GEN-RWD Sandbox ecosystem for privacy-preserving data sharing in healthcare research: the Processor module

Benedetta Gottardelli
Dip. di Diagnostica per Immagini,
Radioterapia Oncologica ed Ematologia
Università Cattolica del Sacro Cuore
Rome, Italy

Roberto Gatta
Dip. di Scienze Cliniche e Sperimentali
Università degli Studi di Brescia
Brescia, Italy

Leonardo Nucciarelli
Dip. di Diagnostica per Immagini,
Radioterapia Oncologica ed Ematologia
Università Cattolica del Sacro Cuore
Rome, Italy

Mariachiara Savino
Dip. di Diagnostica per Immagini,
Radioterapia Oncologica ed Ematologia
Università Cattolica del Sacro Cuore
Rome, Italy

Andrada Mihaela Tudor
Dip. di Diagnostica per Immagini,
Radioterapia Oncologica ed Ematologia
Università Cattolica del Sacro Cuore
Rome, Italy

Mauro Vallati
School of Computing and Engineering
University of Huddersfield
Huddersfield, UK

Andrea Damiani
Gemelli Generator Real World Data
Fondazione Policlinico Universitario Agostino Gemelli (IRCCS)
Rome, Italy

***Abstract*—Recent innovations in computer science and informatics are driving the integration of AI into modern healthcare, extending its applications to medical sectors previously reliant on human expertise. Creating robust and clinically relevant AI models requires extensive data, which can be challenging to gather, particularly when dealing with rare diseases. Data sharing among healthcare entities can address this issue, but legal, privacy, and data ownership concerns hinder such approach.**

To foster data sharing, in this paper we propose the GEmelli GeNerator – Real World Data (GEN-RWD) Sandbox, that provides a secure environment for data analysis without compromising sensitive medical data. This modular architecture serves as a research platform for various stakeholders, including clinical researchers, policymakers, and pharmaceutical companies. Authorized users submit research requests through the GUI, which are processed within the hospital, and the results can be accessed without revealing the original clinical data source. In the context of this paper we present GEN-RWD Sandbox's architecture module in charge of executing the analysis requests, the Processor. Processor's code is openly shared as the GSProcessor R package available at https://gitlab.com/benedetta.gottardelli/GSProcessor.

***Index Terms*—Privacy-preserving data sharing, Distributed analytics, Open source, GEN-RWD Sandbox, GSProcessor.**

I. Introduction

Recent advancements in computer science and informatics are leading to the integration of artificial intelligence in modern healthcare. The use of artificial intelligence is rapidly expanding to many medical sectors previously considered to be reserved to human expertise [1]. In order to create robust and clinically relevant AI models, it is necessary to leverage on large amount of data; this becomes challenging, especially when exploring rare diseases. Sharing clinical data between owners could ease these obstacles, but this is not an actionable road in healthcare due to legal aspects, privacy and data ownership issues. Consequently, multiple methods and systems have been introduced to share data while ensuring the full compliance with privacy regulations and legal requirements. Well-known examples include Distributed data analysis, Secure multi-party computation [2] and Homomorphic Encryption. In recent years, Distributed data analytics has gained popularity due to its capability to conduct data analysis in a distributed fashion, enabling multi-party cooperation in collaborative research effort without requiring data relocation [3]. In specific scenarios, distributed learning is preferred over computationally intensive methods like the latters [4], and it has gained popularity as it allows to train high-quality global models through a central server and decentralized data. Distributed data analysis requires trusted infrastructures and frameworks that allow the performance of analysis requests to be coordinated without the data being shared. In this paper we discuss the GEmelli GeNerator – Real World Data (GEN-RWD) Sandbox, a modular architecture that provides a secure environment where safe interactions with data can be established [5]. A *sandbox* is a controlled and safe environment to experiment with data analysis without risks of leakage of sensitive medical data. The architecture is designed to serve as a research experimentation platform for medical research stakeholders, from non-technicians to programmers, such as clinical researchers, policy makers and pharmaceuticals in a distributed fashion [6]. GEN-RWD Sandbox is designed as a 3-module architecture: Graphical User Interface (GUI), Proxy, and Processor. From a high-level perspective, authorized users can submit their research requests through a GUI, which are then processed within the hospital. Browsing the GUI, users can select both a datamart and an algorithm to perform the desired analysis. Once the analysis is performed, users can view and download the resulting report, from which it is not possible to trace back the original clinical data. The Processor is the central component of the platform, and it is a flexible task executor that performs the analysis by means of R and Python scripts, or Docker images. The Proxy module manages the interaction between the GUI and the Processor, and its tasks are threefold: to check an monitor the GUI for new tasks, to dispatch the logs produced by the Processor to the GUI, and to forward the output results to the GUI. To ensure data ownership and privacy, the Proxy and the Processor are located on the same computer network within the hospital premises, and the communication between these two modules is allowed asynchronously through the filesystem. The Processor module can run on an isolated system governed by the hospital's internal security policies. The only module located outside the hospital walls is the GUI, and the Proxy communicates with the GUI through pool requests via HTTP. In this paper we focus on the Processor in detail, exploring its main functionalities and functions, and a sample analysis on real healthcare data.

