Fall 2023

PyGrapherConnect

Shubham Jain

PyGrapherConnect

A Project Report

Presented to

The Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements of the Class

CS 298

By

Shubham Jain

Dec 2023

The Designated Project Committee Approves the Project Titled

PyGrapherConnect

by

Shubham Jain

Approved for the Department of Computer Science

San Jose State University

Dec 2023

Dr. Leonard Wesley	Department of Computer Science
Aneesh Verma	Software Engineer at LinkedIn
Dr. William B. Andreopoulos	Department of Computer Science

ABSTRACT

The evolving landscape of backend computational systems especially in biomedical research involving heavy data operations which have a gap of not being used properly. It is due to the lack of communication standard between the frontend and backend. This gap presents a problem to researchers who need to use the frontend for visualizing and manipulating their data but also want to do complex analysis. CAPRI a python-based backend system specializing in analyzing Evidential Reasoning data also has the same issue. This project offers a solution PyGrapherConnect module acting as a data conversion layer between CAPRI and PyGrapher, its frontend interface. It translates graph data generated by PyGrapher into a txt format readable by CAPRI for further analytical processing. It makes it easier for researchers working with Evidential Reasoning (ER) models, based on the Dempster-Shafer theory to perform the final belief assessment. PyGrapherConnect acting as bridge between frontend and backend, provides a solution to researchers for representing and manipulating these ER models as graph structures, helping researchers to make intricated analytical deductions.

ACKNOWLEDGMENT

I extend my gratitude to Professor Wesley for consistently guiding and being part of this project. Rohin, Divayaraj for their contribution to this project, and finally Department of Computer Science for their resources and support for the opportunities that came up with this project.

I would also like to thank the committee member, Dr. William B. Andreopoulos, and Dr. Fabio Di Troia, Aneesh for their guidance.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Modern computation tools have rapidly become powerful but also have also become increasingly complex. To fully utilize their capability, the integration of the backend system of these computation tools with a simple user-friendly frontend interface has become an inherent requirement by the researchers to work with the tool. This kind of integration is especially important in domains with huge volumes of data from different source relying on mathematical models for decision making, including fields like biomedical research, financial or weather forecasting, and network analysis. The backend of these systems performs heavy computational processes like running parallel simulations for analyzing the data and making inferences from the data. These complex systems become much more usable and accessible to researchers, when combined with the research specific designed frontend interfaces helping them make the complex data in the format needed for the system they are using in their research.

CAPRI an inhouse developed advanced analytical computational system is the best example of the above use case. It is designed for researchers working in the bio medical field specially making decisions based on Evidential Reasoning models. To arrive on a decision based on Evidential Reasoning models where data is sourced from various sources we need a system like CAPRI, specifically designed to analyze these complex models and make inferences based on it. Initially researchers visualize the data in graphical form they aggregated from various sources to reduce complexity and for better understanding of the data on a frontend interface like a website or desktop app. Furthermore, to make decisions based on the graphical data created from frontend they need to feed the data to a powerful analytical system like CAPRI in a specific format. Thus, there is an integration gap between CAPRI and the frontend user interface.

This project solves this conversion problem by developing PyGrapherConnect a conversion module that translate complicated graphical data to a simple format. The software converts the complex graph data containing various nodes each with many numbers of attributes generated by the PyGrapher, the frontend user interface tool to a .txt file format that would meet the predefined schema acceptable by the CAPRI system.

The development of PyGrapherConnect creates a significant enhancement in academic work done by researchers using CAPRI system for analytical purpose based on their graph data. It gives researchers full freedom to freely explore and visualize their ER models without worrying on its further usage. As our module will process their models seamlessly with high accuracy to enable them to make decision after analyzing their graphical data in CAPRI.

Further in the report, Section 2 explains the background of the field and explain the evolution of backend systems with respect to frontend system. This will help in providing some context and understand the need to develop the PyGrapherConnect. Section 3 of this report will give the literature review explaining us work done by others to solve similar problem. It will include an example to help reader better understand the need of our project work. Section 4 of this report will give the technical aspect of the PyGrapherConnect including its detailed System Design. Section 5 give us a use case of our project how our module acting as a bridge between PyGrapher and CAPRI effectively help researchers assess whether a person has the risk of pancreatic cancer or not. Section 6 of the report explains the results and findings of our project work, giving us the conclusive proof on the working of the PyGrapherConnect. Further section 7 of the report, tell us the shortcoming of this project and lays out ideas for future work that will make our system the go to use tool for graph visualization. Lastly, section 8 of the report provides the conclusion of the objective and findings of our work.

II. BACKGROUND

In this era of highly complex yet powerful computational backend systems, there is a significant need to make them more accessible especially in the field of data analysis and modelling. If the data is given in correct format, they have the computation power to process and analyze them. But these systems often lack simpler ways for researchers to generate the data. Moreover, they also lack mechanisms to convert the data needed by them in the required format.

In past, role of backend system was solely focused on performing computations to solve the task given in hand. But as the complexity of the data and its analysis grew particularly in fields such as artificial intelligence, business analytics it became an essential requirement to integrate frontend and backend. Thus, backend system had to evolve to not being just limited to processing data rather working well with outside tools for making the work of their users easier.

