Information Systems Frontiers
https://doi.org/10.1007/s10796-023-10468-5

Tools for Healthcare Data Lake Infrastructure Benchmarking

Tommaso Dolci[1] · Lorenzo Amata[1] · Carlo Manco[1] · Fabio Azzalini[1] · Marco Gribaudo[1] · Letizia Tanca[1]

Accepted: 21 December 2023

Abstract
Vast amounts of medical data are generated every day, and constitute a crucial asset to improve therapy outcomes, medical treatments and healthcare costs. Data lakes are a valuable solution for the management and analysis of such a variety and abundance of data, yet to date there is no data lake architecture specifically designed for the healthcare domain. Moreover, benchmarking the underlying infrastructure of data lakes is fundamental for optimizing resource allocation and performance, increasing the potential of this kind of data platforms. This work describes a data lake architecture to ingest, store, process, and analyze heterogeneous medical data. Also, we present a benchmark for infrastructures supporting healthcare data lakes, focusing on a variety of analysis tasks, from relational analysis to machine learning. The benchmark is tested on a virtualized implementation of our data lake architecture, and on two external cloud-based infrastructures. Our results highlight distinctions between infrastructures and tasks of different nature, according to the machine learning techniques, data sizes and formats involved.

Keywords Data lake · Medical data · Benchmarking · Data analytics · Machine learning

1 Introduction

With the vast amounts of data generated by individuals, companies, and devices, the problem of dealing effectively and efficiently with big data becomes increasingly critical. It is therefore fundamental, for an increasing number of organizations, to ingest, process and save the most important data along with their features, while removing the unnecessary parts.

One of the sectors that contribute significantly to the generation of new data is *healthcare*. In the healthcare field, data include information from diagnoses, lab exams, medical imaging, wearable devices and genomics, and more. The ability to effectively manage and analyze the large amount of data produced not only represents a key competitive advantage for many organizations, but also allows to improve therapy outcomes, enhance medical research, and reduce healthcare costs.

In order to realize the benefits of big data, organizations must possess the infrastructures and tools necessary to handle and analyze data sources, therefore supporting their development is an extremely important priority. In the healthcare domain, valuable data on patients and medications are usually digitized and saved as Electronic Health Records (EHR). Maintaining EHRs on a large scale enables researchers to identify opportunities for moving healthcare organizations towards *personalized healthcare*, which consists in using diagnostic tests to determine which medical treatments work best for each patient (Chawla and Davis, 2013). Typically, only a portion of the currently adopted EHRs contain data in a format that is structured so that researchers and data scientists can easily use, while most of them consist in unstructured and semi-structured data (i.e., data whose structure does not

✉ Tommaso Dolci
tommaso.dolci@polimi.it

Lorenzo Amata
lorenzo.amata@mail.polimi.it

Carlo Manco
carlo.manco@mail.polimi.it

Fabio Azzalini
fabio.azzalini@polimi.it

Marco Gribaudo
marco.gribaudo@polimi.it

Letizia Tanca
letizia.tanca@polimi.it

[1] Dep. of Electronics, Information and Bionengineering,
Politecnico di Milano, Via Ponzio 34/5, Milano 20133, Italy

Published online: 17 January 2024

follow the classical tabular one of the relational model). It is therefore necessary to develop an efficient data administration pipeline to assist scientists in their efforts to provide personalized treatments.

Among the possible big-data management systems, the adoption of data lakes (Khine and Wang, 2018) is becoming in recent years a valuable solution for storing, processing, and analyzing medical data, thanks to the fact that data lakes offer large storage capacity and a variety of processing tools to integrate and manage different data formats. This is especially important in the field of healthcare, where huge amounts of data is constantly produced, and aggregating and integrating data from different sources, e.g., medical devices, electronic records, drug information, is of paramount importance. Moreover, data lakes may include components for the execution of data analysis algorithms, such as sophisticated statistical and machine learning ones, and this, in turn, allows the creation of decision models for better diagnosis and treatment of diseases (Jagadeeswari et al., 2018). However, despite the presence of well-designed examples of data lake architectures in the literature (Hamadou et al., 2020; Liu et al., 2021), none of them has been specifically designed to meet the requirements of the healthcare scenario.

The first goal of this work is to design and implement a proof-of-concept of a data lake for the healthcare scenario, that can ingest, store, process and analyze data from different types of sources (e.g., medical devices, clinical-trial datasets). Data must be saved in their raw format inside the data lake storage area, and subsequently be processed and transformed in an easily-usable way by scientists and analysts: the resulting system focuses on allowing analyses using both machine learning algorithms and conventional queries. Healthcare data are very heterogeneous and, for unstructured data, we decided to focus our analysis on waveforms of patients' life signals and medical images, in addition to structured data management. These types of data, although being a limited part of the ones usually considered in healthcare, allow to characterize a complete workflow, and thus to exemplify the proposed architecture.

Designing, appropriately allocating resources and managing healthcare data lakes can be an extremely complex task, due to the vast amount of data involved, leading to the the necessity of assessing (benchmarking) the infrastructures supporting the data lakes, in terms of their computational power, storage capacities and resource consumption. This is critical for many reasons: for instance, it allows the optimization of resource allocation by identifying the areas where the system is consuming more resources, or the identification of performance improvements by detecting bottlenecks and assessing CPU and RAM usage. Finally, the creation of a healthcare data lake benchmark helps to establish a standard set of criteria against which the performance of a system can be measured. Therefore, the second goal of this work has been to develop a benchmark tool to assess the resource consumption and execution time of analysis tasks in healthcare data lakes, named SEASHELL (re**S**ource b**E**nchmark for **A**nalysis ta**S**ks in **HE**a**L**thcare data **L**akes). In fact, data analysis and processing are not only the main drivers of the big data phenomenon, but they also represent a critical and computational-intensive task in modern healthcare organizations, from hospitals to medical research centers and universities.

Our benchmark is designed to analyze the data lake's performance under different workload scenarios, by means of a variety of tasks involving real-world applications, using medical data collected from different healthcare organizations. This paper is an extension of our previous work (Manco et al., 2023).

To summarize the scope of this research, the contributions of this work include: *a)* the design and definition of a data lake architecture that can effectively manage both structured and unstructured medical data; *b)* the implementation of a proof-of-concept data lake on the basis of the proposed architecture; *c)* the creation of a benchmark to accurately assess the performances of healthcare data lakes in data-analysis tasks; *d)* the execution of the benchmark on our proposed architecture, and on two more external architectures.

The rest of the paper is organized as follows: Section 2 explores the state of the art on data lake infrastructures for healthcare and data lake benchmarks. Section 3 introduces our data lake architecture for managing medical data. Section 4 describes the technologies necessary to implement our architecture and presents a proof of concept. Section 5 introduces SEASHELL, our benchmark for healthcare data lakes. Section 6 shows the results of executing the benchmark on our architecture and on two additional external infrastructures. Finally, Section 7 concludes the paper and outlines future work.

2 Related Work

2.1 Data Lake Architectures

Data lake architectures are usually classified into Data-Pond Architectures and Zone Architectures (Sawadogo and Darmont, 2021b; Giebler et al., 2019). Pond Architecture (Inmon, 2016) designs a data lake as a set of *data ponds* each dealing with data of a specific type and nature (e.g., structured data, textual data, etc.). Zone Architecture (Giebler et al., 2020) designs a data lake as a set of zones, each associated to data with a different degree of refinement (e.g., raw zone, refined zone, etc.). The number of zones can vary according to the specific architecture, as can the properties of each zone. In general, different architectures are suitable for different scenarios: for instance, a Zone Architecture is more appropriate for healthcare (Barbierato et al., 2021), since different zones, which can be physical or logical, provide the possibility for the data to be refined multiple times while maintaining their initial copy in raw format.

So far, no characteristic and complete data lake architecture has been defined in the literature yet, and this makes it difficult to design and implement this kind of storage. While interesting data lake architectures from the literature (Inmon, 2016; Ravat and Zhao, 2019; Sawadogo and Darmont, 2021b) are still too generic to be implemented, the commercial products implement only few of the features that would be needed. The Data Lake Architecture Framework (Giebler et al., 2021) is presented as a high-level guide for building a comprehensive data lake. This framework defines nine aspects that should be defined to design a complete data lake, including *vertical* aspects (data processes, organization, modeling, flow, storage, infrastructure) and *horizontal* aspects spanning over all the vertical aspects (metadata management, security and privacy, data quality).

Considering the healthcare scenario, most of the architectures proposed by the literature do not qualify as comprehensive solutions. Data lakes are defined in both the scientific and industrial worlds as a repository storing raw data in their native format; however, different definitions focus on different aspects, e.g., in Walker and Alrehamy (2015) governance and metadata management are highlighted, while Madera and Laurent (2016) places importance on the users, presenting an architecture centered on researchers and analysts.

Many data lakes designed for non-healthcare scenarios present robust solutions, but disregard certain requirements that are essential for healthcare. In fact, the design of a data lake is strongly influenced by the kind of scenario in which it is placed. For instance, CoreDB (Beheshti et al., 2017) is an open-source data lake to organize, index and query data and metadata, but does not guarantee the storage of data in their native format. Hydria (Deligiannis et al., 2020) is a data lake for cultural heritage data: it provides an integrated framework that enables easy deployment of data acquisition services, dataset sharing with other stakeholders, search, filtering and analysis of data via visualization tools. Constance (Hai et al., 2016) manages structural and semantic metadata, but scenario-specific features must be included in order for it to be used in real-world applications. ArchaeoDAL (Liu et al., 2021) has been developed using a very effective multilayer approach; however, it is designed to mainly handle relational databases. FEDDL (Flexible Energy Denmark data lake) (Hamadou et al., 2020) is a data lake with the purpose of collecting and sharing energy-related data in Denmark, with integrated AI and machine learning tools to analyze them. FEDDL proposes a solid architecture and provides a technical overview of the whole structure. In fact, their findings represented an important source of inspiration for the definition of our architecture.

In the healthcare field, DIFUTURE is a German initiative centered on the creation of a data lake for medical data storage and integration (Prasser et al., 2018). Similarly, Cappiello et al. (2022a) discusses a data lake federation for the collection, storage and distributed analysis of healthcare data. Moreover, cloud-based solutions offered by providers such as AWS (Amazon Web Services) and Azure are emerging (Hukkeri et al., 2020) and some data lake have been implemented leveraging their services. A medical data lake is proposed in Sha M and Rahamathulla (2020), whose authors suggest a design that relies on cloud services. Similarly, Taher et al. (2019) proposes an IoT-Cloud–based framework for real-time processing of big data in the healthcare domain, implementing AWS. Maini et al. (2018) describes a cloud architecture that make use of Azure functionalities: huge amounts of data are efficiently stored using Azure data lake, to support physicians in detecting heart diseases.

However, the main challenge of cloud technology applications in the healthcare sector is the ownership of sensitive data (Rieke et al., 2020). Proper security measures must be implemented to protect data at each level of data management. Despite cloud providers offering various security services, it is not always clear how to integrate them with the proposed architecture. Moreover, working on proprietary and not on open source software greatly limits the customization of the solutions, and in addition cloud providers usually require monetary payments based on software usage and the hardware made available. Recently, despite the absence of a full-fledged data lake for healthcare, researchers are working towards this direction, for instance by developing query engine components for clinical data (Heinis and Ailamaki, 2017; Weber et al., 2009), by studying federated learning solutions to preserve data privacy (Rieke et al., 2020), by investigating heterogeneous data management in data lakes (Ren et al., 2021) or data quality aspects for medical data lakes (Eder and Shekhovtsov, 2021).

2.2 Data Lake Benchmarks

In recent years, there has been a growing trend towards producing diagnoses using medical data (Raghupathi and Raghupathi, 2014), driven by advances in machine learning and artificial intelligence. For instance, machine learning algorithms proved to be very effective in diabetic retinopathy (Gulshan et al., 2016; Krause et al., 2018) and skin cancer detection (Esteva et al., 2017), frequently achieving results better than humans. Machine learning is also used for early prediction of sepsis (Giacobbe et al., 2021; Mollura et al., 2020), COVID-19 mortality (Karthikeyan et al., 2021), or intensive care unit outcome (Bhattacharya et al., 2017).

However, the large volumes of medical data and their heterogeneity have posed many challenges. For this reason, a benchmark tool is essential to evaluate the performance of hardware resources and processing capabilities of any data lake system, to determine whether the system is performing optimally or there are bottlenecks to be addressed. DLBENCH is a data lake benchmark which do not target a particular application scenario, focused on the analysis of tabular and textual data (Sawadogo and Darmont, 2021). It involves machine learning tasks for text mining analysis, but it does not measure RAM or CPU metrics, which are fundamental to assess data lakes architectures.

