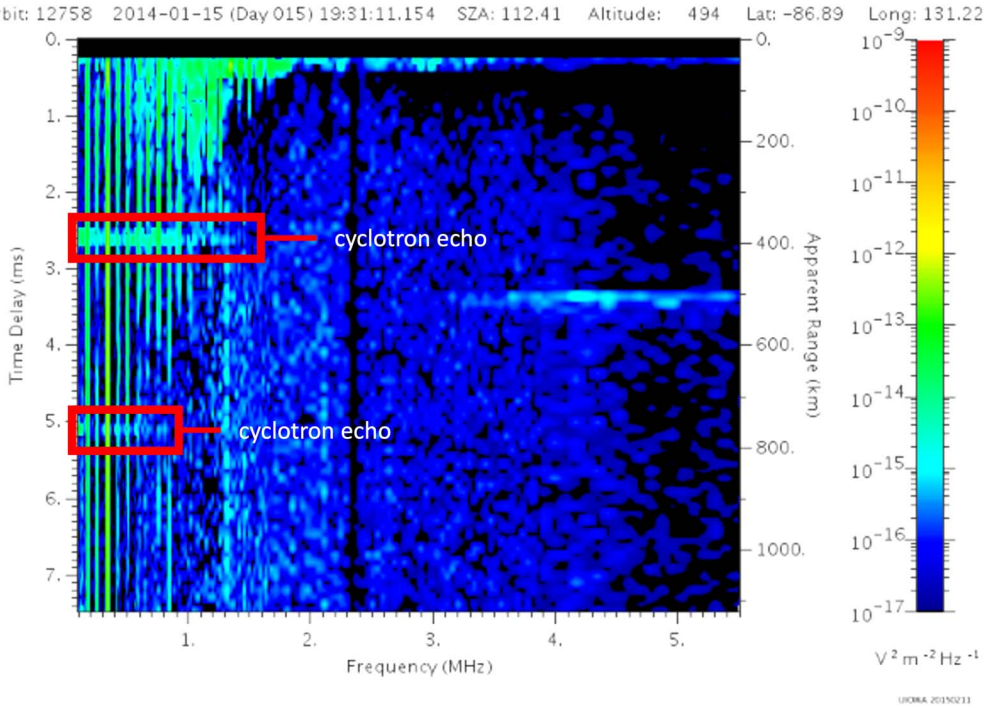

Figure 8. Cyclotron detection with similar oblique echo pattern. This pattern was observed in 16.66% of the detections from the chosen data set.

architecture), and the number of function calls. This results in an average of $0.000000084 per processed GB.

On the other hand, the S3 cost is subject to the stored data size. However, the proposed solution does not store intermediate data (just input and final output files). We would like to highlight that we are generating data in each intermediate S3 bucket. However, we have implemented a mechanism in each lambda function to delete all intermediate files once they have been processed. Therefore, no intermediate data are stored in the system as they are deleted automatically. This limits the S3 cost to $0.00000023. This way, the total cost of processing the 441,919 ionograms is $0.00001.

In order to provide a fair and accurate comparison, tests were conducted using exactly the same Python code in both local and lambda environments. The code was set up locally without a lambda handler, meaning that it does not have a trigger like in AWS, but the internal code is the same. While it is possible to use threads and different cores within Python, the code was not modified for parallelization in lambda in order to avoid falsifying results. Even with this limitation, the number of threads and cores available on a local computer still cannot match the parallelization resources available in lambda.

The results obtained were based solely on the times required to execute exactly the same code, with no modifications for parallelization using threads or cores. This approach allowed for a fair comparison of the performance of local computing resources versus the parallelization capabilities of lambda.

Overall, the results clearly demonstrated the benefits of lambda's extreme parallelization capabilities. Despite the fact that the same code was used in both environments, the lambda environment consistently outperformed local resources. This underscores the power and scalability of lambda for organizations that need to process large amounts of data quickly and efficiently.