II. Processor general description

The Processor module is the cornerstone component of the GEN-RWD Sandbox, responsible for executing tasks according to a black box model that relies on a minimal core structure (Figure 1). This structure comprises a token, an input folder, the processing unit, and an output folder.

The Processor is associated with one or more input folders and initiates the execution of a task when a token is inserted in one of them (Figure 1A). While processed, the token is removed from the input folder thus allowing any other accumulated tokens to be evaded when the processing is finished. Upon completion of the computation, the Processor generates an output token, which is deposited into the output folder (Figure 1B). Execution failures are handled by the Processor modules and all the related logs are forwarded to upper layers. This process is the core foundation of the infrastructure and is essential to the successful execution of any task or computation within the system.

A. Token

The Processor module is an automatic task executor. A task request in our framework is represented by a *token* which is designed to be a collection of zipped files. The token includes details of the algorithm to be used, such as hyperparameters or the script file, and the data to be processed. A token must contain at least one XML describing the task to run and the contents of the token itself, referred to as *XML Descriptor*. Additional data or processing files that would be needed for the task, can be included in the token in order for them to be processed.

XML Descriptor. This is an XML file that lists the content of the token and provides instructions on how the token is to be processed. Inside the token is the XML Descriptor file is called "description.xml" (all lowercase). It consists of the following sections:

- XMLheader: where the date of creation of the token, the user, the nature of the token, how it is to be processed, etc. is declared.
- XMLobj: where the list of the elements zipped as part of the token is provided, indicating the nature and intended use of such elements.

In the *XMLheader* section, a Token instance Unique Identifier (tokenIstanceUID tag), a Run Unique Identifier (runUID tag) and a creation datetime (creationDateTime tag) must be specified. The *tokenInstanceUID* uniquely identifies the token, while the *runID* uniquely identifies the token run that was launched.

XMLheader section example

```
<XMLheader>
    <tokenInstanceUID>421</tokenInstanceUID>
    <runUID>0</runUID>
    <creationDateTime>10/31/2023, 12:04:50
    </creationDateTime>
</XMLheader>
```

Within an XML Descriptor there could be multiple *XMLobj* sections. An XMLobj could be of different types, i.e. *dataSource*, *script*, *run* or *other*, according to the described object.