Over the years, researchers have conducted many studies on benchmarks for big data frameworks and architectures, for instance by assessing performance with queuing network modeling techniques (Barbierato et al., 2021). In this paragraph, we present a selection of works from the literature on benchmarking big data architectures. TCPx-HS verifies big data system performance, availability, and optional energy usage. It allows for the measurement of both hardware and software, such as Hadoop Runtime, Hadoop Filesystem API compliant systems, and MapReduce layers (Transaction Processing Performance Council, 2021). HiBench is a benchmark specifically designed for assessing Hadoop distributed systems (Huang et al., 2010). BigDataBench (Wang et al., 2014) is a comprehensive benchmark for assessing big data frameworks without targeting a specific scenario. It includes nineteen different workload scenarios with varying data inputs. TextBenDS includes two sets of queries for producing workloads, for keywords discovery and document ranking, to be used in Hadoop and Spark cluster architectures. The extracted metrics concern the query execution time, but do not consider other hardware consumption metrics (Truică et al., 2020). Finally, SPEC CPU is a widely-used benchmark suite designed to measure the performance of CPU in distributed big data systems, simulating various real-world applications, such as scientific computing, database management, and image processing (SPEC, 2017).

Overall, literature lacks benchmark solutions specifically designed for healthcare data lake architectures. Given the importance of medical data analysis for improved disease diagnosis, prevention, and personalized medicine, the analysis layer of big data architectures is especially fundamental in the context of healthcare. Moreover, merging too many aspects on a single benchmark may lead to lack of precision, collection of unnecessary metrics and a dispersive solution (Bajaber et al., 2020). Therefore, our benchmark is focused on the resource consumption of analysis tasks within healthcare data lakes.

3 Data Lake Architecture

3.1 Goals and Requirements

In order to retrieve valuable insights and helpful knowledge easily from data, healthcare researchers require a unified system (Cappiello et al., 2022a) that allows to:

- manipulate, process, and analyze different types of data, from fully structured to unstructured, allowing their storage inside the system in raw native format;
- handle different types of data effectively, with open possibilities for integrating information from IoT devices, with the system serving as a centralized location for all data collected by various injector systems;
- provide a distributed file system where data can be stored in files and directories to allow for efficient data loading, while guaranteeing high availability and fault tolerance without impacting application performance;
- meet clinical requirements by allowing fast and efficient analysis of new combinations of data.

From this list of requirements, it is evident that one of the most critical characteristic of medical data is heterogeneity: from structured data to images, healthcare produces a large variety of different files. Data heterogeneity has been recently listed as one of the biggest challenges of healthcare data management (Canham et al., 2021). A data lake is the most appropriate framework to adequately manage this factor. In fact, data lakes can provide a unified platform for all relevant data generated by healthcare systems, operating as a repository for both structured data collected from traditional databases and unstructured data derived from various other sources. Data lakes are extremely fast and versatile because they implement a scalable architecture to store data in their native form, while remaining easily accessible and centralized for end users. Furthermore, data lakes can be fully equipped with various security layers to ensure data integrity and compliance with privacy and regulations, which is particularly important in the healthcare context.

This work focuses primarily on the definition of the physical components of a data lake architecture, and conceptual aspects such as security and privacy are left for future work since their components are outside the main data management pipeline (Giebler et al., 2021).

3.2 Multi-layered Architecture

In our case study, we want to define, as a first step, a functional implementation similar to other solutions from the literature (Hlupić et al., 2022), proposing a multi-layered approach based on incorporating layers with separation of concerns. In fact, this type of architecture has the advantage of clearly highlighting the functions to implement for a given data lake, which allows for an easy matching to the corresponding required technologies: the key concept is that each layer communicates with the adjoined ones, and the data follow a pipeline over all the layers. Figure 1 illustrates the specific layers considered in the proposed architecture.

In particular, starting from the top of the diagram, the **Data Ingestion** layer has the task of ingesting heterogeneous data in raw format from various data sources and into the data lake. The two main options to load data into a data lake are *batch* and *streaming*. The choice of implementing a batch-oriented or stream-oriented data ingestion layer depends very much on the context to which the data lake is applied: in fact, as the context changes, we may have more or fewer data sources providing streaming data.

The **Data Storage** layer is the core of the data lake. It contains raw-data repositories, but also transformed data; it provides support for different forms and structures of the data, including file storage and raw-record storage.

The **Data Transformation** layer provides the potential for the scalable execution of operations such as data cleaning and data transformation, in order to achieve a predefined final form. In this layer, different data representations are created in order to transform raw data into easily accessible data formats, based on algorithms' and users' needs.

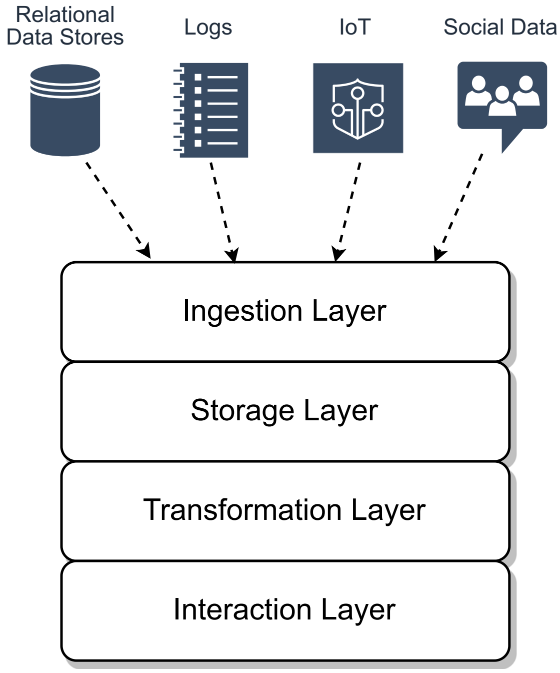


Fig. 1 Overview of the multi-layered data lake architecture described in (Hlupić et al., 2022)

Finally, the **Data Interaction** layer provides end-users with access to the data created in the transformation layer. Users can access data in order to perform exploration tasks, create and apply analytical queries and visualize the stored data using various visualization tools.

Multi-layered data lakes frequently also include horizontal layers (e.g., metadata management) to include aspects that span over all the data lake pipeline from data ingestion to data interaction (Hamadou et al., 2020). We further discuss these layers in Section 4.3.

3.3 Overview of the Architecture

In the healthcare domain, a Zone Architecture for data lakes is preferred (Barbierato et al., 2021). In fact, data should go through different levels of refinement while maintaining a copy of the data in their raw format, characteristics that Pond Architecture does not provide (Inmon, 2016; Sawadogo and Darmont, 2021b). These functionalities are essential for healthcare researchers to allow them and analysts to perform more than a single transformation and analysis on the same data. More in detail, the proposed architecture is composed of five different zones: Transient Landing Zone, Raw Zone, Process Zone, Refined Zone and Consumption Zone, as illustrated in Fig. 2.

Figure 3 shows a high-level view of the proposed system, illustrating the main layers of the architecture, where each layer is represented by its corresponding component. Data are first ingested by the Ingestion component in the Transient Landing Zone, and then permanently saved in the Raw Zone by the Data Storage Component. Then, they are processed inside the Process Zone in different ways according to their format. Moreover, the output of the transformation performed by the Processing Component, known as "refined data", is stored in the Refined Zone. The Interaction Component is the module that deals with data analysis and querying; it has free access to both the Raw and Refined zones of the Data Storage, and produces data in the Consumption Zone.

From Fig. 3 it can also be observed that each component is associated with at least one zone. In particular, the **Data Ingestion Component** represents the Transient Landing Zone: batch and streaming data land here after being extracted from the data sources and pushed into the data lake. Data are transferred without any transformations, in accordance with the first steps in the Extract-Load-Transform (ELT) paradigm. Vital signs from hospitalized patients, reports written in natural language, and structured data on patients and lab events are all ingested regardless of their format. In this work, we focused on a batch-type data ingestion.

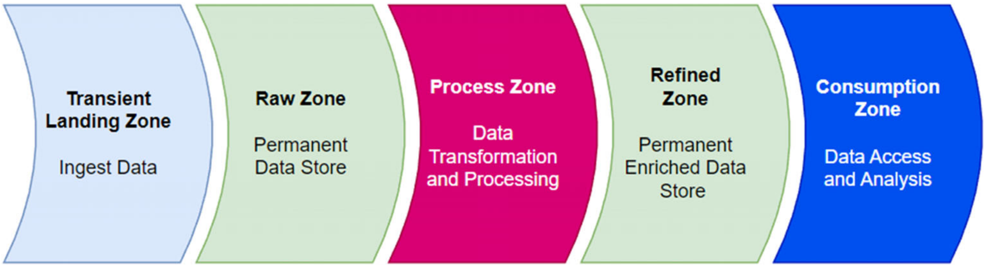


Fig. 2 Overview of the adopted zones

The **Storage Component** represents the central repository where large volumes of medical data are stored. This module must provide a distributed file system while guaranteeing high availability and fault tolerance. In particular, it contains two zones, namely the Raw Zone, i.e., the area where the data are permanently stored in their native raw format, and the Refined Zone, where data are stored in a form suitable for the end-users and where processed data are saved. In order for scalability to be guaranteed, this component should be able to increase its storage capacity when needed.

The **Processing Component** represents the Process Zone. This is where data are brought to be prepared and processed. For instance, we can process time series data before saving them in the Refined Zone, exposing them to a transformation procedure to make them more easily interpretable by the final user.

Finally, the **Interaction Component** contains the Consumption Zone in charge of granting access to the data in the Refined and in the Raw Zone. Here users can perform queries on structured data by directly using Python libraries, or apply various data analysis techniques. For instance, reports written in natural language can be subject to language-understanding tasks, or data can be used for training machine learning models, which currently represents a very common application for medical data. Results obtained from this component can be returned to the Raw Zone for later use. Saving results is very important to allow further analysis: for example, keywords extracted from reports can be later mined to find patterns or apply clustering techniques.

4 Implementation

In this section we describe an implementation of the data lake architecture described in Section 3, from the technologies needed to implement each component to the data flow through them, and finally we present a proof-of-concept of our solution.

4.1 Technologies

Many data-management tools can be implemented inside a data lake (Agrahari and Rao, 2017). In our solution, the key tools for data management are two: the file system, responsible for collecting the data in a distributed storage, and a tool for performing ELT-type data ingestion that can easily connect to various data sources and to the distributed storage.

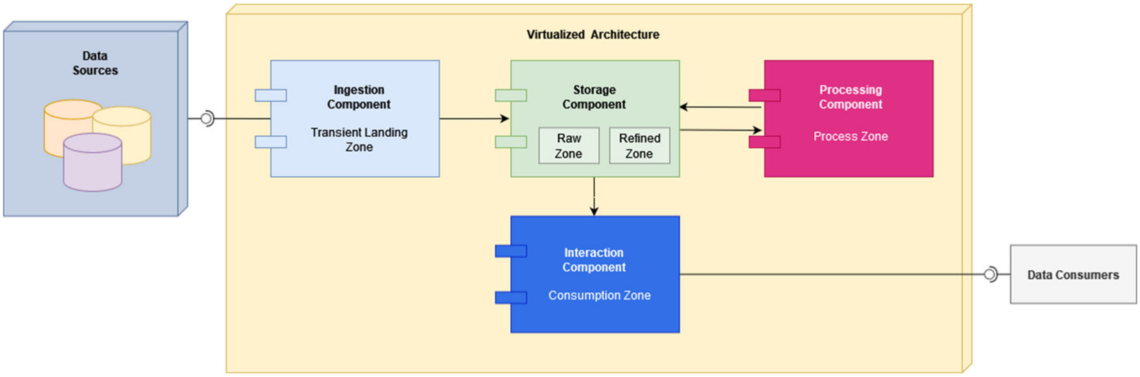


Fig. 3 High-level component view of the proposed data lake architecture

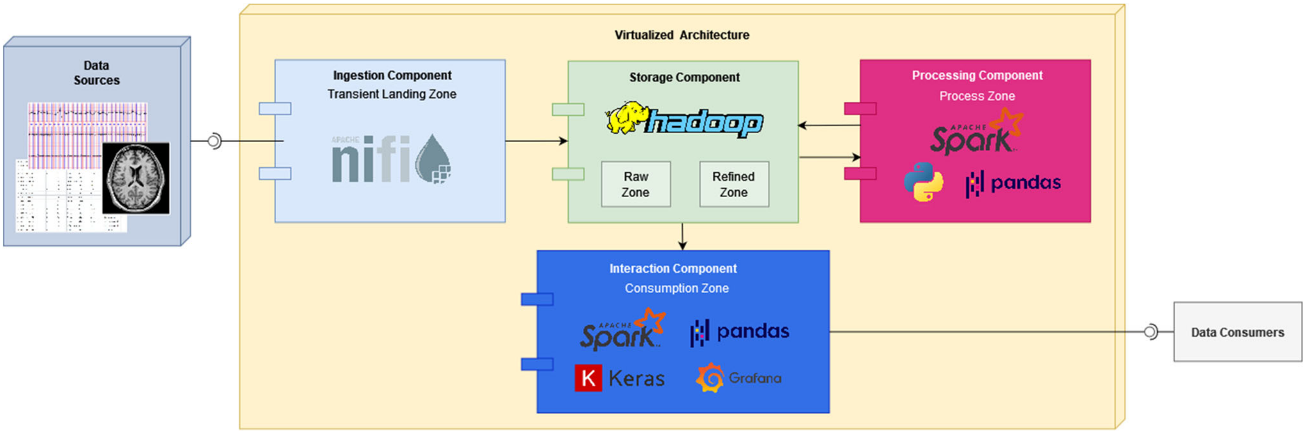


Fig. 4 Technical overview of the system, where each component of the architecture is associated to the corresponding technology that implements it

In particular, to fulfill the file-system requirements listed in Section 3.1 we chose HDFS (Hadoop Distributed File System).[1] In fact, HDFS provides high-performance access to data across highly scalable Hadoop clusters and can handle big volumes of both structured and unstructured data. Each cluster is formed by a single main node, called Namenode, and by a variable number of secondary nodes called Datanodes. Additionally, in HDFS it is possible to specify the number of copies of a file that should be maintained, ensuring appropriate data replication.