The final cost for the anomaly-detection classifier in the Mars atmosphere includes hosting in S3, but without cold storage, as intermediate images are deleted in each step of the process because they are not longer needed. As a result, the overall cost for processing 120 GB of data is reduced while the architecture's efficiency is maximized.

4.4. Validation of the Proposed Serverless Architecture

To perform statistical analysis on a large data set of approximately 400,000 images of the Mars atmosphere, a random sampling of 120 images (0.03%) was taken. The sampling process was designed to be representative and unbiased, with random selection of different ranges of zenith angle, latitude, longitude, and dates. This ensured that the sample was diverse and representative of the overall data set.

After this random selection process, each of the 120 sampled images was manually verified to ensure accuracy and validity. This verification process involved analyzing each image for any anomalies or irregularities, and cross-referencing the results with the statistical analysis of the larger data set.

These are the results:

1. The number of oblique echoes was 90, with the rest being false positives.
2. These 30 false positives correspond mainly (20 ionograms) to cyclotron echoes, which have a similar pattern (see Figures 8 and 9).

While a false-detection rate of 25% may seem high, the technique used to generate an initial data set is still extremely valuable because no oblique echoes were undetected. By using automated processes to detect anomalies in the Martian atmosphere, we are able to generate an initial data set quickly and efficiently without the need for manual inspection of every image.

This initial data set can then be used to train the first machine-learning model for detecting this type of anomalies, with the understanding that the initial model will have a higher

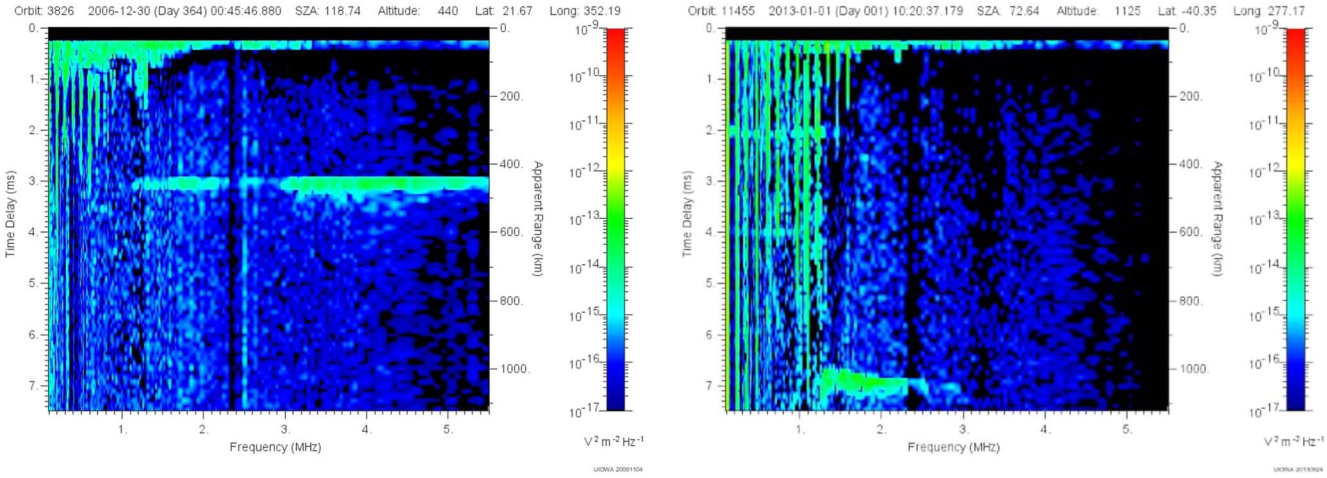

Figure 9. Two examples of ionograms that produced false positives (other than cyclotrons).

false-detection rate due to the limitations of the initial data set. However, this can still be improved over time by incorporating feedback from manual inspections and incorporating additional data to the refining process.

Eventually, with the help of the distributed architecture and trained machine-learning models, we can expect to achieve an extremely efficient classification model with a much lower false-positive detection rate.