A *dataSource XMLobject*, describing data to be used by the process, must contain a *dataSourceType* tag where a data source is specified, a *dataSourceFileName* tag where the data identifier within the source is specified and an alias for the dataset itself (*alias* tag).

dataSource XMLobject section example

```
<XMLobj objType='dataSource'>
    <dataSourceType>csv</dataSourceType>
    <dataSourceFileName>data.csv</dataSourceFileName>
    <alias>data01</alias>
</XMLobj>
```

A *script XMLobject* includes details of the script to be executed, and where to find it. As the processor can run both attached script files and docker images, in the *scriptType* tag it is mandatory to specify either "docker" or "script". If the script type is "docker" then a *dockerImageName* tag must be filled with the name of the docker image to be run. If the script type is "script" then a corresponding programming language, i.e. "R" or "Python", must be specified in *scriptProgrammingLanguage* tag. Additionally, the *scriptFileName* tag must provide the script file name. For both script types, an object alias must be provided in the dedicated *alias* tag.

script XMLobject section example for Docker images

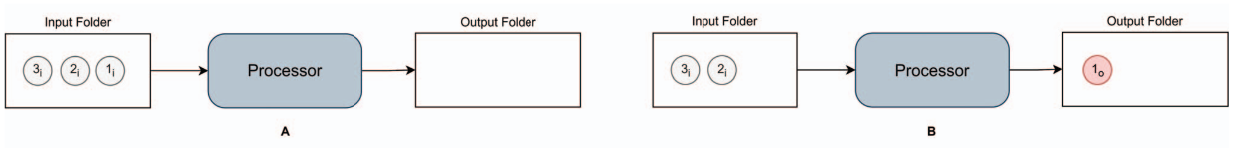

Fig. 1. The fundamental architecture of the Processor module. The Processor monitors one or more input folders, and upon the post of a token into an input folder, it processes and executes the script contained within the token (**A**). The Processor then reports the result of the execution to the designated output folder in the form of a new token (**B**).

```
 <XMLobj objType='script'>
    <scriptType>docker</scriptType>
    <dockerImageName>clustering</dockerImageName>
    <alias>clustering</alias>
</XMLobj>
```

script XMLobject section example for script files

```
<XMLobj objType='script'>
    <alias>clustering</alias>
    <scriptType>script</scriptType>
    <scriptProgrammingLanguage>Python
    </scriptProgrammingLanguage>
    <scriptFileName>clustering.py</scriptFileName>
</XMLobj>
```

Another XMLobj can be used to describe any additional resources stored in the token. In the event, for instance, that a python script is placed in the token, the package file required for execution (”requirements.txt”) can also be provided. When such a file is provided, the processor creates an ad hoc python environment for execution of the script, otherwise the script is executed in the general environment of the machine where the processor is installed.

other XMLobject section example

```
<XMLobj objType='other'>
    <alias>requirementFile</alias>
    <filename>requirements.txt</filename>
</XMLobj>
```

A *run XMLobj* describes the nature of the association between the script and the data source(s) and, in addition, provides instructions on how processing should take place for that specific run.

run XMLobject section example

```
<XMLobj objType='run'>
    <alias>01</alias>
    <scriptAlias>clustering</scriptAlias>
    <otherAlias>requirementFile</otherAlias>
    <dataSourceAlias>data01</dataSourceAlias>
</XMLobj>
```

B. Processing unit

The Processing unit of the Processor module is an open source R package, named GEN-RWD Sandbox - Processor (GSProcessor), available as GitLab repository at https://gitlab.com/benedetta.gottardelli/GSProcessor.git. The R package is compatible with R (v4.0.4) and Linux OS.

The package consists of a single main function, ’bckGndProcessor()’, which launches, in the background on the host machine, an instance of a Processor module. The input arguments of this function comprise paths of Processor’s folders (Figure 2) and execution parameters. The detailed meanings of these function options are as follows.

`input_folder.dir`: A string containing a physical location on the file-system accessible by the GSProcessor. A GSProcessor instance continuously monitors its input_folder and, when a token is deposited, it takes the token, opens it, and processes it in accordance with its contents. In doing so, it removes the token from the input_folder.