Secondly, we selected Apache NiFi[2] as tool for data ingestion. NiFi has many advantages compared to its competitors (Alwidian et al., 2020), mainly:

- it provides both batch and real-time data transfer,
- it can easily interact with the Hadoop ecosystem, and particularly with HDFS,
- it supports the most widely used communication protocols, and
- it scales linearly to a high number of nodes.

Additionally, each node of NiFi provides the same functionality as every other node: this makes the system very robust to node failures.

Since unstructured data constitute a significant percentage of healthcare information, a remarkable amount of time is spent in the ingestion of waveforms; therefore, to support stream ingestion we adopted Apache Kafka,[3] an open-source event processing platform that can be easily integrated with NiFi, while preserving system scalability (Isah and Zulkernine, 2018).

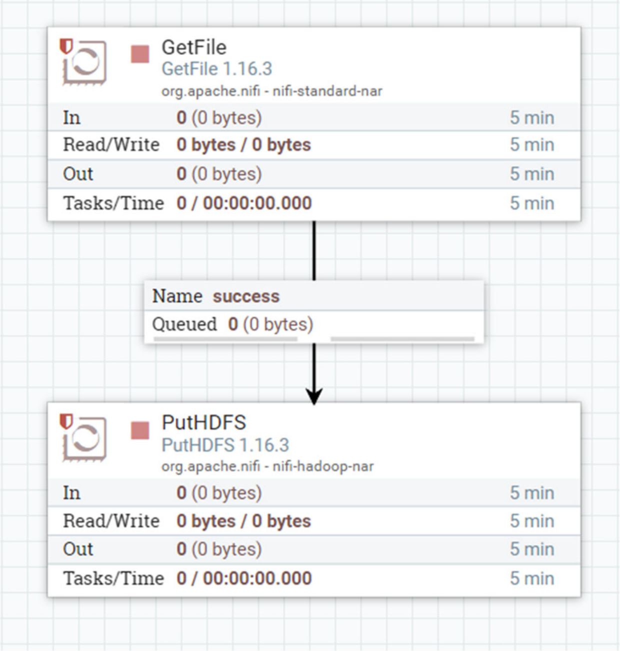
The processing functionalities of the data lake are implemented by leveraging Apache Spark (Zaharia et al., 2016), a unified engine for big-data processing with high flexibility and speed. Spark supports a variety of operations, from SQL queries to advanced machine learning algorithms, which make it the perfect candidate for managing and analysing medical data, considering their high hetereogeneity.

Finally, the Interaction Component is produced using a variety of technologies and modules that allow the execution of different analyses and inspection of previous results, such as Apache Spark MLlib (Meng et al., 2016), Keras (Chollet et al., 2015), Pandas (McKinney et al., 2010) or directly Python itself. We also implemented the open-source monitoring tool Grafana (Chakraborty and Kundan, 2021), an improved tool for Apache Spark used to display system metrics in a more pleasant and compact way w.r.t. the default Graphite. The full technical overview of the system is shown in Fig. 4.

4.2 Data Flow

In this section we describe the data flow in our data lake architecture. The first stage is the ingestion into the Transient Landing Zone: this phase starts when raw data are extracted from the sources to start their path inside the data lake, and ends when these data are placed within the storage area provided by HDFS. The entire process is handled by Apache NiFi. The flow diagram in Fig. 5 shows the work of the two NiFi processors used to manage the ingestion phase: the GetFile processor fetches the data from the data source and then hands them to the PutHDFS processor; PutHDFS establishes the connection to HDFS and inserts the data into the distributed storage. Once the distributed memory is reached, the data are divided into blocks and replicated into different Datanodes for a number of times equal to the replication factor, in our case equal to 3. HDFS ensures that this operation is performed automatically to increase data availability and

[1] https://hadoop.apache.org
[2] https://nifi.apache.org
[3] https://kafka.apache.org



GetFile
GetFile 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

PutHDFS
PutHDFS 1.16.3
org.apache.nifi - nifi-hadoop-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Fig. 5 Data flow in Apache NiFi

make the system more robust to individual Datanode failures. Throughout the ingestion phase, the data are neither modified nor transformed: this is essential to guarantee that the raw data are kept in the Raw Zone as they were provided by the sources.

At this point, the raw data saved within the distributed storage are ready to be processed. This task is performed inside the Process Zone, and the processing procedure depends on the data format and type; for instance, we considered the processing of time series data (waveforms) extracted from MIMIC-III (Johnson et al., 2016b), a popular medical dataset containing both structured and unstructured data. Physicians and medical researchers may need to process the raw data in order to transform the waveforms into a more easily interpretable format. This task involves the addition of information—such as the identifier of the patient and the starting date and time of the measures—to the file containing the signal measurements, and this information is contained in the header file associated with the waveforms. The process takes advantage of the *hdfs* Python library,[4] used to handle communication with the storage, not only when the processor node needs to take data to be processed, but also when there is the need to rewrite data into the Refined Zone after processing. For what concerns the execution of data requests in the system, two scenarios may occur, described below.

Scenario 1

In this scenario, the data requested by the user have already been processed because of some earlier request, and therefore they are already present in the Refined Zone. In this case, the current request is immediately addressed, and the data are directly retrieved from the Refined Zone.

Scenario 2

This scenario involves real-time processing of the data requested by the user, due to the fact that the same request has never been processed. In this case, initially the data are searched in the Refined Zone anyway but, since some or all of the data are not available there, the request must be forwarded to the Raw Zone. The requested data found in the Raw Zone are sent to the Process Zone, ready to be processed in real-time. After the processing, following the normal flow of data, they are saved into the Refined Zone, which is finally ready to meet the user request. This flow is shown in Fig. 6.

4.3 Horizontal Layers

While the literature on medical data lakes mainly focuses on *vertical layers* (e.g., data storage, exploration, etc.) (Ren et al., 2021), the Data Lake Architecture Framework (Giebler et al., 2021) identifies three foundational horizontal aspects for data lake architectures: data quality, metadata management, security and privacy. These aspects can either be realized by independent *horizontal layers* (Hamadou et al., 2020), or addressed by existing vertical layers. In this section, we describe how to consider these aspects in our architecture, suggesting adequate solutions and strategies for their implementation in line with the components described in Section 4.1.

First, we consider the four main dimensions of data quality: accuracy, completeness, consistency, timeliness (Batini et al., 2009). Completeness, accuracy and consistency are addressed right after data ingestion and storage: raw data can be processed in the Data Transformation Layer (realized by the Processing Component), before being re-loaded in the Refined Zone of the Storage Component. On the other hand, the timeliness property is addressed by streaming ingestion in the Data Ingestion Layer.

One additional challenge in medical data lakes is data cataloging and metadata management (Cappiello et al., 2022b). In this respect, metadata management spans all the zones of the architecture, as metadata can be produced at any stage, thus constituting an independent horizontal layer. Relevant metadata range from technical information to quality or provenance attributes: a scalable solution for defining and managing metadata in data lakes is Atlas,[5] an Apache tool that was recently validated in the context of healthcare data architectures (Kumar, 2023).

[4] https://pypi.org/project/hdfs

[5] https://atlas.apache.org

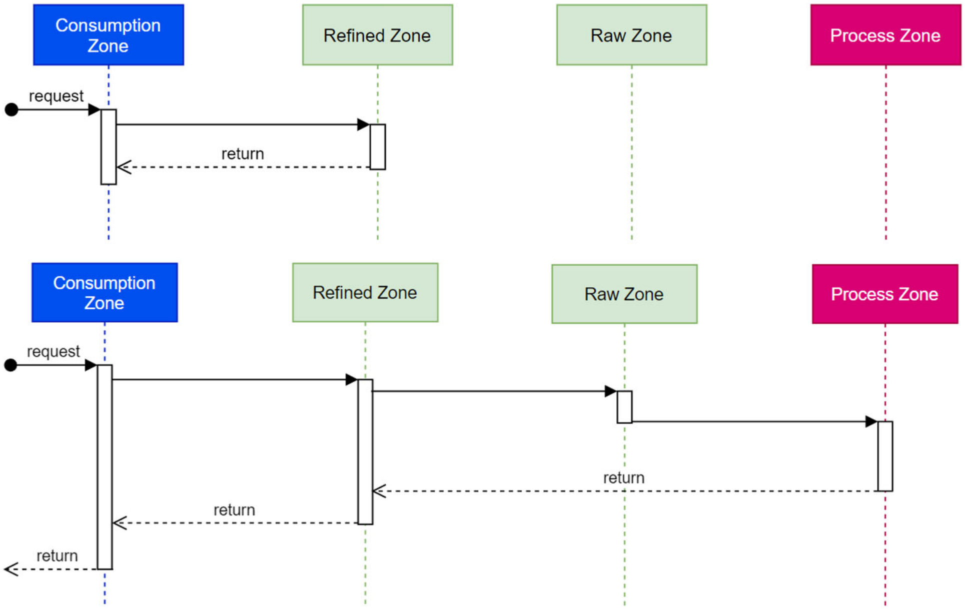


Fig. 6 Two sequence diagrams illustrating user requests to access data. On the top diagram, data are already available in the Refined Zone. On the bottom diagram, data are not yet stored in the Refined Zone, thus need be pre-processed

Regarding data security, in this work we focus on a local evaluation of the proposed system, but we are aware that, in real-case scenarios, data architectures may employ external resources that may be critical w.r.t. security (e.g., cloud); in this case, an appropriate security policy must be adopted, for instance through data encryption (Couto et al., 2019). Nonetheless, in the context of healthcare it is preferable to use private cloud infrastructures, directly under the control of the hospital or the medical research center. Regarding access control and authentication, appropriate Apache tools based on the Hadoop platform can be included.[6]

Another important issue concerns privacy, due to the need of properly managing and anonymizing personal data. In our solution this procedure is addressed during the data collection stage, when personal data are either appropriately collected with the consent of the patient in accordance with the GDPR rules, or suitably anonymized before ingestion and storage in the data lake. In accordance with this view, the datasets used in our experiments contain anonymized data, such as in the case of MIMIC (Johnson et al., 2016b).

4.4 Proof of Concept

This section describes a proof of concept of our solution, focusing on the vertical layers of the architecture from the ingestion of data sources to the evaluation of simple data analysis tasks in the Consumption Zone. The considered layers are implemented on several container-based virtual machines using Ubuntu 22.04 as the operating system, running on a single physical machine with a quad core CPU Intel Core i7-4790S 3.2 GHz, 16 GB RAM, GPU NVIDIA GeForce GTX 745, and 2 TB Hybrid HDD. The following are the specific nodes of the architecture we implemented:

- a single ingestion node, implementing Apache NiFi and representing the Transient Landing Zone;
- a single node implementing an HDFS Namenode for managing data storage;
- a cluster of five nodes, each implementing an HDFS Datanode instance, representing the Storage Component and incorporating both Raw Zone and Refined Zone;
- a single Apache Spark driver node;
- a two-node cluster with each node running an Apache Spark executor instance;
- a single analysis node, equipped with useful libraries to perform analysis and to access data (Consumption Zone).