Earlier researchers used backend systems involving large volume of data mostly used to run SQL queries to fetch the data and used to manually make inferences based on their custom queries. But as new fields like decision support systems [1] started using this backend system it evolved. Now researchers can not only query the data but rather just feed the data and these powerful backend systems will automatically make inferences from it.

The role of frontend systems has been very instrumental in making these backend systems accessible to non-technical users. Frontend's user-friendly interfaces enable them to engage with their data visually for basic understanding and later for conclusion they can perform operations on backend system. Only limitation will be the language used by the frontend and backend for working on that data is different. Thus in between there is a need for conversion modules that can act as a translator between these two systems.

This data conversion module acting as an intermediary enables a seamless user experience for researchers working on this complex backend system. It takes away the pain point of arriving at a bad conclusion based on the corrupted data as these modules while conversion also verify the integrity of data matches as per the schema of the backend system. If it doesn't match it can simply notify the researcher about the corrupted data. Additionally handling large volumes of data with high accuracy and no significant delay in conversion makes them an essential part of the whole combined system.

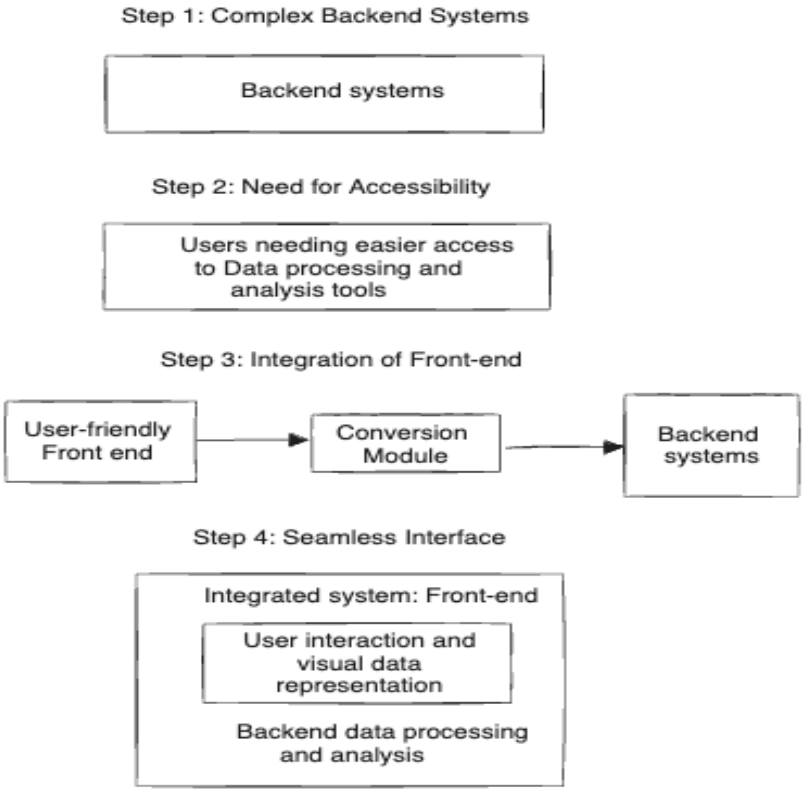


Figure 1: Depicts Evolution of Backend Systems

An example to better explain the evolution of Backend Systems is Customer Relationship Management systems. Earlier when they were introduced, they were primarily backend-focused storing customer information in simple databases. As their importance in business started to grow, they began integrating several other features like sales, analytical services, integrating various backend services into one system. This made it very complex and inaccessible for users with little

experience of using them. This led to the development of simpler user-friendly interface allowing clients to visualize their complex data and quickly make decisions. A conversion module was also developed similarly used to make their system communicate with each other. Modern CRMs like Salesforce [2] allow business to quickly analyze their customer's data, sales and devise new strategies efficiently. Due to integration of frontend and backend CRM systems became a vital part of businesses leading to wider adoption in the industry. This evolution effectively demonstrates the value of conversion module for integrating complex backend systems with a simple easy user interface.

III.LITERATURE REVIEW

This literature review section will explore various approaches made for the integration of backend systems with front end. It gathers its knowledge from different sources like academia, case studies, and industry standard practices.

Card et al. [3] discusses the need for user interfaces in their work that human-computer interaction changes significantly after the introduction of graphical user interface in software design. While GUI made it easier to visualize the data for the end user it was difficult for developer to fully utilize the capability of the backend system. The main issue developers encountered was backend and frontend perform different operations thus had different design language and needed a way to communicate with each other not only efficient but accurately also.

One-way developers solved this problem is by creating data translation layer also called conversion modules that can translated the data generated by one system into a language understandable by the other system. For example, Vasileios et al. [4] in their work lay out the importance of conversion module in biological research, where integration is a necessity for the data aggregated from various sources to perform comprehensive analysis and reach a conclusion. The paper highlights how developing a converter module translating data generated from one source into another format acceptable by other system solves the integration problem. We also need to be varied of translation error that may occur due to corrupt data. Croset et al. [5] paper also points out the same problem, stating data integration may fail for biomedical research project where data comes from many sources and needs to be combined in unified representation for better understanding of diseases and discovery of new treatments for patients.