5. Conclusions and Future Work

In the present contribution we have proposed a serverless computing architecture to process the ionograms produced by the MARSIS instrument on board the Mars Express mission. In particular, this work has focused on the automatic detection of a particular anomaly (oblique echoes), a process which traditionally has to be performed manually, so it can be used in other instruments.

The resulting platform implemented on AWS Lambda provided much better performance results in comparison with local infrastructure solutions and at a very low cost. In 20 minutes the entire experiment data set was processed by the proposed architecture and only paying for those resources that were used ($0.00001). On the other hand, the local solutions could not complete even 0.01% of it, and their cost does not support the pay-as-you-go model. This makes the proposed architecture a valuable solution for researchers that need a fast, cheap, and reliable anomaly-detection mechanism for the MARSIS ionograms.

Being extremely modular (as it can be seen in Figure 2), mechanisms for detecting additional anomalies can be implemented as AWS Lambda functions and easily integrated in the proposed architecture. This way, all available detection mechanisms can be invoked in parallel once a new ionogram appears in the S3 input bucket.

The computational solution described here was developed specifically for MARSIS data; however similar solutions can be applied to data from other instruments. The approach we present clearly demonstrates that serverless computing can provide significant benefits for processing planetary science data sets.

The current architecture can be improved by reducing the usage of S3 buckets and incorporating temporary folders connected to other lambda functions.

Additionally, the architecture can be further optimized by eliminating the final detection step and incorporating a machine-learning model in SageMaker that is triggered by a lambda function. However, this requires the initial data set to be generated and the model to be trained beforehand. Once the model is uploaded to SageMaker, it can be triggered by a lambda function for anomaly detection.

Acknowledgments

D.P., J.L.V.-P., N.S., L.V., and R.M. acknowledge support through the IN-TIME (Grant Agreement 823934) and EYE (Grant Agreement 101007638) projects from the European Commission. D.P., J.L.V.-P., and R.M.-V. acknowledge support through the EDGECLOUD (RTI2018-096465-B-I00) and EDGEDATA (S2018/TCS-4499) projects. B.S.-C. acknowledges support through UK-STFC Ernest Rutherford Fellowship ST/V004115/1. Finally, all authors would like to thank Dr. Roberto Orosei from the Istituto Nazionale di Astrofisica (Italy) and the rest of the MARSIS operations team for their valuable help during the preparation of this contribution.

Software: The software developed for implementing the architecture described in this paper can be found at the FigShare repository: https://doi.org/10.6084/m9.figshare.22210822.v2.

ORCID iDs

David Pacios https://orcid.org/0000-0001-9299-5292
José Luis Vazquez-Poletti https://orcid.org/0000-0002-6241-8141
Beatriz Sánchez-Cano https://orcid.org/0000-0003-0277-3253
Rafael Moreno-Vozmediano https://orcid.org/0000-0001-9723-8100
Nikolaos Schetakis https://orcid.org/0000-0002-8893-219X
Luis Vazquez https://orcid.org/0000-0003-4054-1197
Dmitrij V. Titov https://orcid.org/0000-0001-7673-9061

References

Ahmad, Z., Jehangiri, A. I., Ala'anzy, M. A., et al. 2021, IEEE Access, 9, 53491
Andrews, D. J., André, M., Opgenoorth, H. J., et al. 2014, JGRA, 119, 3944
Andrews, D. J., Opgenoorth, H. J., Edberg, N. J. T., et al. 2013, JGRA, 118, 6228
Baldini, I., Castro, P., Chang, K., et al. 2017, Research Advances in Cloud Computing (Berlin: Springer), 1