The three nodes running Apache Spark are meant to process raw data and realize the Process Zone. It is important to note that the Hadoop nodes may also be used to conduct data analyses through the MapReduce framework, but we decided to focus on Apache Spark (Zaharia et al., 2016), due to it being faster than MapReduce, more documented and more popular (Xin, 2014). Figure 7 shows the network architecture of the nodes that implement the core of the data lake, i.e., the Storage and Processing Components. We run Hadoop version 3.2.1 and Spark version 3.0.2.

We adopted Docker[7] to virtualize the nodes, and configured the system with the required tools. The Hadoop cluster can extend or reduce the number of Datanodes, by leveraging the potential of an orchestration tool such as Docker

[6] https://ranger.apache.org

[7] https://www.docker.com

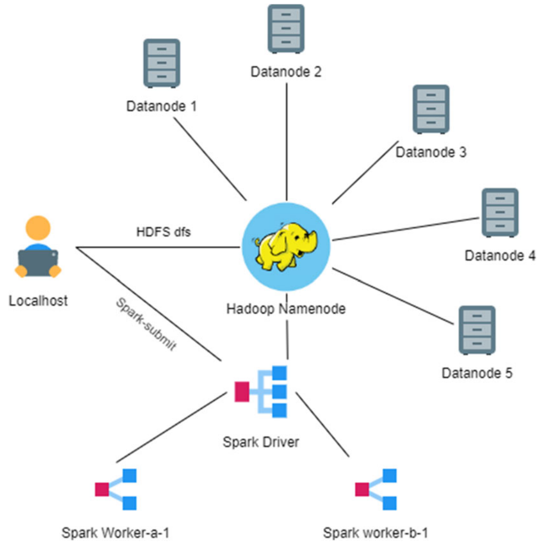


Fig. 7 Network architecture of the nodes used to implement the Storage and Processing Components, realizing the Raw, Process and Refined Zones of the data lake

Compose, thus favoring the scalability of the system. The considered setup can be easily migrated to run across multiple virtual or physical machines, using for example Kubernetes.[8]

Additionally, the following requirements are essential for the proper execution of the system: *(a)* the nodes must be instantiated within the same network; *(b)* the Namenode and Datanodes communicate with each other properly: the Namenode must hold information about the IP addresses of the Datanodes in the network, and each Datanode must be familiar with the IP address of the Namenode and fellow Datanodes; *(c)* the ingestion node should have all the information needed to access the Storage Component (IP address, port number, etc.), and the Hadoop namespace configuration files contained in the Namenode must be copied to the Apache NiFi node; *(d)* the processing node and the analysis node implements the *hdfs* library to establish and manage communication towards the distributed storage.

After configuring the entire system, we have a virtual prototype architecture to: (i) store medical data exploiting the features of the Hadoop cluster, (ii) run Spark applications for processing it, and (iii) consume the results of these analyses. The next step is to test the system to ensure the architecture's effectiveness and to evaluate the components' compatibility and communication.[9]

Table 1 Execution time and transfer rate for the ingestion phase

Batch (MB)	GetFile (s)	PutHDFS (s)	Full Time (s)	Transfer Rate (MB/s)
11	0.5	4.5	5.0	2.20
156	5.0	33.0	38.0	4.11
305	8.0	44.0	52.0	5.87
1024	35.0	75.0	110.0	9.31
1741	52.0	106.0	158.0	11.02
3738	86.0	194.0	280.0	13.35

4.5 Evaluation

We performed an evaluation of the ingestion and storage components to assess the behavior of the proposed architecture as the size of the data batch varies. Table 1 shows the results of transferring data to the HDFS cluster using Apache NiFi. In particular, we evaluated the ingestion time of structured data and waveforms, from MIMIC III clinical database and MIMIC III waveform database respectively. Measurements are replicated 20 times per batch size. Similarly, Table 2 shows evaluation results of accessing data from the Storage Component. Our simulation involves both scenarios described in Section 4.2, i.e., when data is already present in the Refined Zone, and when it is not.

From this evaluation, it is very clear that Apache NiFi and HDFS work very well with large batches of data. On the other hand, due to the risk of lower transfer rates, using small batch sizes is not recommended. Moreover, regarding HDFS, due to its approach of dividing data into fixed-size blocks, to avoid flooding the Datanodes with unfilled blocks it is better not to ingest an amount of data whose size is much higher than the block size. For what concerns the access phase, since the processing phase is the slowest step in the pipeline, it is appropriate to process larger data asynchronously with respect to user requests, at least for intensive data processing procedures, like the case of waveforms. In fact, real-time processing is efficient only for data of smaller size. More detailed information on this evaluation can be found in our previous work (Manco et al., 2023).

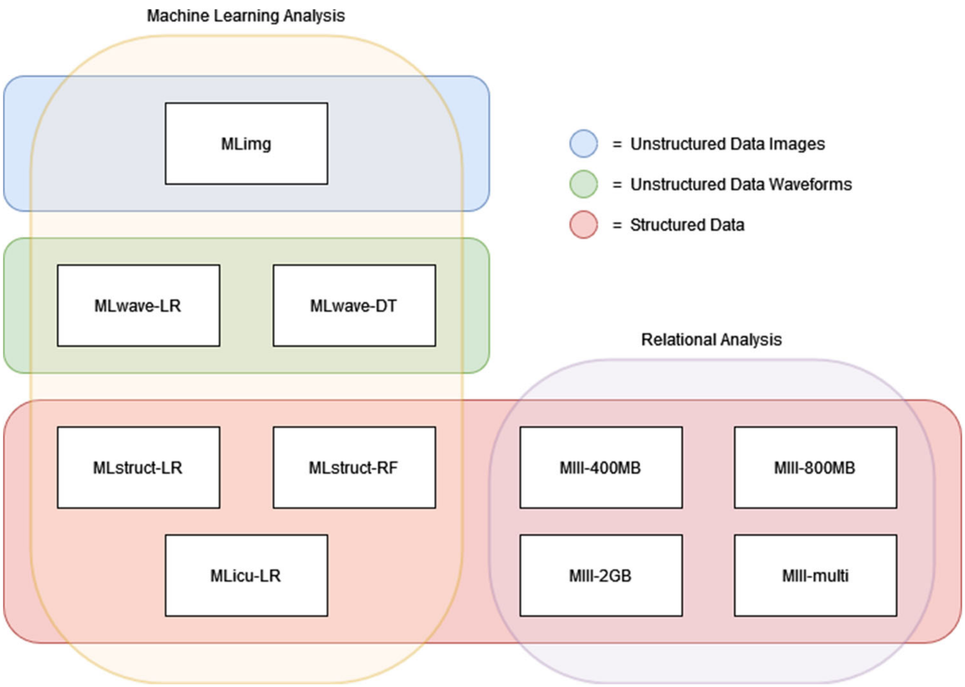
Table 2 Comparison between access times of data already available in the Refined Zone and yet to be processed in the Raw Zone

Batch (MB)	Refined Zone (s)	Raw Zone (s)
0.1	0.030	0.095
1	0.094	1.213
11	0.467	10.789
53	1.330	51.342
108	3.713	107.432

[8] https://kubernetes.io

[9] The proof-of-concept implementation of the data lake architecture is available at: https://github.com/MancoCarlo/healer-prototype.

Fig. 8 Overview of the analysis tasks in SEASHELL



5 Benchmark

This section presents SEASHELL, our benchmark for assessing resource consumption and execution time when performing analysis tasks with medical data, in infrastructures supporting data lake architectures. In fact, when working with data lakes, it is very important to consider the performance and the correct resource sizing of the underlying infrastructure, which in turn requires performance analyses as close as possible to real applications. Our benchmark focuses on a variety of medical data analysis tasks, ranging from relational analysis to the application of machine learning techniques.

SEASHELL consists of ten different tasks, differing in terms of: the type of data employed (structured, unstructured), the type of analysis (relational analysis, machine learning), and the size of the input file. This allows for a wider perspective on the results of the performance analysis: each task is a generic analytic process involving medical data that generalize healthcare-related research. From a thorough research in the literature, we selected tasks involving relational analyses, decision support systems for disease detection and classification (Dritsas and Trigka, 2022; Deekshatulu et al., 2013; Khemphila and Boonjing, 2011), models for mortality risk assessment (Iwase et al., 2022; Bhattacharya et al., 2017), machine learning applied to waveform data (Alarsan and Younes, 2019), and neural networks applied to medical imaging (Lundervold and Lundervold, 2019). Figure 8 provides a general overview of the ten tasks contained in SEASHELL, classified according to the nature of the analysis and the type of data involved.[10]

5.1 Datasets

This section provides an insight of the datasets involved in the analysis processes of SEASHELL. The dataset used for relational analyses is the MIMIC-III Database (Johnson et al., 2016b)[11], released by PhysioNet (Goldberger et al., 2000). MIMIC stands for Medical Information Mart for Intensive Care, and is a large, freely-available database comprising de-identified health-related data associated with over forty thousand patients who stayed in critical care units. It includes a structured clinical database (Johnson et al., 2016a) and a related waveform database (Moody et al., 2020). The clinical database includes 26 structured tables, comprising 6 tables to track patient stays, 8 tables providing data from critical care units, 7 tables containing data collected in the hospital database, and 5 tables dictionaries for cross-referencing codes against their respective definitions, i.e., International Classification of Diseases (ICD) and Current Procedural Terminology (CPT). In the waveform database, records typically contain ten or more time series of patients' vital signs sampled once per second or once per minute. Each of these records is composed of a header file which shows informa-

[10] Code from the benchmark tasks is available at: https://github.com/TommasoD/SEASHELL.

[11] https://physionet.org/content/mimiciii

tion about the measurements (number of samples, start time of the measurement session, description about the observed signals) and a matching signal file.

MIMIC includes tables whose dimension range from few MB up to GB, and whose nature is well-fitted for relational analysis, but not much for the application of machine learning algorithms, due to the lack of attributes appropriate for forecasting or labeling. For this reason, we selected two additional relational datasets to apply machine learning algorithms to: the Stroke Prediction Dataset and the ICU Patients Mortality Prediction Dataset.

The Stroke Prediction Dataset[12] includes data on 5110 different patients, with 13 columns about general and clinical information of each patient (Soriano, 2021). This dataset is well suited to train a machine learning model to predict the likelihood of a patient suffering a stroke, based on input parameters such as gender, age, diseases, and smoking habits.

The ICU Patients Mortality Prediction Dataset[13] is a PhysioNet dataset created for the 2012 Computing in Cardiology Challenge on predicting ICU mortality, containing vital signs information of 4000 patients that stayed in ICU for at least 48 hours (Silva et al., 2012). Each row refers to a unique patient and is associated with 15 attributes containing the mean value of multiple consecutive vital-signs recordings (e.g., sodium, arterial pH, arterial oxygen). Additional columns express categorical information such as age and gender, and a label tells whether the patient died during the ICU stay.

Nowadays, healthcare data are becoming predominantly unstructured (Nancy and Maheswari, 2020), with waveforms of vital signs and medical images frequently used to train machine learning models for disease prediction. To cover this class of data, first we consider the ECG Dataset, a collection of waveforms from the following three PhysioNet databases:

- MIT-BIH Arrhythmia Database (Moody and Mark, 2001, 2005),[14]
- MIT-BIH Normal Sinus Rhythm Database (Moody, 1999),[15]
- The BIDMC Congestive Heart Failure Database (Baim et al., 1986, 2000).[16]

The ECG Dataset contains data on ECGs of patients with different heart conditions: arrhythmia, normal sinus rhythm and congestive heart failure (ARR, NSR, CHF). The dataset is intended as an input file for different multi-class classification models, with the objective to classify the ECG signals according to the three labels ARR, NSR, and CHF. It contains the long-term ECG recordings of a total of 162 patients, 96 from the MIT-BIH Arrhythmia Database, 30 from the The BIDMC Congestive Heart Failure Database, and 36 rows from the MIT-BIH Normal Sinus Rhythm Database. The following pre-processing steps were applied to the raw data files of the three original databases to properly build the ECG Dataset: *(i)* retrieve the scaling from the info file and apply it to the raw data, *(ii)* re-sample all the data at a common rate of 128 hertz, *(iii)* divide each data file consisting of two ECG recordings into two different data records, *(iv)* truncate the data to a common length of 65,536 samples.

Finally, concerning medical images, we consider the Brain MRI Images Dataset[17], containing magnetic resonance imaging (MRI) scans suited for brain tumor detection with machine learning (Chakrabarty, 2019). This dataset contains a total of 253 MRI scans belonging to two classes: presence of a brain tumor (155 images), and absence of a brain tumor (98 images). Table 3 presents an overview of the datasets selected in SEASHELL tasks.