Shiyong Xiong et al. [6] in their work address the challenges of integrating backend systems with front-end interfaces in data-drive environments like complex data requests, data display differences in the current data transmission processes. Their work involving General Data Layer Components Model provides in-depth understanding in designing an intermediatory layer acting as bridge between different systems. The model developed with Vue components in this paper offers a solution to improve communication and maintain data integrity among various complex systems. Their work provides an understanding for the need of efficient and reliable middleware module for facilitating data flow between frontend and backend.

CAPRI [7], a powerful backend analytical system built on python implements the Shafer-Dempster [8] theory of belief functions and the John Lowrance et al [9] implementation of evidential reasoning. The Shafer-Dempster theory of belief functions through its mathematical theory offers an alternative to traditional probabilistic theory for the mathematical representation of uncertainty by allowing allocation of a probability mass to sets of intervals. Evidential Reasoning system are a form of normalized probabilistic reasoning based on the representation of probabilistic ignorance by intervals of possible values and its framework is provided by the implementation of Evidential Reasoning by John Lowrance et al. Thus, the base foundation built upon mathematical and theoretical frameworks of CAPRI is solid, making it a powerful handy tool for researchers working on Evidential Reasoning.

However, for full utilization of its capabilities it requires data to be in a specific format with specific schema. The challenge of integrating CAPRI system with a fronted tool, that too in a particular format dependent on the value of node attributes, is very similar to issues faced in the field of data integration. Apache NiFi [10] and Talend [11] addressed a similar problem of combining data from various sources signifying the importance of well-

defined standards for interoperability of data between various systems. Therefore, the requirement of data conversion layer especially in biomedical domain is very essential, as signified by Vasileios et al. [4]

To better understand, suppose a researcher with various Evidential Reasoning models wants to analyze and make decision on it. CAPRI system based on the graph type needs each node data to be in a specific format as represented in Figure 2.

a. The first line of the .txt file will be
   # FILENAME: <the name of this .txt file>

b. The second line of the .txt file will be
   #FILETYPE: analysis

c. The third line of the .txt file will be
   # <- followed by a blank/empty line.

d. For each node in the JSON file construct the following entry.

   If the value of the ANALYSISNODETYPE: attribute/slot for the node is "Input" then generate the following lines

   STARTANALYSISNODE:
   ANALYSISNODENAME: <Value of the ANALYSISNODENAME attribute/slot for the node>
   ANALYSISNODETYPE: Input
   FRAMENAME: <Value of the name of the node>
   FROMANALYSISNODES: <Value of the ANALYSISNODENAME attribute/slot for the node>
   TOANALYSISNODES: <Value of the ANALYSISNODENAME attribute/slot for the node>
   ENDANALYSISNODE:

   Repeat the above steps for every Input type node in the JSON file.

Figure 2: Depicts the strict requirement for node data in CAPRI.

CAPRI system only accepts txt file input. A researcher with no access to frontend tool will have to use his own imagination to make the relationship between these nodes and create the .txt file. This is a very risky and complex approach as without visualizing their models they will not have a better overall understanding of how to utilize the aggregated data specific to each Evidential Reasoning model.

To eliminate this problem faced by the researchers we need a frontend tool to generate the graph data even though the main analysis for arriving at a decision is performed in CAPRI. PyGrapher, a graph visualization software is frontend system used by researchers to visualize their Evidential Reasoning models and generate the graph data. But like cases discussed in above sections both PyGrapher and CAPRI speaks different language. PyGrapher generated the graph data in JSON file while CAPRI only accepts .txt file that too with specific schema for each node as depicted in Figure 2. Therefore, PyGrapherConnect development was critical to address this communication problem. It acts as a translator between CAPRI and PyGrapher.

PyGrapherConnect ability to generate txt file format based on the node properties, like Gallery and Analysis types, is very similar to data transformation tools like Apache Camel [12] and Informatica PowerCenter [13]. The module's role in converting JSON files from PyGrapher into .txt files for CAPRI system behaves like the ETL (Extract, Transform, Load) processes, used in data warehousing techniques.

The wider implication of building PyGrapherConnect lies in its potential to simplify graph management for researchers working in ER models, similar to progress observed with ArangoDB [14] and Amazon Neptune [15], another graph-based system.

Specifically related to area of ER models, linking the output of PyGrapher and CAPRI through PyGrapherConnect is very similar to growing trend in data science and analytics where data from various sources is combined to be represented in standard format for a

more comprehensive analysis. Two of the widely used data integration platform are IBM DataStage [16] and Microsoft SQL Server Integration Services (SSIS) [17] work similar to our tool.

The next section will provide the system design of PyGrapherConnect and explain its magic of linking PyGrapher and the CAPRI system. It will explain the technical aspect of our project in detail, giving reader a broader understanding on the significance of it for seamless integration between PyGrapher and CAPRI.