Balla, D., Maliosz, M., & Simon, C. 2021, in 2021 IEEE 14th Int. Conf. on Cloud Computing (CLOUD), 147, doi:10.1109/CLOUD53861.2021.00028
Belgacem, A., & Beghdad Bey, K. 2022, Cluster Computing, 25, 1
Burkat, K., Pawlik, M., Balis, B., et al. 2021, in IEEE 17th Int. Conf. on eScience (New York: IEEE), 40
Castro, P., Ishakian, V., Muthusamy, V., et al. 2019, Commun. ACM, 62, 44
Daw, N., Bellur, U., & Kulkarni, P. 2020, in Proc. 21st Int. Middleware Conf., Middleware '20 (New York: ACM), 356
Eismann, S., Scheuner, J., van Eyk, E., et al. 2020, arXiv:2008.11110
Eivy, A., & Weinman, J. 2017, IEEE Cloud Computing, 4, 6
Fallows, K., Withers, P., Morgan, D., et al. 2019, JGRA, 124, 6029
Fouladi, S., Wahby, R. S., Shacklett, B., et al. 2017, in 14th USENIX Symp. on Networked Systems Design and Implementation (NSDI 17) (Boston, MA: USENIX Association)
Gurnett, D., Huff, R., Morgan, D., et al. 2008, AdSpR, 41, 1335
Gurnett, D. A., Kirchner, D. L., Huff, R. L., et al. 2005, Sci, 310, 1929
Hendrickson, S., Sturdevant, S., Harter, T., et al. 2016, in 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16) (Denver, CO: USENIX Association), https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/hendrickson
Houssein, E. H., Gad, A. G., Wazery, Y. M., et al. 2021, Swarm Evol. Comput., 62, 100841
John, A., Ausmees, K., Muenzen, K., Kuhn, C., & Tan, A. 2019, in Proc. 12th IEEE/ACM Int. Conf. on Utility and Cloud Computing Companion, UCC '19 Companion (New York: ACM), 43
Jonas, E., Schleier-Smith, J., Sreekanti, V., et al. 2019, arXiv:1902.03383
Langlais, B., Thébault, E., Houliez, A., et al. 2019, JGRE, 124, 1542
Lester, M., Sanchez-Cano, B., Potts, D., et al. 2022, JGRA, 127, e2021JA029535
Malawski, M., Gajek, A., Zima, A., et al. 2020, Fut. Gen. Comp. Syst., 110, 502
Malawski, M., Juve, G., Deelman, E., et al. 2012, in SC '12: Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis (New York: IEEE)
Němec, F., Morgan, D. D., Diéval, C., & Gurnett, D. A. 2015, JGRA, 120, 3226
Orosei, R., Jordan, R., Morgan, D., et al. 2015, P&SS, 112, 98
Panneerselvam, A., & Subbaraman, B. 2018, IJET, 7, 174
Quevedo, S., Merchán, F., Rivadeneira, R., & Dominguez, F. X. 2019, in 2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM) (New York: IEEE)
Rajasekar, P., & Palanichamy, Y. 2021, JAIHC, 12, 7621
Raman, R., Livny, M., & Solomon, M. 1998, in Proc. The Seventh Int. Symp. on High Performance Distributed Computing (New York: IEEE), 140
Rodriguez, M. A., & Buyya, R. 2017, Concurr. Comput., 29, e4041
Sabbioni, A., Rosa, L., Bujari, A., Foschini, L., & Corradi, A. 2021, in 2021 IEEE Symp. on Computers and Communications (ISCC), 1
Sánchez-Cano, B., Blelly, P.-L., Lester, M., et al. 2019, JGRA, 124, 4556
Sánchez-Cano, B., Radicella, S., Herraiz, M., Witasse, O., & Rodríguez-Caderot, G. 2013, Icar, 225, 236
Sánchez-Cano, B., Witasse, O., Herraiz, M., et al. 2012, GI, 1, 77
Vazquez-Poletti, J. L., & Llorente, I. M. 2018, CSE, 20, 73
Villamizar, M., Garces, O., Ochoa, L., et al. 2016, in 16th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid) (New York: IEEE), 179
Withers, P., Morgan, D., & Gurnett, D. 2015, Icar, 251, 5