5.2 Analysis Tasks

As depicted in Fig. 8, we can broadly divide the analysis tasks contained in SEASHELL into three main groups: relational analysis over structured data (four tasks), machine learning analysis over structured data (three tasks), and machine learning analysis over unstructured data, i.e., images and waveforms (three tasks).

The relational analysis over structured data contains three tasks that execute a single large query, differing in the input file size, and one task that runs multiple queries. The machine learning analysis over structured data contains one task centered on early prediction of ICU mortality, and two tasks for predicting the onset of strokes, using different machine learning models. Finally, the set of tasks involving machine learning analysis over unstructured data contains two tasks analyzing waveforms with different machine learning models, and one task analyzing medical images by means of convolutional neural networks.

All the tasks are intended to be run in Hadoop data lake architectures similar to the one presented earlier in the paper, where HDFS is the file system responsible for managing the input files. However, except for the simple single-query relational tasks, we also define task variants suited to read the data sources directly from the local file system, to test the performance of a set of computational resources before building the actual data lake infrastructure, or to test external infrastructure that do not make use of HDFS, such as cloud-based

[12] https://kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

[13] https://physionet.org/content/challenge-2012

[14] https://physionet.org/physiobank/database/mitdb

[15] https://physionet.org/physiobank/database/nsrdb

[16] https://physionet.org/physiobank/database/chfdb

[17] https://kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection

Table 3 Overview of the datasets included in SEASHELL, specifying the contents and source of each dataset, and the prediction task for which each one is more appropriate

Dataset	Type	Contents	Source	Prediction	Reference
MIMIC-III	S	Mixed	PN	Multipurpose	Johnson et al. (2016a, b)
Stroke Prediction	S	Tabular	K	Stroke	Soriano (2021)
ICU Patients	S	Tabular	PN	ICU Mortality	Silva et al. (2012)
MIT-BIH Arrhythmia	U	Waveforms	PN	Heart Disease	Moody and Mark (2001, 2005)
MIT-BIH Normal	U	Waveforms	PN	Heart Disease	Moody (1999)
BIDMC Congestive	U	Waveforms	PN	Heart Disease	Baim et al. (1986, 2000)
Brain MRI	U	Images	K	Brain Tumor	Chakrabarty (2019)

S = Structured Data, U = Unstructured Data, K = Kaggle, PN = PhysioNet

architectures. Table 4 summarizes the main characteristics of each task contained in SEASHELL.

To conclude, analysis tasks were designed for both structured and unstructured data, with a focus on images and waveforms. In the following sections, we will describe more in detail each of these task, stressing their unique characteristics and highlighting the motivations behind their definition.

5.3 Relational Analysis

SEASHELL contains four tasks on relational analysis, characterized by different input data sizes and number of SQL queries involved. The goal is to observe variations in the resource performance according to these differences. All these tasks make use of relational tables contained in MIMIC-III clinical database. The relational analysis is performed using Spark SQL[18] with Adaptive Query Execution, an optimization technique to choose the most efficient execution plan based on run time statistics.

MII-400MB is the first task of the relational analysis class, operating on the *OutputEvents* table. The size of the input file is 400MB and the table comprises 4.35 million rows and 13 columns of life signals and medical treatments of patients, recorded during their ICU stay. This task executes an SQL query by counting the number of rows in the *OutputEvents* table with code *40055* in the column *ItemID* and an integer value > 200 in the column *Value*. The *ItemID* column from the *OutputEvents* table is a foreign key linked with the primary key from the table *D-Items*, where each row refers to a unique medical item and contains some information about it. The key 40055 is the unique ID of the Foley catheter. Therefore, the query provides the number of occurrences of a Foley catheter urine value higher than 200ml during ICU stays.

MII-800MB works similarly, by grouping the rows from the *Prescription* table based on the attribute *Drug-Type*, and then counting the number of rows within each group. The *Prescription* table consists of medical prescriptions communicated by physicians for the patient's medical treatment, comprising 4.16 million rows and 19 columns, for a size of 800MB. **MIII-2GB** processing the largest input file, i.e., the *LabEvents* table, weighting 2GB and containing 28 million rows and 9 columns of laboratory measurements. The query groups rows by *ItemID* and simply counts the number rows contained in each group.

Finally, **MIII-multi** performs multiple queries, and then it stores the results into a PostgreSQL database, to be available for future inspection or analysis. MIII-multi works with the following four MIMIC tables: *Admissions* (58,976 rows and 19 columns), *Patients* (46,520 rows and 8 columns), *Diagnoses-ICD* (65,1047 rows and 5 columns), *D-ICD-Procedures* (14,567 rows and 4 columns). The first of the three queries joins *Admissions* and *Patients*, then it counts the number of rows with admission type "emergency", grouped by gender. The second one takes the same joined table and counts the number of admissions concerning the diagnosis of sepsis, again grouped by gender. The third and last query counts the number of diagnosis regarding the thyroid, after joining the tables *Diagnoses-ICD* and *D-ICD-Procedures*.

5.4 Machine Learning on Structured Data

This class features three tasks, simulating the training phase of popular machine learning models used to support decision making in the context of healthcare. The tasks analyzed in this section and the following one are based on supervised learning methods of different natures: statistical models (e.g., logistic regression), simple classification algorithms (e.g., decision tree), and more complex ensemble methods (e.g., random forest).

First, **MLicu-LR** processes data form the ICU Patients Mortality Prediction Dataset to train a machine learning model for predicting patient deaths in intensive care units (ICU). This task is common example of decision support system for healthcare, as it allows hospitals to better manage medical resources and costs, take quick decisions for patients most at risk, and overall improve the quality of patient

[18] https://spark.apache.org/sql

Table 4 Overview of the analysis tasks featured in SEASHELL

Task Name	Data	Dataset	HDFS	Local	Type	Model	Size
MIII-400MB	Struct.	MIMIC	✓	-	Rel.	–	400MB
MIII-800MB	Struct.	MIMIC	✓	-	Rel.	–	800MB
MIII-2GB	Struct.	MIMIC	✓	-	Rel.	–	2GB
MIII-multi	Struct.	MIMIC	✓	✓	Rel.	–	30MB
MLstruct-LR	Struct.	Stroke	✓	✓	ML	LR	316KB
MLstruct-RF	Struct.	Stroke	✓	✓	ML	RF	316KB
MLicu-LR	Struct.	ICU	✓	✓	ML	LR	427KB
MLwave-LR	Unstruct.	ECG	✓	✓	ML	LR	67MB
MLwave-DT	Unstruct.	ECG	✓	✓	ML	DT	67MB
MLimg	Unstruct.	MRI	✓	✓	DL	R50	5.8GB

ICU = ICU Patients Mortality Prediction Dataset, Stroke = Stroke Prediction Dataset, ECG = ECG Dataset, MRI = Brain MRI Images Dataset, Rel. = relational analysis, ML = machine learning, DL = deep learning, DT = decision tree, LR = logistic regression, RF = random forest, R50 = ResNet50

care (Iwase et al., 2022; Bhattacharya et al., 2017). This task comprises a first pre-processing phase using Pandas, then it leverages MLlib (Meng et al., 2016), a popular library for training machine learning systems in Apache Spark environments. During the initial pre-processing phase, this task fills missing values of numerical attributes with the median value, and missing values of categorical features with the most frequent value. Numerical features are then grouped, with each group being identified by a number (e.g., age groups identified by a label from 0 to 8). The dataset is well-balanced, and no further operation on the rows is required. The resulting dataset is split into training and test sets (80/20 ratio). We employ logistic regression, a machine learning classification approach that predicts the likelihood of specific classes based on the correlation with dependent variables, such as ICU deaths with age or the reason for admission (e.g., cardiac, trauma, surgical).

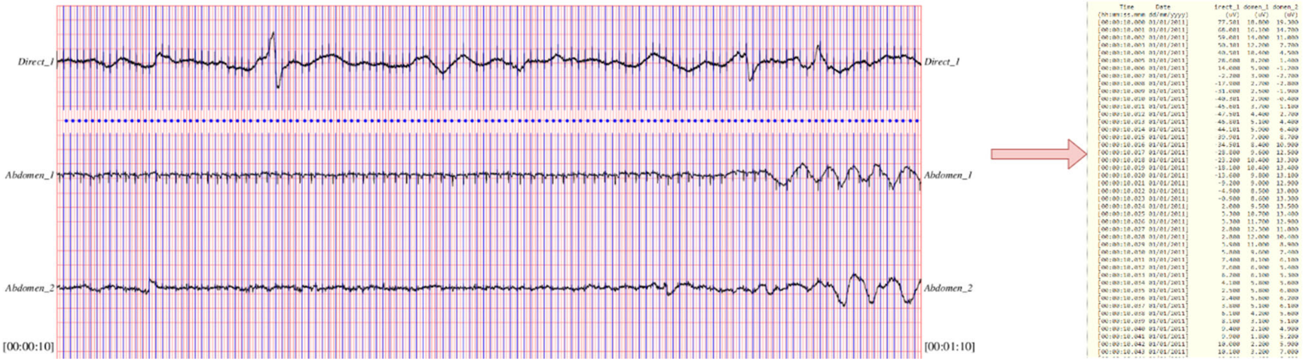
Another task with the same decision support goal in the field of healthcare is stroke risk prediction (Dritsas and Trigka, 2022; Khosla et al., 2010), i.e., predicting if a patient is or is not at risk of stroke (binary classification). The goal of the two remaining tasks **MLstruct-LR** and **MLstruct-RF** is to train a machine learning model for stroke risk prediction using the data contained in the Stroke Prediction Dataset, respectively by means of logistic regression and random forest, two popular classification algorithms. Following the usual data analysis pipeline for training machine learning models, these tasks consist in three steps: pre-processing phase, model training and testing. The dataset is mostly complete, with values missing only in the column *BMI* (i.e., body mass index). During pre-processing, missing values are filled with the median value, categorical features are grouped and assigned a numerical label per group (e.g., gender from male-female to 0-1), data type is cast from int64 to integer to conform to the standard, and data is split between training and test with the usual 80/20 ratio. The training data structure displays the current label, the expected label, and the probability of making a correct guess.

5.5 Machine Learning on Unstructured Data

This class contains two tasks related to the application of machine learning techniques to unstructured data in waveform format, and one task for managing medical images.

Applying machine learning models for heart disease classification is a common task in the literature (Alarsan and Younes, 2019; Deekshatulu et al., 2013; Khemphila and Boonjing, 2011). **MLwave-LR** and **MLwave-DT** make use of the ECG Dataset to train such a model, by making use of two different multi-class classification algorithms: multinomial logistic regression and decision tree respectively. The goal is to classify an electrocardiogram (ECG) signal into the corresponding label, detecting if a patient is suffering from some kind of heart disease. An important step for these tasks is to transform the time series contained in the ECG Dataset into a collection of features. Figure 9 shows a portion of a generic ECG signal lasting 1 minute, whereas a portion of its structured representation for the first 43ms is shown on the right. MLwave-LR and MLwave-DT employ the Wavelet Toolbox from MatLab for carrying out the feature extraction, which helps the algorithm to learn patterns and subsequently classify the signals. Three signal classes are considered: arrhythmia, congestive heart failure, and normal sinus rhythm, i.e., absence of any pathology. Each class is represented by a label (ARR, CHF, NSR) associated to each data entry. These tasks contain the usual Python and Spark initialization, a pre-processing step to process the waveforms, and the training and testing of the model.