IV.SYSTEM DESIGN OF PYGRAPHERCONNECT

PyGrapherConnect is the conversion module we build as part of this project which convert graph data from PyGrapher into a format compatible with CAPRI's analytical tools. PyGrapherConnect is completely built using python and its architecture consist of several key components each playing a vital role to make the module as a whole function correctly.

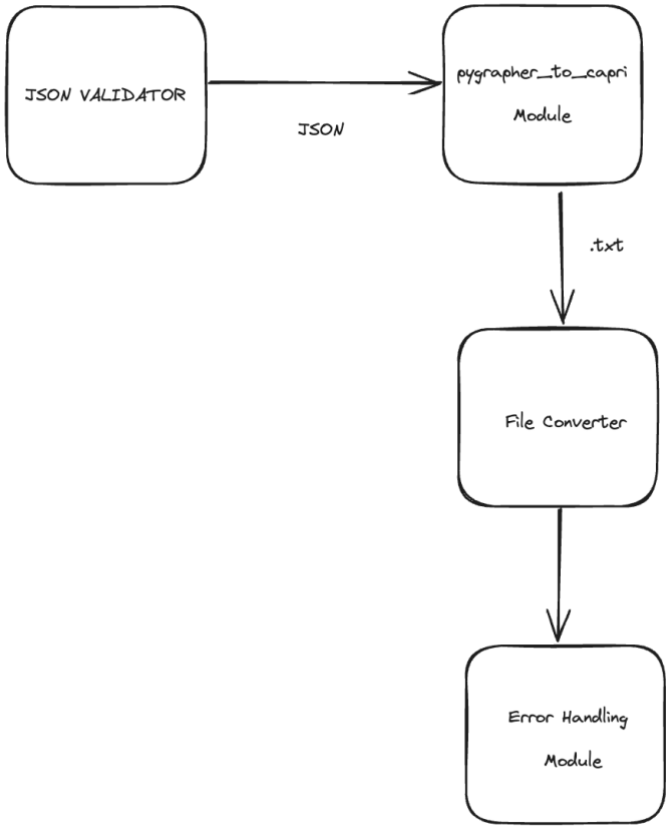


Figure 3: PyGrapherConnect Architecture

Figure 3 in the report gives the basic representation of the PyGrapherConnect system's architecture. The process starts with the **JSON Validator** module, which is responsible for verifying the schema of the JSON file received from PyGrapher. This check is

necessary to ensure that the graph data we are converting meets the required standards as per the CAPRI system and formats before any conversion process begins.

Next is the **pygrapher_to_capri** Module. This module acts like the central processing unit of the PyGrapherConnect, converting JSON files exported by PyGrapher Frontend into the .txt format needed by the CAPRI system. Currently it supports two graph types– Gallery and Analysis – and generates the text file based on the **graphType** property in the JSON file. This module is intentionally designed such a way that it functions are very modular allowing it to support more graph types in future.

After the conversion process, the **File Converter** module takes over. It performs any necessary file format adjustments to ensure that the output is in the precise .txt format required by CAPRI system. In brief its role is to ensure that the converted data is in the correct file extension format accepted by CAPRI system. Similar to pygrapher_to_capri its design is also modular, thus in future it can support more.

Finally, the **Error Handling Module** is utilized for making sure the conversion process is reliable end to end for the user thus making the whole system reliable and robust. During the conversion process if any error occurs this module takes care of it keeping the end user experience smooth. Based on the type of exception that occurs the module displays the error message to user thus enhancing the stability of the whole PyGrapherConnect operations.

In summary, PyGrapherConnect architecture is designed to seamlessly bridge the PyGrapher and Capri systems, facilitating efficient data conversion and processing.

4.1 JSON Validator

The validation of JSON data for the conversion is a necessary step especially when dealing with complex graph data, such as that handled by PyGrapher. This step is very crucial given the generated data serves as a foundation for ER models, where precision and accuracy in graph representation are paramount. The JSON Validator module within PyGrapherConnect role is to scrutinize the validity of the JSON file given to the PyGrapherConnect.

In simple terms, the JSON Validator is a Gate Keeper deciding whether to allow a file to enter the PyGrapherConnect module or not. The PyGrapher frontend generates heavy graphical data with complex and nested node properties, the JSON Validator make sure that the content file being passed to PyGrapherConnect meets the JSON schema. This validation process is critical, as without the check incorrect txt file might be generated by our backend thus leading to incorrect data analysis in the Capri system.

Specifications

Currently our backend system supports two text files format supported by the CAPRI system.

a. Gallery .txt file

b. Analysis .txt file

Determining the Text File Type

A key field within the PyGrapher JSON file, called " graphType" determines the type of .txt file that will be generated. If the field value is:

“Gallery”: It results in the creation of a gallery .txt file.

“Analysis”: It results in the creation of an analysis .txt file.

If the "Graph Type" field contain any value aside from "Gallery" or "Analysis," PyGrapherConnect will stop the conversion process and alert the user with an error message courtesy to the error handling module, explained thoroughly later in the upcoming sections.

Structure of Gallery Type .txt File

The gallery types .txt file has a predefined structure as follows:

a. # FILENAME: The value of this property is the name of our generated file.

b. #FILETYPE: The value of this property indicates the file type as gallery.

c. #: A comment sign succeeded by a blank line.