`output_folder.dir`: A string containing a physical location on the file-system accessible by the GSProcessor. It serves as the destination folder where the processor stores the computation results in the form of tokens. As the result of the computation itself is considered a token, it contains an XML file for its identification, along with other specified output files.

`tmp_folder.dir`: A string containing a physical location on the file-system accessible by the GSProcessor. It indicates a local folder that the processor uses for temporary storage of files. It is typically cleaned before processing a new token or when it has finished processing one.

`sync_folder.dir`: A string containing a physical location on the file-system accessible by the GSProcessor. It is a local folder that the processor uses for storing logs and handling parallel execution of multiple tokens.

`override.repeated.tUID`: Boolean value. If `TRUE` if two processes have the same unique ID, temporary files and logs are overwritten. By default it is set to `FALSE`.

`stopAfterExecution`: Integer number. Number of processes handled after which the GSProcessor instance stops. By default it is set to `Inf`. This parameter is useful when you don’t want the GSProcessor to run infinitely but to run a specific number of processes and then to be stopped.

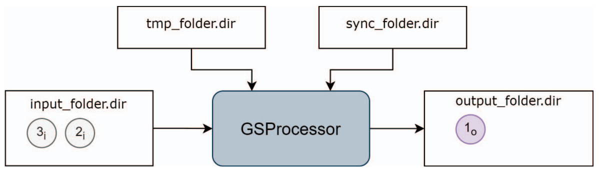

Fig. 2. Extended architecture of a GG-RWD Sandbox Processor (GSProcessor) instance.

The Processor module, encapsulated in the GSProcessor R package, is designed as a flexible task executor. It is indeed able to run not only R and Python scripts attached to the input token but also Docker images. Moreover, the Processor is able to process tasks in parallel launching, through system calls, each execution in background and keeping track to each one's log.

C. GEN-RWD Sandbox integration and load balancing

The Processor module is the cornerstone of the GEN-RWD Sandbox, as it is in charge of actually executing user's requests. User requests are collected by the GUI module, whose task is to interact with the users. The GUI, being located on the internet and highly exposed, does not interact with data that must remain within the hospital walls, but passes on the request to underlying module, the Proxy (Figure 3). One of Proxy's main tasks is to communicate with the Processor module, via file-system. In fact, the Proxy forwards user requests to the Processor module by creating the corresponding input token and dropping it in the Processor's input folder. By constantly monitoring the Processor's output and sync folder, the Proxy returns to the GUI output results and process logs. The Proxy module enables the processor to execute a predetermined list of permitted scripts contained within Docker images. This functionality ensures, within the Sandbox ecosystem, that only scripts deemed safe for the privacy of data and other resources on the host machine are executed. The list is administered at the Proxy level, and therefore, by the Sandbox manager, which is the data provider.

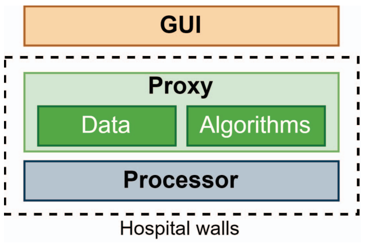

Fig. 3. GEN-RWD Sandbox's general architecture. The data is located and processed within the hospital walls, all that goes outside are aggregated results from which it is not possible to trace back to the given patient. In this way, the privacy and ownership of the data is maintained.

The modules of the GEN-RWD Sandbox can be combined in various configurations. Specifically, the Proxy module can handle multiple instances of the Processor (Figure 4A) and at the same time also multiple GUIs (Figure 4B). The latter configuration is meant for having customized GUI for different kind of users while the former is designed to further parallelize and balance the computational workload that may result from high request traffic from the GUI(s). The Proxy monitors the number of forwarded, completed, and ongoing requests for each instance of the Processor and assigns a new task to the processor with the least load. The Proxy is also able to evaluate if the task requires special computation resources (i.e. a GPU) and forward it to the Processor instance located to a machine that meets the specific requirements.