Finally, **MLimg** features a deep learning task on medical images. In fact, the use of imaging data combined with advanced neural networks is receiving more and more attention lately in the context of disease prediction (Lundervold



```
     Time          Date              irect_1 domen_1 domen_2
(hh:mm:ss.mmm dd/mm/yyyy)              (uV)    (uV)    (uV)
[00:00:10.000 01/01/2011]            77.501  18.800  19.300
[00:00:10.001 01/01/2011]            68.801  16.100  14.700
[00:00:10.002 01/01/2011]            59.601  14.000  11.000
[00:00:10.003 01/01/2011]            50.101  12.200   7.700
[00:00:10.004 01/01/2011]            40.501  10.400   4.500
[00:00:10.005 01/01/2011]            28.600   8.200   1.400
[00:00:10.006 01/01/2011]            14.600   5.900  -1.200
[00:00:10.007 01/01/2011]            -2.200   3.900  -2.700
[00:00:10.008 01/01/2011]           -17.900   2.700  -2.800
[00:00:10.009 01/01/2011]           -31.600   2.500  -1.900
[00:00:10.010 01/01/2011]           -40.501   2.900  -0.400
[00:00:10.011 01/01/2011]           -45.601   3.700   1.100
[00:00:10.012 01/01/2011]           -47.501   4.400   2.700
[00:00:10.013 01/01/2011]           -45.801   5.100   4.400
[00:00:10.014 01/01/2011]           -44.101   5.900   6.400
[00:00:10.015 01/01/2011]           -39.901   7.000   8.700
[00:00:10.016 01/01/2011]           -34.501   8.400  10.900
[00:00:10.017 01/01/2011]           -28.800   9.600  12.500
[00:00:10.018 01/01/2011]           -23.200  10.400  13.300
[00:00:10.019 01/01/2011]           -18.100  10.400  13.400
[00:00:10.020 01/01/2011]           -13.600   9.800  13.100
[00:00:10.021 01/01/2011]            -9.200   9.000  12.900
[00:00:10.022 01/01/2011]            -4.900   8.500  13.000
[00:00:10.023 01/01/2011]            -0.500   8.600  13.300
[00:00:10.024 01/01/2011]             2.600   9.500  13.500
[00:00:10.025 01/01/2011]             3.300  10.700  13.400
[00:00:10.026 01/01/2011]             3.300  11.700  12.900
[00:00:10.027 01/01/2011]             2.800  12.300  11.000
[00:00:10.028 01/01/2011]             2.800  12.000  10.400
[00:00:10.029 01/01/2011]             3.900  11.000   8.900
[00:00:10.030 01/01/2011]             5.800   9.600   7.400
[00:00:10.031 01/01/2011]             7.400   8.100   6.100
[00:00:10.032 01/01/2011]             7.600   6.900   5.400
[00:00:10.033 01/01/2011]             6.200   6.100   5.300
[00:00:10.034 01/01/2011]             4.100   5.800   5.600
[00:00:10.035 01/01/2011]             2.500   5.800   6.000
[00:00:10.036 01/01/2011]             2.400   5.600   6.200
[00:00:10.037 01/01/2011]             3.800   5.100   6.100
[00:00:10.038 01/01/2011]             6.100   4.200   5.600
[00:00:10.039 01/01/2011]             8.100   3.100   5.100
[00:00:10.040 01/01/2011]             9.400   2.100   4.900
[00:00:10.041 01/01/2011]             9.900   1.800   5.200
[00:00:10.042 01/01/2011]            10.000   2.200   5.900
[00:00:10.043 01/01/2011]            10.100   3.200   7.000
```

Fig. 9 Structured representation of time series

and Lundervold, 2019). In particular, this task focuses on training a model for detecting the presence of tumor cells in MRI scans of the human brain, learning from the Brain MRI Images Dataset. In accordance with the goal of achieving diversity in the developed processing tasks, we decide to exploit the features of another very important Python library for machine learning applications, namely Keras,[19] with Tensorflow[20] as back-end. Since the collection of MRI scans produces a feature matrix that is too large to fully fit in memory, we trained the model incrementally on small portions of data (batches) loaded one at a time, for multiple consecutive epochs. First, MLimg leverages transfer learning (Weiss et al., 2016) to extract features from the MRI dataset using a pre-trained convolutional neural network model named ResNet50 (He et al., 2016), with a depth of 50 neural layers. Convolutional neural networks are a class of artificial neural networks commonly used to analyze visual imagery.

In compliance with the principles of transfer learning, we upload a pre-trained version of the network, trained on over one million images from the ImageNet database (Deng et al., 2009). ResNet50 is able to transform an image into a structured representation of its features, which can be later used to train the final model for disease prediction. Therefore, after extracting the features from the MRI images and assigning their corresponding label, the final classification model is trained with a stochastic gradient descent optimizer in batches of 32 images for 25 epochs.

6 Testing and Results

In this section, we describe how we tested SEASHELL on three different infrastructures for storing and analyzing data, and provide a brief discussion on the results, to demonstrate the usefulness of our benchmark. The first is the cluster architecture presented in Section 3, implemented locally according to the description provided in Section 4. The other two are cloud-based infrastructures from external providers, namely Databricks and Google.

The goal is to show the potential of SEASHELL in extracting hardware resource consumption and temporal metrics from executing the benchmark tasks on the proposed infrastructures. Moreover, SEASHELL provides useful insights both in the case of infrastructures that realize traditional multi-component data lake architectures, and for external data-management infrastructures. In the first case, the tasks read the input files from HDFS, in accordance to the architecture design previously presented, while in the second case data are loaded from a local file system.

Local Cluster Architecture

This infrastructure is based on HDFS, and implements the components previously described in this paper. The computational resources available are listed in Section 4.4. In this case, we employ Grafana, an open-source monitoring tool that provides interactive dashboards to monitor and analyze Spark cluster performance metrics and application logs.

Databricks Cloud Architecture

Cloud computing delivers computing services over the internet (Qian et al., 2009), providing on-demand access to shared resources that can be rapidly provisioned and released with minimal effort. In the healthcare industry, cloud computing is increasingly being used to support a wide range of healthcare services, including EHRs, medical imaging, clinical research, and patient monitoring (Kagadis et al., 2013; Parsonson et al., 2012; Calabrese and Cannataro, 2015). As the healthcare industry continues to adopt digital technologies, cloud computing is likely to play an increasingly important role in supporting healthcare services and improving patient outcomes. For this reason, we test a simple cloud-computing architecture composed by an Apache Spark node working as

[19] https://keras.io/api

[20] https://tensorflow.org

both Driver and Executor, hosted by the Community Edition Databricks.[21] This node is equipped with a dual core CPU, 15.3GB RAM and sufficient storage space for loading the input datasets. Moreover, the Community Edition Databricks provides a pre-installed version of Ganglia, a monitoring tool for cluster metrics that well fits with our purpose of extracting performance metrics and resource usages.

Google Colab

The last infrastructure analyzed is based on Google Colab,[22] running on Google Cloud Platform and providing a virtual machine instance for each user's session. The instances are equipped with a variety of resources, including CPU, RAM and GPU, which can be used for executing Python code, running machine learning models, and other computational-intensive tasks. The resources available while testing our benchmark with Colab included a single core Intel Xeon CPU 2.20GHz, 12.7GB of RAM, NVIDIA Tesla T4 16GB GPU, and 100GB of storage space. In order to consume the resource consumption metrics produced by task executions, we take advantage of Netdata,[23] a flexible monitoring tool for collection and visualization of performance indicators.

6.1 Metrics

The resource consumption metrics that we monitored during the execution of each task of the benchmark are the following:

- execution time,
- RAM and VRAM[24] utilization,
- CPU and GPU utilization,
- run time and CPU time.

The experiments are repeated up to ten times for each infrastructure in order to retrieve enough measurements for computing the mean execution time value of each task. By means of the aforementioned monitoring tools, we are also able to visualize a variety of plots, such as RAM and CPU utilization, which is especially useful to compare tasks and find the most resource intensive scenarios.

6.2 Training Results

In this section, we report the results of the training procedure related to the machine learning tasks, showing that the data and models involved in the benchmark are precise and accurate. For binary classification tasks (MLstruct-LR, MLStruct-RF and MLicu-LR), a compact way to represent the training results is using the accuracy, precision and area values under the receiver operating characteristic (ROC) curve. Accuracy is the ratio between the correctly predicted observations and the total number of observations:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} ,$$

where TP stands for true positive, TN for true negative, FP for false positive and FN for false negative. Precision is the ratio of correctly predicted positive observations over the total predicted positive observations:

$$Precision = \frac{TP}{TP + FP} .$$

The ROC curve is a graphical pattern that plots the true positive rate (TPR) against the false positive rate (FPR). TPR, also named Recall, or Sensitivity, is the probability that an actual positive will test positive, and FPR, also named Specificity, is the probability that an actual negative will test negative:

$$TPR = \frac{TP}{TP + FN} ,$$

$$FPR = \frac{TN}{TN + FP} .$$

Regarding the analysis tasks using waveforms (MLWave-LR and MLWave-DT), since the classification task is multi-class, we need to consider the weighted precision, i.e., the weighted mean of precision with weights equal to class probability. Moreover, the area under the ROC curve is not a good metric for multi-class problems, therefore we consider the F1 Score. F1 is the harmonic mean between Precision and Recall:

$$F1 = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} ,$$

where in our case P is the weighted precision and R the weighted recall. Table 5 presents details of the performance of each trained model.

Table 5 Model performance for each machine learning task

Type	Task	Accuracy	Precision	ROC/F1
Binary	MLstruct-LR	95.85	91.88	83.39
	MLstruct-RF	94.42	91.25	74.93
	MLicu-LR	84.77	83.62	77.33
Multi	MLwave-LR	85.46	84.12	81.90
	MLwave-DT	96.84	96.70	96.73
CNN	MLimg	93.00	-	-

CNN = Convolutional Neural Network

[21] https://www.databricks.com/product/faq/community-edition

[22] https://colab.research.google.com

[23] https://www.netdata.cloud

[24] While a computer has system RAM, most contemporary graphics cards have access to a dedicated set of memory known as Video RAM, or VRAM.

6.3 Benchmark Results

Table 6 shows the detailed results from the three single query relational analysis tasks from SEASHELL (i.e., MIII-400MB, MIII-800MB, MIII-2GB), executed on the local cluster architecture. Usage of RAM and execution time are shown with a finer granularity, for both the two executors and the driver implemented in the Spark nodes.

Table 7 displays the execution time and RAM metrics for different SEASHELL tasks executed on each of the three infrastructures described previously. Regarding the overall execution time, we recorded values ranging from around 30 seconds up to 15 minutes. Similarly, we recorded a RAM consumption ranging from around 150MB up to 1GB. This demonstrates that SEASHELL well covers a variety of tasks, both in nature and in the amount of computational power needed for their execution. Moreover, the differences in both resource consumption and execution time between different infrastructures allows to highlight important distinctions between the three. Some differences between machine learning models emerged in terms of both RAM consumption and execution time: random forest is slower and consumes more resources than logistic regression, even though we obtained similar accuracy from both models. Additionally, concerning machine learning training on waveform data, logistic regression appeared faster than decision tree, but at the expense of a higher resource consumption. It is worth noting that, as expected, the use of GPU as a computational component produces a significant reduction in the overall execution time required to complete the analysis task, but at the expense of higher resource consumption.

Finally, Fig. 10 depicts the CPU utilization graphs of four different tasks executed on the Databricks cloud architecture and plotted with Ganglia, showing how the usage of the CPU varies across time in accordance to the nature of the analysis.

6.4 Discussion

According to the experimental results, the overall execution times and resource consumption depend on various factors: input data file size, complexity of the task, underlying infrastructure.

As expected, the amount of RAM used for completing a task increases as the input file sizes increase; however, we found no evidence of a computational usage increase depending on the input file size. Machine learning tasks require more computational and storage resources with respect to relational tasks, unless relational input data critically grow in size. In fact, MIII-2GB takes way more time and resources than smaller queries or multiple ones, suggesting that multiple sequential executions on smaller sized input data is more feasible when available RAM decreases.

Some important differences among machine learning models can also be highlighted: for instance, random forest executions are slower and consume more resources than a logistic regression, despite having comparable accuracy (Table 5). Moreover, in the tasks involving waveforms, logistic regression is considerably faster than decision trees, but at the expense of lower accuracy and slightly higher resource consumption, at least in the local cluster architecture. Furthermore, the type of input data drastically influences the amount of resources needed: for instance, working with medical images (e.g., MLimg) requires way more RAM than working with structured data or waveforms. These results suggest a careful consideration of the machine learning models to execute, in view of the available resources.

Finally, it is worth noting that GPU usage in Colab results in a reduction in the time required to complete the analysis task, at the expense of a higher resource consumption. However, GPUs are more expensive than regular resources, and in real-case scenarios they might not be worth the investment, especially if medical researchers work with small amounts of data and traditional machine learning models (e.g., logistic regression). On the other hand, intense usage of deep learning models may require GPU acceleration.