Figure 4 gives the structure of a single Node.

```
STARTFRAME:
FRAMENAME: [Node Name]
TRUEPROP: [TRUEPROP Value]
FROMFRAME: [FROMFRAME Value]
TOFRAME: [TOFRAME Value]
COMPATREL: [COMPATREL Value]
ENDFRAME:
```

Figure 4: Node Structure for Gallery Type.txt file

Structure of Analysis Type .txt File

The analysis types .txt is formatted as follows:

a. # FILENAME: The value of this property is the name of our generated file.

b. #FILETYPE: The value of this property is the name of our generated file.

c.#: Followed by a blank line.

Figure 5 gives the structure of a single Node.

```
STARTANALYSISNODE:
ANALYSISNODENAME: [ANALYSISNODENAME Value]
ANALYSISNODETYPE: [Node Type]
FRAMENAME: [Node Name]
FROMANALYSISNODES: [FROMANALYSISNODES Value]
TOANALYSISNODES: [TOANALYSISNODES Value]
ENDANALYSISNODE:
```

Figure 5: Node Structure for Analysis Type.txt file

This node structure also changes according to various node types such as "Input," "Discount," "Translate," "Fuse," and "Interpret," ensuring each node's unique properties are correctly represented in the .txt file.

The detailed specifications given here ensure that the pygrapher_to_capri.py module effectively transforms PyGrapher's JSON outputs into well-structured .txt files, ready for analysis by the CAPRI system. This transformation enables the researchers to feed their graph data to the CAPRI system allowing them to interpret and process the graph data with the precision and accuracy required for advanced modeling and analytical tasks.

The JSON Validator can validate large number of JSON files within very short time. Its built considering JSON Schema [18] and using Python, with its widely used built in library [19], The major choice of choosing these technologies is the use and wider community support available.

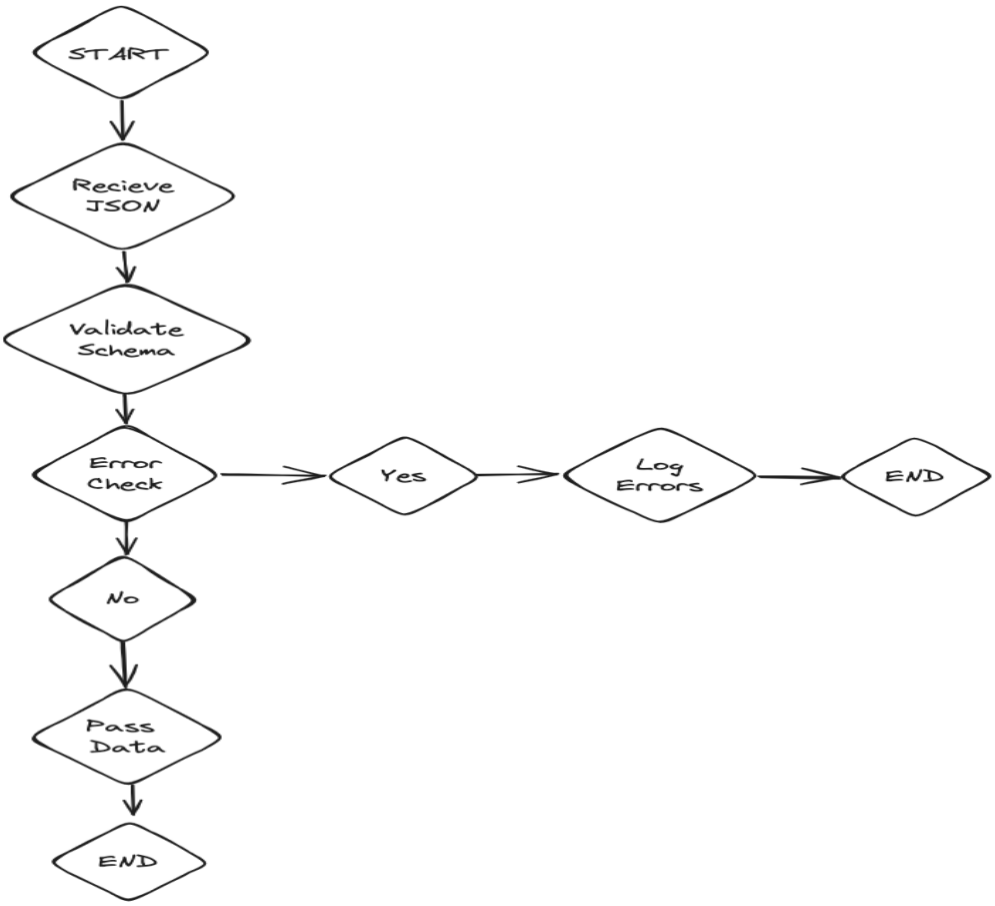


Figure 6: JSON Validator Flowchart

In case of JSON files passing the JSON Validator validation the module then passes the file to the PyGrapher_to_capri module thus fulfilling its role of data quality inspector ensuring data quality.

In the event of validation failure, the Validator with the help of Error handling module generates detailed logs with stack trace enabling precise debugging of the issue. This ensures the necessary diagnostic can be performed immediately to rectify any error immediately and prevent its future occurrence.