III. Introductory example using rectal patient data

We are now in the position to present an example of how to run an analysis with the GEN-RWD Sandbox Processor module. The example reproduces the process mining analysis presented in [7]. The aim of the analysis conducted by Savino et al. in [7] was to evaluate the adherence of a real-word cohort of rectal cancer patients to the European Society of Medical Oncology (ESMO) guidelines using pMineR software [8].

1) Analysis: Data included $1,895$ events referring to 453 non-metastatic rectal cancer patients treated at the Fondazione Policlinico Universitario A. Gemelli IRCCS, between January 2017 and December 2021. The ESMO guidelines [9], published in 2017, propose specific recommendations for rectal cancer treatment according to the TNM staging risk groups (early, intermediate, locally advanced and advanced): *a)* Neoadjuvant chemoradiotherapy followed by TEM surgery or conservative approach (Long-course branch). *b)* Neoadjuvant radiotherapy at 25 Gy followed by TME surgery (Short-course branch). *c)* Neoadjuvant radiotherapy at 25 Gy followed by Folfox chemotherapy and TME surgery. The example uses Pseudo-Workflow, a formalism in pMineR software, to translate clinical guidelines into a computer-interpretable format. This method describes guidelines' recommendations in terms of nodes and rules, updating patient status based on condition. The pMineR software engine generates a workflow diagram, comparing the real event logs to the Pseudo-Workflow version.

2) Token preparation & Processor execution launch: We generated a token folder where we collected the data in csv format, the algorithm script file and the XML Descriptor compiled as below.

XML Descriptor

```
<xml>
  <XMLheader>
    <tokenInstanceUID>0001</tokenInstanceUID>
    <runUID>0001</runUID>
    <creationDateTime>2023-10-17</creationDateTime>
  </XMLheader>
  <XMLobj objType='dataSource'>
    <dataSourceType>csv</dataSourceType>
    <dataSourceFileName>rectalCancerEventLogs.csv
    </dataSourceFileName>
    <alias>EventLogs</alias>
    </XMLobj>
  <XMLobj objType='script'>
    <alias>ConfChecking</alias>
    <scriptType>script</scriptType>
    <scriptProgrammingLanguage>R
    </scriptProgrammingLanguage>
    <scriptFileName>ConformanceChecking.r
    </scriptFileName>
    <processorConformanceClass>R
    </processorConformanceClass>
  </XMLobj>
  <XMLobj objType='run'>
    <alias>0001</alias>
    <scriptAlias>ConfChecking</scriptAlias>
    <dataSourceAlias>EventLogs</dataSourceAlias>
  </XMLobj>
```

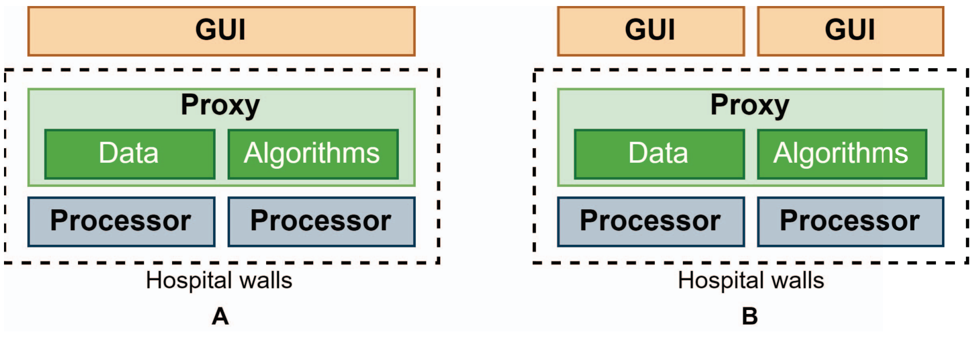

Fig. 4. GEN-RWD Sandbox's possible configurations.

```
</xml>
```

The Conformance Checking algorithm was encapsulated in an R script (*ConformanceChecking.r*) that we adapted for compatibility with the Processor, as described in the "token-Templates" folder in the GitLab repository. Once the token was complete, we zipped it and launched a GSProcessor instance using:

```
library(GSProcessor)

objS <- bckGndProcessor(
            input_folder.dir = "./inputFolder/",
            output_folder.dir = "./outputFolder/",
            tmp_folder.dir = "./tmpFolder/",
            sync_folder.dir = "./syncFolder/",
            override.repeated.tUID = TRUE
            )

objS$start()
```