7 Conclusions

The amount of medical data produced every day is a great asset for healthcare organizations, but managing and analyzing them require proper infrastructures and tools. Data lakes represent an excellent solution to this problem, but the literature lacks research on data lakes specifically designed for healthcare. Additionally, benchmarking infrastructures supporting data lake architectures is of paramount importance. In this work, we described a data lake architecture for efficient data ingestion, storage and processing of different types of data in the context of healthcare, and SEASHELL, a bench-

Table 6 Execution time and RAM metrics for both executors and driver from the execution of the single-query relational analysis tasks from SEASHELL on the local cluster architecture

Task	Avg RAM (MB)			Max RAM (MB)			Execution Time (s)
	Exec 1	Exec 2	Driver	Exec 1	Exec 2	Driver	
MIII-400MB	153	84	63	200	103	110	31
MIII-900MB	255	145	92	349	253	130	64
MIII-2GB	323	106	80	500	223	120	228

Table 7 Execution time and RAM metrics collected from the execution of SEASHELL on the three considered infrastructures

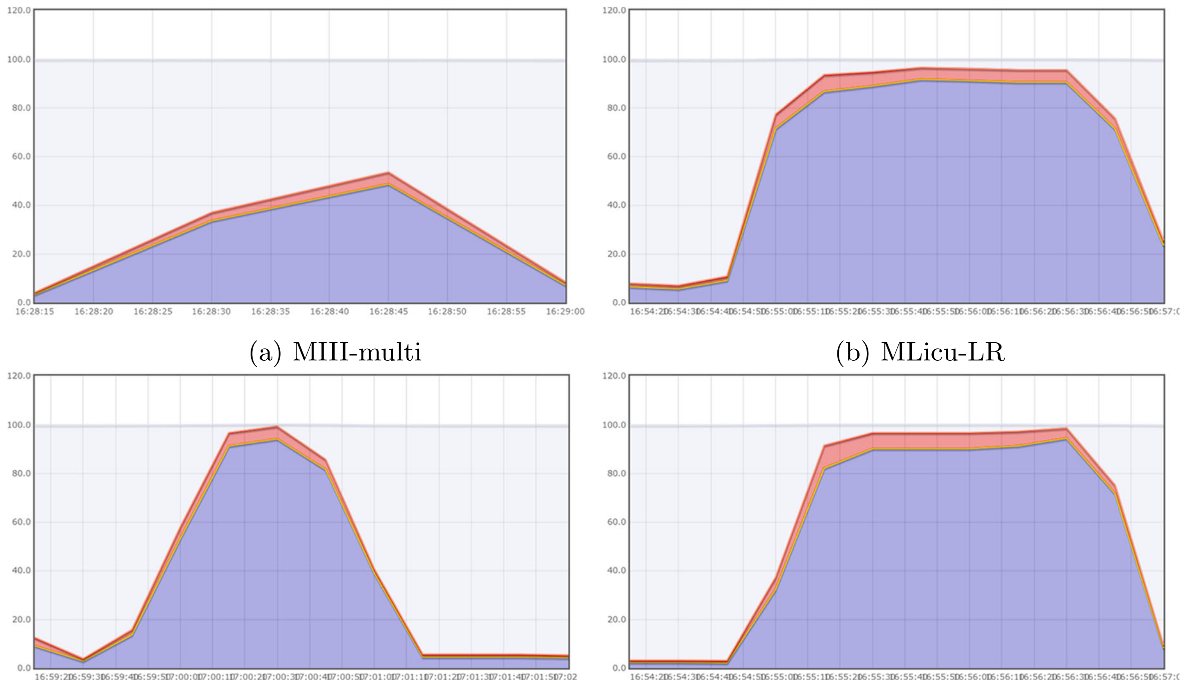
Task	Avg RAM (MB)			Max RAM (MB)			Execution Time (s)		
	HDFS	Cloud	Colab	HDFS	Cloud	Colab	HDFS	Cloud	Colab
MIII-multi	152	172	135	235	237	234	34	42	36
MLstruct-LR	259	336	436	292	560	750	96	76	69
MLstruct-RF	378	298	840	462	370	1500	660	967	559
MLicu-LR	348	345	364	445	430	436	129	97	83
MLwave-LR	300	388	455	446	520	700	76	99	55
MLwave-DT	238	411	394	297	640	700	112	110	61
MLimg	952	-	818*	1180	-	1100*	162	-	124*

HDFS = local cluster architecture implementing HDFS, Cloud = Databricks cloud architecture, Colab = Google Colab. * = leveraging the GPU, results are the following: execution time 90s, avg RAM 1.6GB, max RAM 2GB, avg VRAM 988MB, max VRAM 1045MB

mark for assessing resource consumption and execution time in healthcare data lakes involving real-world medical applications.

Our architecture allows data analysis and querying, and can manage data regardless of their generation speed or volume. Emphasis is placed on the storage of non-structured data, e.g., medical waveforms, which are seldom considered in traditional systems. Additionally, we implemented a proof-of-concept of the architecture, leveraging a variety of technologies for each of the planned components: ingestion, storage, processing, and interaction.

SEASHELL includes a variety of data analysis tasks, producing many workload scenarios. This is achieved by involving different data models (structured, unstructured) from the healthcare domain, different data formats (tabular, images, time series), different sizes of input data, and different types of analyses (relational analysis, machine learning, deep learning). All tasks implement real-world medical data applications, ranging from relational analysis to the creation of machine learning models for disease detection (Alarsan and Younes, 2019) and prediction (Dritsas and Trigka, 2022), from automated risk assessment of ICU mor-



(a) MIII-multi

(b) MLicu-LR

(c) MLwave-LR

(d) MLwave-DT

Fig. 10 CPU utilization comparison between different SEASHELL tasks on the Databricks cloud architecture

tality vIwase2022 to medical imaging analysis (Lundervold and Lundervold, 2019). To extract resource metrics, we leverage different open-source monitoring tools.

For future work, we plan to implement our data lake architecture in a real-world scenario in collaboration with hospitals and medical research centers, to test the efficiency of our solution with larger volumes of data and with more computational resources available. At the same time, it is important to test our benchmark on already existing data lake solutions from healthcare organizations, and to expand the list of analysis tasks featured. Moreover, it is worth considering the implementation and testing of the horizontal aspects described in Section 4.3, such as user access management to ensure data security, and metadata management to better organize the system and improve data access time.

Acknowledgements We are grateful to Enrico Barbierato and Giuseppe Serazzi for their advice during the definition and realization of this work, and the support in the revision of the paper.

Author Contributions All authors contributed to the definition and the design of this research. The data lake architecture was mainly created and implemented by Carlo Manco, with contributions from Tommaso Dolci, Fabio Azzalini, Marco Gribaudo and Letizia Tanca. The implementation and testing of the benchmark was mainly conducted by Lorenzo Amata, with contributions from Tommaso Dolci, Fabio Azzalini, Marco Gribaudo and Letizia Tanca. The first draft of the manuscript was written by Tommaso Dolci, Carlo Manco and Lorenzo Amata, and all authors contributed to the revision and improvement of the paper. All authors read and approved the final manuscript.

Funding This work has been partially supported by the Health Big Data Project (CCR-2018-23669122), funded by the Italian Ministry of Economy and Finance and coordinated by the Italian Ministry of Health and the network Alleanza Contro il Cancro.

Data Availability MIMIC-III Dataset is available with credentialed access on the PhysioNet website: https://physionet.org/content/mimiciii, and the MIMIC-III Waveform Database alone at: https://physionet.org/content/mimic3wdb-matched. The remaining datasets are freely available online. Stroke Prediction Dataset at: https://kaggle.com/datasets/fedesoriano/stroke-prediction-dataset. ICU Patients Mortality Prediction Dataset at: https://kaggle.com/datasets/msafi04/predict-mortality-of-icu-patients-physionet and from PhysioNet: https://physionet.org/content/challenge-2012. Brain MRI Images Dataset at: https://kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection. MIT-BIH Arrhythmia Database at: https://physionet.org/physiobank/database/mitdb. MIT-BIH Normal Sinus Rhythm Database at: https://physionet.org/physiobank/database/nsrdb. BIDMC Congestive Heart Failure Database at: https://physionet.org/physiobank/database/chfdb.

Code Availibility Code regarding the tasks included in the benchmark is available at: https://github.com/TommasoD/SEASHELL. The proof-of-concept implementation of the data lake architecture is available at: https://github.com/MancoCarlo/healer-prototype.

Declarations

Ethics Approval and Consent to Participate Not applicable.

Consent for Publication Not applicable.

Conflict of Interest Not applicable.

References

Agrahari, A., & Rao, D. (2017). A review paper on big data: technologies, tools and trends. *International Research Journal of Engineering and Technology, 4*(10), 10.

Alarsan, F. I., & Younes, M. (2019). Analysis and classification of heart diseases using heartbeat features and machine learning algorithms. *Journal of Big Data, 6*(1). https://doi.org/10.1186/s40537-019-0244-x

Alwidian, J., Rahman, S. A., Gnaim, M., et al. (2020). Big data ingestion and preparation tools. *Modern Applied Science, 14*(9), 12–27.

Baim, D. S., Colucci, W. S., Monrad, E. S., et al. (1986). Survival of patients with severe congestive heart failure treated with oral milrinone. *Journal of the American College of Cardiology, 7*(3), 661–670. https://doi.org/10.1016/S0735-1097(86)80478-8

Baim, D. S., Colucci, W. S., Monrad, E. S., et al. (2000). Bidmc congestive heart failure database. *PhysioNet*. https://doi.org/10.13026/C29G60

Bajaber, F., Sakr, S., Batarfi, O., et al. (2020). Benchmarking big data systems: A survey. *Computer Communications, 149*, 241–251. https://doi.org/10.1016/j.comcom.2019.10.002

Barbierato, E., Gribaudo, M., Serazzi, G., et al. (2021). Performance evaluation of a data lake architecture via modeling techniques. In: Performance Engineering and Stochastic Modeling. Springer, pp. 115–130.

Batini, C., Cappiello, C., Francalanci, C., et al. (2009). Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR), 41*(3), 1–52.

Beheshti, A., Benatallah, B., Nouri, R., et al. (2017). Coredb: a data lake service. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 2451–2454.

Bhattacharya, S., Rajan, V., & Shrivastava, H. (2017). Icu mortality prediction: a classification algorithm for imbalanced datasets. In: Proceedings of the AAAI Conference on Artificial Intelligence. https://doi.org/10.1609/aaai.v31i1.10721

Calabrese, B., & Cannataro, M. (2015). Cloud computing in healthcare and biomedicine. *Scalable Computing: Practice and Experience, 16*(1), 1–18.

Canham, S., Ohmann, C., Boiten, J. W., et al. (2021). *EOSC-Life Report on data standards for observational and interventional studies, and interoperability between healthcare and research data*. EOSC-Life: Tech. rep.

Cappiello, C., Gribaudo, M., Plebani, P., et al. (2022a). Enabling real-world medicine with data lake federation: A research perspective. In: VLDB Workshop on Data Management and Analytics for Medicine and Healthcare, Springer, pp. 39–56.

Cappiello, C., Gribaudo, M., Plebani, P., et al. (2022b). Enabling real-world medicine with data lake federation: A research perspective. In: VLDB Workshop on Data Management and Analytics for Medicine and Healthcare, Springer, pp. 39–56.

Chakrabarty, N. (2019). Brain mri images for brain tumor detection. https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection

Chakraborty, M., & Kundan, A. P. (2021). Grafana. In: Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software. Springer, pp. 187–240.

Chawla, N. V., & Davis, D. A. (2013). Bringing big data to personalized healthcare: a patient-centered framework. *Journal of general internal medicine, 28*(3), 660–665.

Chollet, F., et al. (2015). Keras. https://keras.io

Couto, J., Borges, O. T., Ruiz, D. D., et al. (2019). A mapping study about data lakes: An improved definition and possible architectures. In: SEKE, pp. 453–578.

Deekshatulu, B., Chandra, P., et al. (2013). Classification of heart disease using k-nearest neighbor and genetic algorithm. *Procedia technology, 10*, 85–94.

Deligiannis, K., Raftopoulou, P., Tryfonopoulos, C., et al. (2020). Hydria: An online data lake for multi-faceted analytics in the cultural heritage domain. *Big Data and Cognitive Computing, 4*(2), 7.

Deng, J., Dong, W., Socher, R., et al. (2009). Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition, Ieee, pp. 248–255.

Dritsas, E., & Trigka, M. (2022). Stroke risk prediction with machine learning techniques. *Sensors, 22*(13), 4670. https://doi.org/10.3390/s22134670

Eder, J., & Shekhovtsov, V. A. (2021). Data quality for federated medical data lakes. *International Journal of Web Information Systems, 17*(5), 407–426.

Esteva, A., Kuprel, B., Novoa, R. A., et al. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature, 542*(7639), 115–118.

Giacobbe, D. R., Signori, A., Del Puente, F., et al. (2021). Early detection of sepsis with machine learning techniques: A brief clinical perspective. *Front Med (Lausanne), 8*, 617486.

Giebler, C., Gröger, C., Hoos, E., et al. (2019). Leveraging the data lake: Current state and challenges. In: Proceedings of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK), pp. 179–188. https://doi.org/10.1007/978-3-030-27520-4_13

Giebler, C., Gröger, C., Hoos, E., et al. (2020). A zone reference model for enterprise-grade data lake management. In: 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), IEEE, pp. 57–66.

Giebler, C., Gröger, C., Hoos, E., et al. (2021). The data lake architecture framework. In: Database Systems for Business, Technology and Web (BTW). Gesellschaft für Informatik, Bonn. https://doi.org/10.18420/btw2021-19

Goldberger, A. L., Amaral, L. A., Glass, L., et al. (2000). Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation, 101*(23), e215–e220.