4.2 PyGrapher_to_capri Module

As mentioned in previous sections, the pygrapher_to_capri.py module is the most critical subsystem of our PyGrapherConnect designed to interface directly with the CAPRI system. Its primary function is to convert the JSON output generated by PyGrapher, containing complex graph data with different node attributes, into a .txt file format that can be readily ingested by the CAPRI system for further analysis.

It makes our whole PyGrapherConnect system a simple python script that needs to be run by the user along with the JSON file. The user can run the script from the terminal or in a python integrated development environment. The script will produce the output as per the naming convention defined in the schema and based on the graph file type provided. For convenience of the users the file is generated and automatically saved in the same directory the script is run.

System Requirements and Dependencies

For the pygrapher_to_capri.py module to perform optimally, several system prerequisites and dependencies must be met. Since the whole PyGrapherConnect system is built on Python, this module takes advantage of its powerful data handling libraries. Python's versatility and the rich support of its ecosystem make it an ideal choice for this module's development. The system requirements for running the module include:

- Python interpreter (version 3.x recommended for its improved features and support)
- Access to the standard Python libraries json and os, which handle JSON file manipulation and operating system interactions, respectively.

For successfully running the script all dependencies must be met. The Json file input to the module must aligns specific to the JSON structures and schemas as specified in previous sections, which PyGrapher outputs.

Module Architecture Overview

The pygrapher_to_capri.py module comprises of several components that aids in this data transformation process. The main three components of the architecture are:

- Input Processor
- Converter
- Output Generator

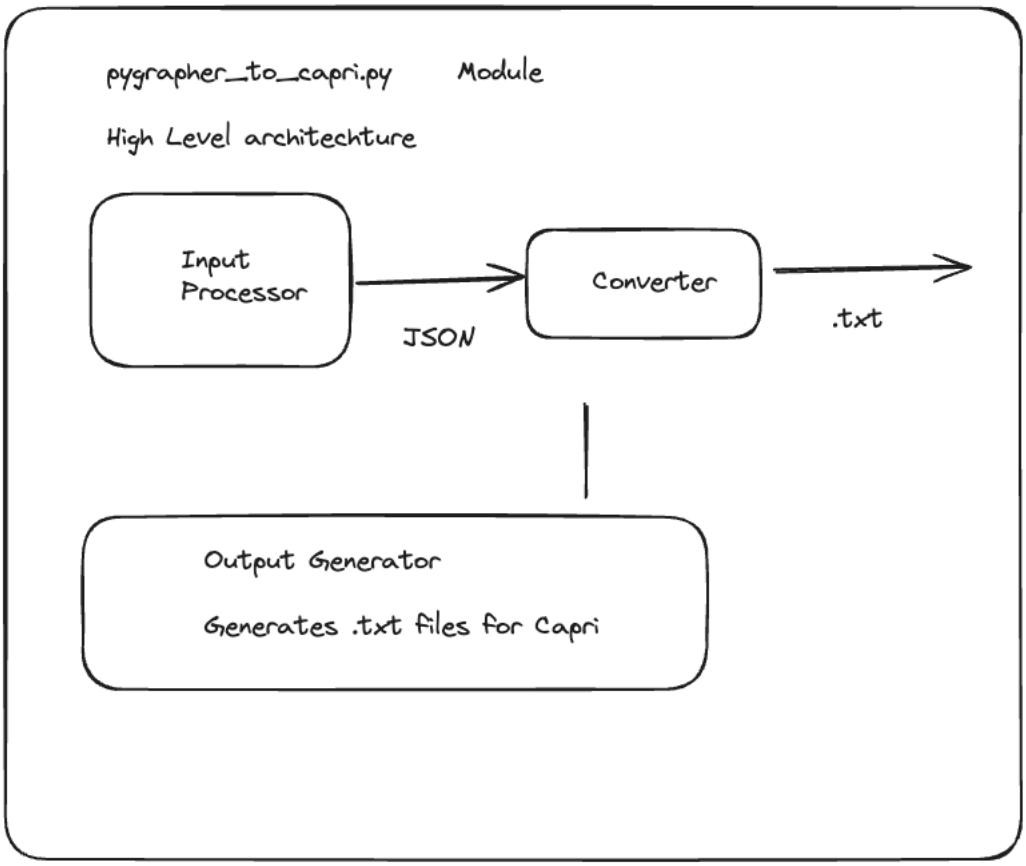


Figure 7: High-Level Architecture of pygrapher_to_capri.py Module

Input Processor

The Input Processor act as gatekeeper similarly to JSON validator but the major difference is that input processor also validates the contents of the JSON files has the necessary nodes present for file conversion to be successful. In brief, it makes sure that the JSON file has required structure and types as specified in the PyGrapher specifications section.

The validation logic within the Input Processor draws from established practices in data verification. The steps consist of:

- With the built-in Json library, the graph data inside the JSON file is parsed and converted into a native Python dictionary.
- Checking the value of the Graph Type field for determining which conversion pipeline to proceed i.e. — whether to produce a gallery or an analysis type .txt file.
- Validate the data parsed contains all necessary key attributes required for successful conversion.