The token was then dropped into $./inputFolder/$ and the execution started.

3) Results: Figures 5 shows the report obtained in the GSProcessor output folder. The results indicate the same adherence to the ESMO guidelines as described in the previous work. In the 'Early' risk group only 11 (23.91%) patients underwent TME surgery and were compliant to the ESMO guidelines (Fig. 5A). The 'Intermediate' risk group presents 20 (18.18%) patients fully compliant to the ESMO guidelines following the short-course, long-course and TME surgery branches (Fig. 5B). In the 'Locally Advanced' risk group 5 (7.94%) patients completed the short-course or the long-course branches as prescribed by the ESMO guidelines (Fig. 5C), and finally, the 'Advanced' risk group has the highest percentage of non-compliant patients with only 4 (1.71%) patients fully adhering to the guidelines (Fig. 5D).

More token examples with synthetic data are publicly available in the "tokenExamples" folder of the GitLab repository.

IV. Discussion

We presented the code and functionality of the Processor module within the GEN-RWD Sandbox architecture, a platform designed to facilitate privacy preserving data sharing between hospitals and external entities. The transparency and open-source nature of this module, which is made available through the R package GSProcessor, is instrumental in promoting trust and collaboration in the medical data-sharing domain. The Processor module stands out as a versatile task executor that possesses the capability to seamlessly execute both R and Python scripts, in addition to handling Docker images. This inherent flexibility plays a pivotal role in enhancing the overall adaptability of the GEN-RWD sandbox, ensuring that it can cater to a wide array of user requirements. At its core, the architecture of the Processor module comprises a processor unit and a few folders. This simplicity in design not only makes it easy to integrate at higher levels but also renders it highly adaptable for various purposes. Hospitals, in particular, stand to benefit from the Processor module's ability to automate data extraction and data quality pipelines, especially in environments where data are collected on a daily basis. One critical consideration for the Processor module is load balancing, as it plays a pivotal role in ensuring that computational requests are handled efficiently and effectively. Recognizing the significance of this aspect, we our future work will focus on enhancing the Processor module by exploring the possibility of adapting the package to support distributed computing. This development would be a game-changer, enabling the platform to efficiently distribute computational tasks across multiple local resources, thereby optimizing performance and scalability.

References

[1] Yu, KH., Beam, A.L. & Kohane, I.S. Artificial intelligence in healthcare. *Nat Biomed Eng* 2, 719–731 (2018). https://doi.org/10.1038/s41551-018-0305-z
[2] Wirth, F.N., Meurers, T., Johns, M. *et al.* Privacy-preserving data sharing infrastructures for medical research: systematization and comparison. *BMC Med Inform Decis Mak* 21, 242 (2021). https://doi.org/10.1186/s12911-021-01602-x
[3] Xu, J., Glicksberg, B. S., Su, C., Walker, P., Bian, J., & Wang, F. (2021). Federated Learning for Healthcare Informatics. *Journal of healthcare informatics research*, *5*(1), 1–19. https://doi.org/10.1007/s41666-020-00082-4
[4] Hamza, R., Hassan, A., Ali, A., Bashir, M.B., Alqhtani, S.M., Tawfeeg, T.M., Yousif, A. Towards Secure Big Data Analysis via Fully Homomorphic Encryption Algorithms. *Entropy*. 2022; 24(4):519. https://doi.org/10.3390/e24040519