Gulshan, V., Peng, L., Coram, M., et al. (2016). Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama, 316*(22), 2402–2410.

Hai, R., Geisler, S., & Quix, C. (2016). Constance: An intelligent data lake system. In: Proceedings of the 2016 international conference on management of data, pp. 2097–2100.

Hamadou, H. B., Pedersen, T. B., & Thomsen, C. (2020). The danish national energy data lake: Requirements, technical architecture, and tool selection. In: 2020 IEEE International Conference on Big Data, IEEE, pp. 1523–1532.

He, K., Zhang, X., Ren, S., et al. (2016). Deep Residual Learning for Image Recognition. In: Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, CVPR '16, pp. 770–778. https://doi.org/10.1109/CVPR.2016.90

Heinis, T., & Ailamaki, A. (2017). Data infrastructure for medical research. *Found Trends Databases, 8*(3), 131–238. https://doi.org/10.1561/1900000050

Hlupić, T., Oreščanin, D., Ružak, D., et al. (2022). An overview of current data lake architecture models. *2022 45th Jubilee International Convention on Information* (pp. 1082–1087). IEEE: Communication and Electronic Technology (MIPRO).

Huang, S., Huang, J., Dai, J., et al. (2010). The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), pp. 41–51. https://doi.org/10.1109/ICDEW.2010.5452747

Hukkeri, T. S., Kanoria, V., & Shetty, J. (2020). A study of enterprise data lake solutions. *International Research Journal of Engineering and Technology (IRJET), 7.*

Inmon, B. (2016). *Data Lake Architecture: Designing the Data Lake and avoiding the garbage dump* (1st ed.). LLC, Denville, NJ, USA: Technics Publications.

Isah, H., & Zulkernine, F. (2018). A scalable and robust framework for data stream ingestion. In: 2018 IEEE International Conference on Big Data, IEEE, pp. 2900–2905.

Iwase, S., Nakada, Ta., Shimada, T., et al. (2022). Prediction algorithm for icu mortality and length of stay using machine learning. *Scientific reports, 12*(1), 12912. https://doi.org/10.1038/s41598-022-17091-5

Jagadeeswari, V., Subramaniyaswamy, V., Logesh, R., et al. (2018). A study on medical internet of things and big data in personalized healthcare system. *Health information science and systems, 6*(1), 1–20.

Johnson, A., Pollard, T., & Mark, R. (2016a) MIMIC-III clinical database. PhysioNet. https://doi.org/10.13026/C2XW26

Johnson, A., Pollard, T., Shen, L., et al. (2016). MIMIC-III, a freely accessible critical care database. *Scientific data, 3*(1), 1–9.

Kagadis, G. C., Kloukinas, C., Moore, K., et al. (2013). Cloud computing in medical imaging. *Medical physics, 40*(7), 070901.

Karthikeyan, A., Garg, A., Vinod, P. K., et al. (2021). Machine learning based clinical decision support system for early covid-19 mortality prediction. *Frontiers in Public Health, 9*. https://doi.org/10.3389/fpubh.2021.626697

Khemphila, A., Boonjing, V. (2011). Heart disease classification using neural network and feature selection. In: 2011 21st International Conference on Systems Engineering, IEEE, pp. 406–409.

Khine, P. P., & Wang, Z. S. (2018). Data lake: a new ideology in big data era. In: ITM web of conferences, EDP Sciences, p. 03025.

Khosla, A., Cao, Y., Lin, C. C. Y., et al. (2010). An integrated machine learning approach to stroke prediction. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 183–192.

Krause, J., Gulshan, V., Rahimy, E., et al. (2018). Grader variability and the importance of reference standards for evaluating machine learning models for diabetic retinopathy. *Ophthalmology, 125*(8), 1264–1272.

Kumar, P. (2023). A minimum metadata model for healthcare data interoperability. Master's thesis, Politecnico di Milano, available at https://hdl.handle.net/10589/204642

Liu, P., Loudcher, S., Darmont, J., et al. (2021). Archaeodal: A data lake for archaeological data management and analytics. In: 25th International Database Engineering & Applications Symposium, pp. 252–262.

Lundervold, A. S., & Lundervold, A. (2019). An overview of deep learning in medical imaging focusing on MRI. *Zeitschrift für Medizinische Physik, 29*(2), 102–127. https://doi.org/10.1016/j.zemedi.2018.11.002

Madera, C., & Laurent, A. (2016). The next information architecture evolution: the data lake wave. In: Proceedings of the 8th international conference on management of digital ecosystems, pp. 174–180.

Maini, E., Venkateswarlu, B., & Gupta, A. (2018). Data lake-an optimum solution for storage andanalytics of big data in cardiovascular disease prediction system. *International Journal of Computational Engineering & Management (IJCEM), 21*(6), 33–39.

Manco, C., Dolci, T., Azzalini, F., et al. (2023). HEALER: A data lake architecture for healthcare. In: Proceedings of the Workshops of the EDBT/ICDT 2023 Joint Conference, vol 3379. CEUR-WS.org.

McKinney, W., et al. (2010). Data structures for statistical computing in python. In: Proceedings of the 9th Python in Science Conference, pp. 51–56. https://doi.org/10.25080/Majora-92bf1922-00a

Meng, X., Bradley, J., Yavuz, B., et al. (2016). Mllib: Machine learning in apache spark. *The journal of machine learning research, 17*(1), 1235–1241.

Mollura, M., Mantoan, G., Romano, S., et al. (2020). The role of waveform monitoring in sepsis identification within the first hour of intensive care unit stay. In: 2020 11th Conference of the European Study Group on Cardiovascular Oscillations (ESGCO), pp. 1–2. https://doi.org/10.1109/ESGCO49734.2020.9158013

Moody, B., Moody, G., Villarroel, M., et al. (2020). MIMIC-III waveform database matched subset. PhysioNet. https://doi.org/10.13026/c2294b

Moody, G. (1999). MIT-BIH normal sinus rhythm database. PhysioNet. https://doi.org/10.13026/C2NK5R

Moody, G., & Mark, R. (2001). The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine, 20*(3), 45–50. https://doi.org/10.1109/51.932724

Moody, G., & Mark, R. (2005). MIT-BIH arrhythmia database. PhysioNet. https://doi.org/10.13026/C2F305

Nancy, A. M., & Maheswari, R. (2020). A review on unstructured data in medical data. *J Crit Rev, 7*, 2202–2208.

Parsonson, L., Grimm, S., Bajwa, A., et al. (2012). A cloud computing medical image analysis and collaboration platform. In: Cloud Computing and Services Science, Springer, pp. 207–224.

Prasser, F., Kohlbacher, O., Mansmann, U., et al. (2018). Data integration for future medicine (DIFUTURE). *Methods Inf Med, 57*(S 01), e57–e65

Qian, L., Luo, Z., Du, Y., et al. (2009). Cloud computing: An overview. In: Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1, Springer, pp. 626–631.

Raghupathi, W., & Raghupathi, V. (2014). Big data analytics in healthcare: promise and potential. *Health Information Science and Systems, 2*(1). https://doi.org/10.1186/2047-2501-2-3

Ravat, F., & Zhao, Y. (2019). Data lakes: Trends and perspectives. In: International Conference on Database and Expert Systems Applications, Springer, pp. 304–313.

Ren, P., Li, S., Hou, W., et al. (2021). Mhdp: an efficient data lake platform for medical multi-source heterogeneous data. In: Web Information Systems and Applications: 18th International Conference, WISA 2021, Kaifeng, China, September 24–26, 2021, Proceedings 18, Springer, pp. 727–738.

Rieke, N., Hancox, J., Li, W., et al. (2020). The future of digital health with federated learning. *npj Digital Medicine, 3*(1). https://doi.org/10.1038/s41746-020-00323-1

Sawadogo, P., & Darmont, J. (2021). Benchmarking data lakes featuring structured and unstructured data with dlbench. *Big Data Analytics and Knowledge Discovery* (pp. 15–26). Cham: Springer International Publishing.

Sawadogo, P., & Darmont, J. (2021). On data lake architectures and metadata management. *Journal of Intelligent Information Systems, 56*(1), 97–120.

Sha, M.M., & Rahamathulla, M. P. (2020). Cloud-based healthcare data management framework. *KSII Transactions on Internet and Information Systems (TIIS), 14*(3), 1014–1025.

Silva, I., Moody, G., Scott, D. J., et al. (2012). Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In: 2012 Computing in Cardiology, IEEE, pp. 245–248.

Soriano, F. (2021). Stroke prediction dataset. https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

SPEC. (2017). SPEC CPU Benchmarks. https://www.spec.org/cpu/. Accessed 24 Mar 2023.

Taher, N. C., Mallat, I., Agoulmine, N., et al. (2019). An iot-cloud based solution for real-time and batch processing of big data: Application in healthcare. In: 2019 3rd international conference on bio-engineering for smart technologies (BioSMART), IEEE, pp. 1–8.

Transaction Processing Performance Council. (2021). TCPx-HS benchmark specification. Specification 1.0, Transaction Processing Performance Council. https://www.tpc.org/tpcx-hs/

Truică, C. O., Apostol, E. S., Darmont, J., et al. (2020). TextBenDS: a generic textual data benchmark for distributed systems. *Information Systems Frontiers, 23*(1), 81–100. https://doi.org/10.1007/s10796-020-09999-y

Walker, C., & Alrehamy, H. (2015). Personal data lake with data gravity pull. In: 2015 IEEE Fifth International Conference on Big Data and Cloud Computing, IEEE, pp. 160–167.

Wang, L., Zhan, J., Luo, C., et al. (2014). Bigdatabench: A big data benchmark suite from internet services. In: 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 488–499. https://doi.org/10.1109/HPCA.2014.6835958

Weber, G. M., Murphy, S. N., McMurry, A. J., et al. (2009). The shared health research information network (shrine): a prototype federated query tool for clinical data repositories. *Journal of the American Medical Informatics Association, 16*(5), 624–630.

Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big data, 3*(1), 1–40.

Xin, R. (2014). Apache spark officially sets a new record in large-scale sorting. https://www.databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html. Accessed 11 July 2023.

Zaharia, M., Xin, R. S., Wendell, P., et al. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM, 59*(11), 56–65.

Tommaso Dolci is a PhD Researcher with Lero, the Science Foundation Ireland Research Centre for Software at the University of Limerick. He holds a master's degree in Computer Science and Engineering from Politecnico di Milano, Italy. Previously, he worked as Research Assistant in Politecnico di Milano. His research publications appeared on multiple journals and international conference proceedings. His research interests lie at the intersection of responsible engineering, AI technologies and computer ethics, with a focus on fairness in NLP, algorithmic bias and digital humanism.

Lorenzo Amata achieved his master's degree in Computer Science and Engineering at Politecnico di Milano, Italy. During his final thesis, he was part of the databases group of the Department of Electronics, Information and Bioengineering. His research thesis focused on benchmarking data lake infrastructures for healthcare, specifically regarding advanced Big Data analytics.

Carlo Manco holds a master's degree in Computer Science and Engineering from Politecnico di Milano, Italy. During his final thesis, he was part of the databases group of the Department of Electronics, Information and Bioengineering, focusing his research on data lake architecture deployment and optimisation, in the specific context of healthcare.

Fabio Azzalini is a Postdoctoral Researcher at the Department of Electronics, Information and Bioengineering of Politecnico di Milano, Italy. He holds a PhD in Data Analytics and Decision Sciences from Politecnico di Milano, where he previously graduated in Computer Science and Engineering. He has published in multiple international scientific journals and conferences on databases and data science (e.g., VLDB Journal, JDIQ, CIKM, ICDE, DASFAA). His research focuses mainly on data integration and computer ethics.

Marco Gribaudo is an Associate Professor at the Department of Electronics, Information and Bioengineering of Politecnico di Milano, Italy. He works in the performance evaluation group. His current research interests are multi-formalism modelling, queueing networks fluid models, mean field analysis and spatial models. The main applications to which the previous methodologies are applied comes from Big Data applications, cloud computing, multi-core architectures and wireless sensor networks.

Letizia Tanca is a Full Professor at the Department of Electronics, Information and Bioengineering of Politecnico di Milano, where she has held several institutional positions. She is the author of about 200 publications on databases, context-aware management of mobile data, semantic information and Big Data analysis. She has long been reviewer for top international DB journals and conferences and is currently Associate Editor of PVLDB 2023 and Program Chair of EDBT 2024. She has acted for 6 years in the Board of the association Informatics Europe, and more recently as Vice-President of the GII (Italian Group of Computer Engineering Academics). She coordinates her department's group of the Health Big Data Project.