Converter

The Converter is responsible for the actual transformation of data present in the JSON format to make it ready for our output .txt files. It performs various steps as outlined below that map the JSON structure It executes a series of steps that map the JSON structure present in the graphical data generated by our PyGrapher frontend to simple lines of text that matches with schema as defined in CAPRI's input requirements.

Following are the step involved in the conversion process:

- Since the JSON structure contains several nodes, we iterate over each node.
- For every node, we extract the required info from the attributes and store them as strings, string names matches the .txt file structure defined in CAPRI system's schema.
- Finally, the metadata such as filename and filetype are added as headers in the .txt file, this part is modular that has placeholders allowing for future additional information to be added if required by the CAPRI system.

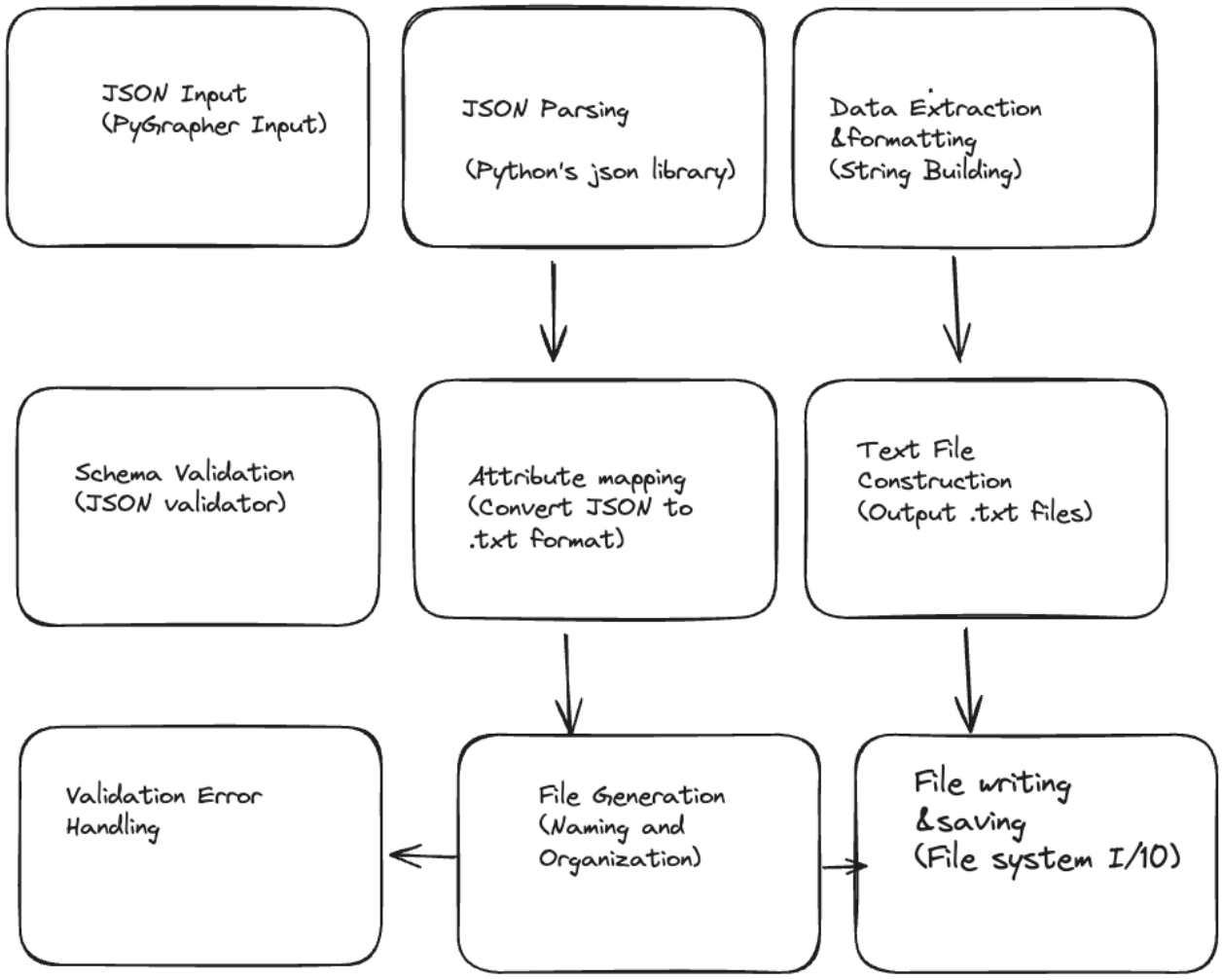


Figure 8: Workflow of the PyGrapherConnect system with the Converter module

Output Generator

After the conversion happens, the Output Generator activates to give the finishing touch to our converted .txt file. It names the file depending upon the value of the Project Name field and adds the suffix ("_gallery.txt" or "_analysis.txt") as per the Graph Type. The naming convention helps researchers to identify and differentiate between the various .txt files generated for the CAPRI system [20].