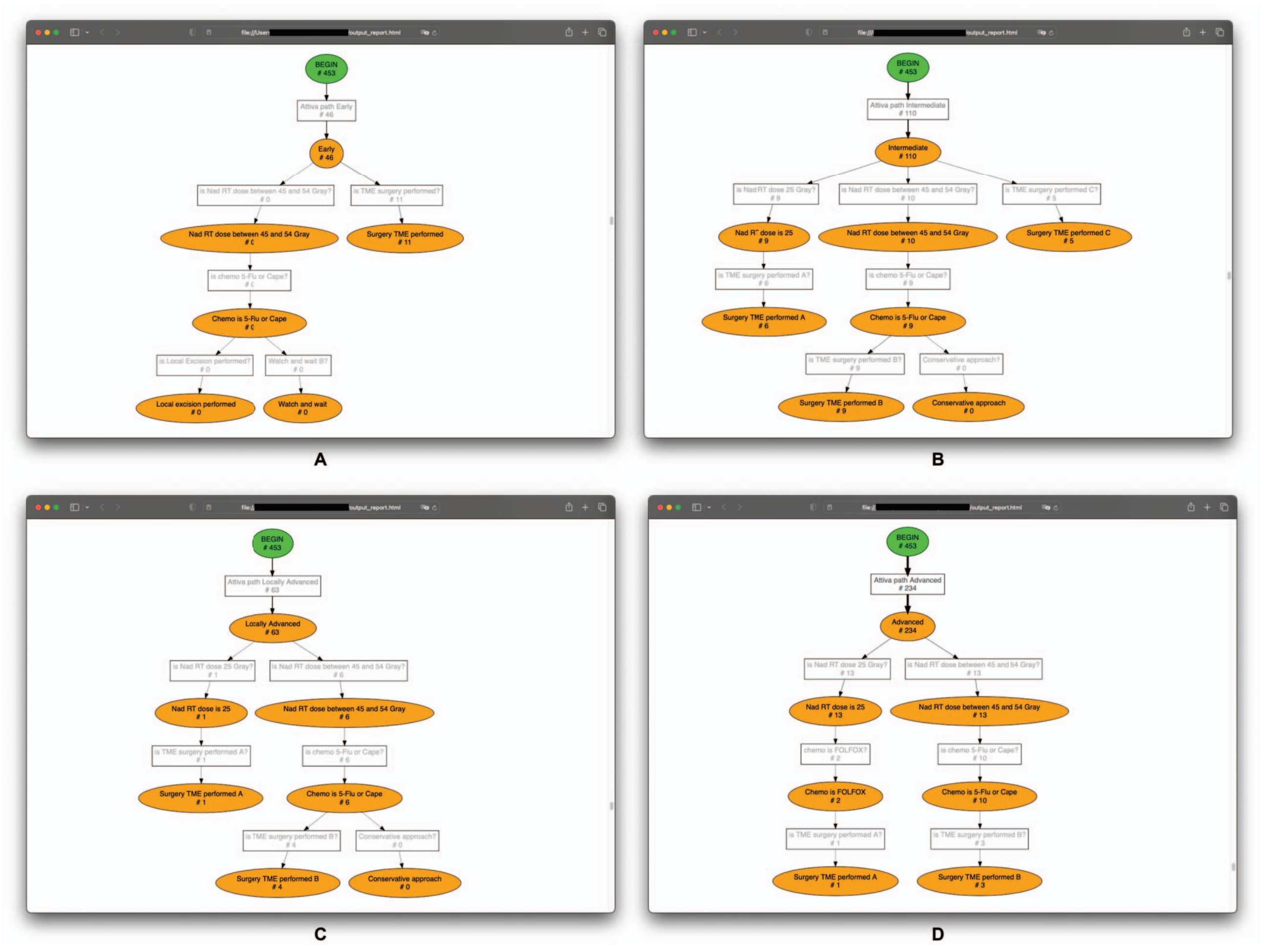
file://Users[redacted]output_report.html
BEGIN # 453
Attiva path Early # 46
Early # 46
is Nad RT dose between 45 and 54 Gray? # 0
is TME surgery performed? # 11
Nad RT dose between 45 and 54 Gray # 0
Surgery TME performed # 11
is chemo 5-Flu or Cape? # 0
Chemo is 5-Flu or Cape # 0
is Local Excision performed? # 0
Watch and wait B? # 0
Local excision performed # 0
Watch and wait # 0
A
file:///[redacted]output_report.html
BEGIN # 453
Attiva path Intermediate # 110
Intermediate # 110
is Nad RT dose 25 Gray? # 9
is Nad RT dose between 45 and 54 Gray? # 10
is TME surgery performed C? # 5
Nad RT dose is 25 # 9
Nad RT dose between 45 and 54 Gray # 10
Surgery TME performed C # 5
is TME surgery performed A? # 6
is chemo 5-Flu or Cape? # 9
Surgery TME performed A # 6
Chemo is 5-Flu or Cape # 9
is TME surgery performed B? # 9
Conservative approach? # 0
Surgery TME performed B # 9
Conservative approach # 0
B
file://[redacted]output_report.html
BEGIN # 453
Attiva path Locally Advanced # 63
Locally Advanced # 63
is Nad RT dose 25 Gray? # 1
is Nad RT dose between 45 and 54 Gray? # 6
Nad RT dose is 25 # 1
Nad RT dose between 45 and 54 Gray # 6
is TME surgery performed A? # 1
is chemo 5-Flu or Cape? # 6
Surgery TME performed A # 1
Chemo is 5-Flu or Cape # 6
is TME surgery performed B? # 4
Conservative approach? # 0
Surgery TME performed B # 4
Conservative approach # 0
C
file://[redacted]/output_report.html
BEGIN # 453
Attiva path Advanced # 234
Advanced # 234
is Nad RT dose 25 Gray? # 13
is Nad RT dose between 45 and 54 Gray? # 13
Nad RT dose is 25 # 13
Nad RT dose between 45 and 54 Gray # 13
chemo is FOLFOX? # 2
is chemo 5-Flu or Cape? # 10
Chemo is FOLFOX # 2
Chemo is 5-Flu or Cape # 10
is TME surgery performed A? # 1
is TME surgery performed B? # 3
Surgery TME performed A # 1
Surgery TME performed B # 3
D

Fig. 5. GEN-RWD Sandbox's output report of the conformance checking analysis for Early (A), Intermediate (B), Locally Advanced (C) and Advanced (D) risk groups.

[5] Gottardelli, B., Gatta, R., Nucciarelli, L., et al. GEN-RWD Sandbox: Bridging the Gap Between Hospital Data Privacy and External Research Insights with Distributed Analytics, 28 December 2023, PREPRINT (Version 1) available at Research Square [https://doi.org/10.21203/rs.3.rs-3816282/v1]

[6] European Parliament and Council of the European Union, “Regulation (EU) 2016/679 of the European Parliament and of the Council.” [Online]. Available: https://data.europa.eu/eli/reg/2016/679/oj

[7] Savino, M. et al. (2023). A process mining approach for clinical guidelines compliance: real-world application in rectal cancer. *Frontiers in Oncology*, *13*, 1090076.

[8] Gatta, R., et al.: pminer: An innovative r library for performing process mining in medicine. In: Artificial Intelligence in Medicine: 16th Conference on Artificial Intelligence in Medicine, AIME 2017, Vienna, Austria, June 21-24, 2017, Proceedings 16. pp. 351–355. Springer (2017)

[9] Glynne-Jones, R., et al.: Rectal cancer: Esmo clinical practice guidelines for diagnosis, treatment and follow-up. Ann Oncol 28(suppl 4), iv22–iv40 (Jul 2017)