The Output Generator also make sure the generated file is saved in the system directory. Each line of the text is correctly encoded and formatted, following the guidelines for .txt file creation [21]. The output .txt files are saved in a same directory from where our PyGrapherConnect ran.

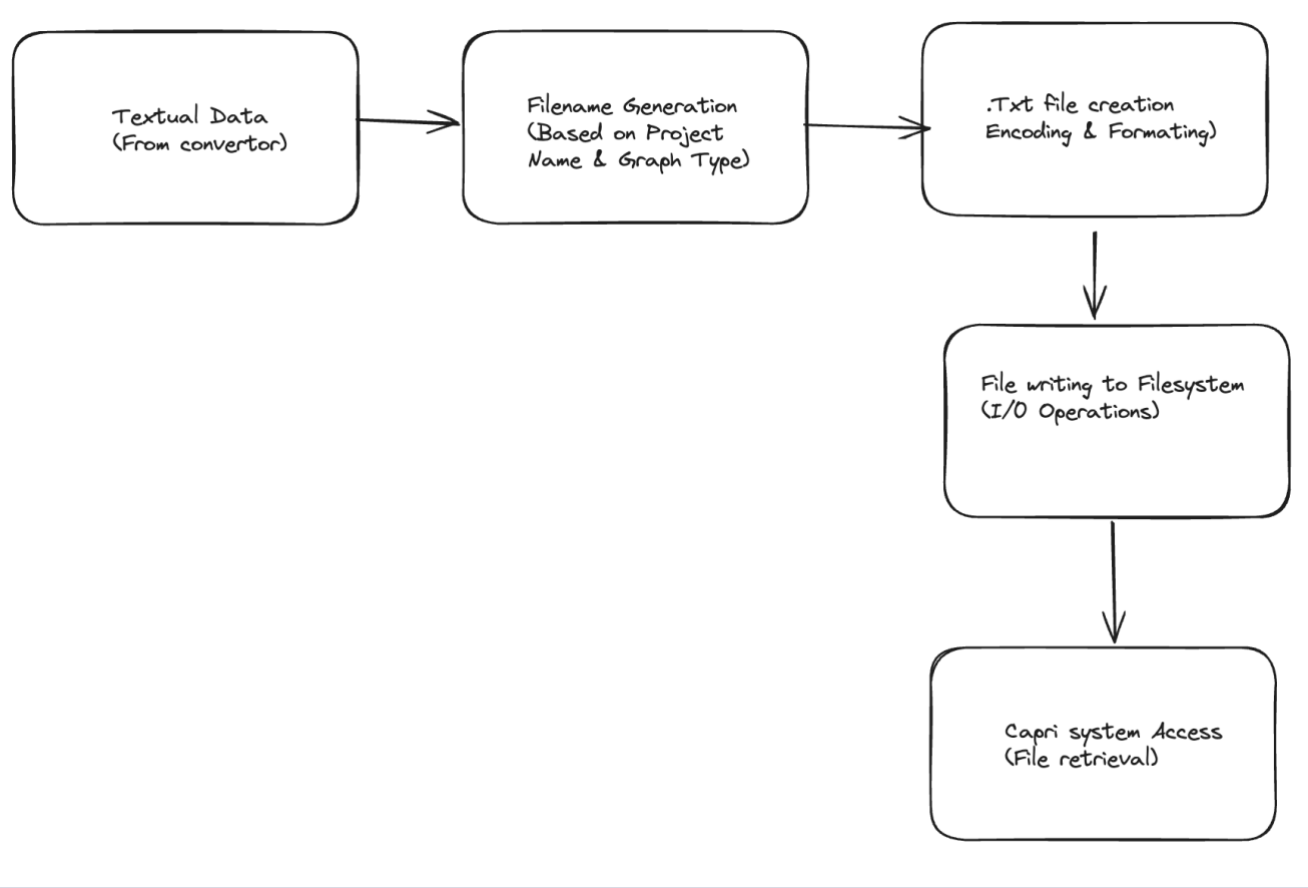


Figure 9: Workflow of the PyGrapherConnect system with the help of Output Generator

Debugging

For debugging the pygrapher_to_capri.py module we have error handling and logging part. During an exception in Input Processor, such as an invalid graph type or missing data, the error routine starts:

This routine includes:

- Error Identification: First the name of the issue is identified based on the error type like data mismatch, missing node property, invalid graph type.

- Logging: For every error that occurs its stack trace is logged. This is an essential for full stack debugging and makes our project as whole adheres to the best practices in software development.
- Notification: For better user experience instead of crashing the system when error occurs it notifies user with an error message
- Graceful Exit: When an exception occurs, after displaying the error message the module ends the program gracefully instead of abruptly shutting it down.

Integration with the Capri System

The output of the pygrapher_to_capri.py module is a text file that can be used in the CAPRI system for further analytical processing. One notable thing about our system is its seamless integration with the CAPRI as the generated txt file do not require further preprocessing. This seamless integration is direct result of collaborative efforts of the frontend and backend team as well as our advisor for providing the schema making sure the generated file meets the schema required for valid file for the CAPRI system.

This seamless integration is made possible due to:

- File Directory Structure
- File Format Consistency
- Automated Ingestion

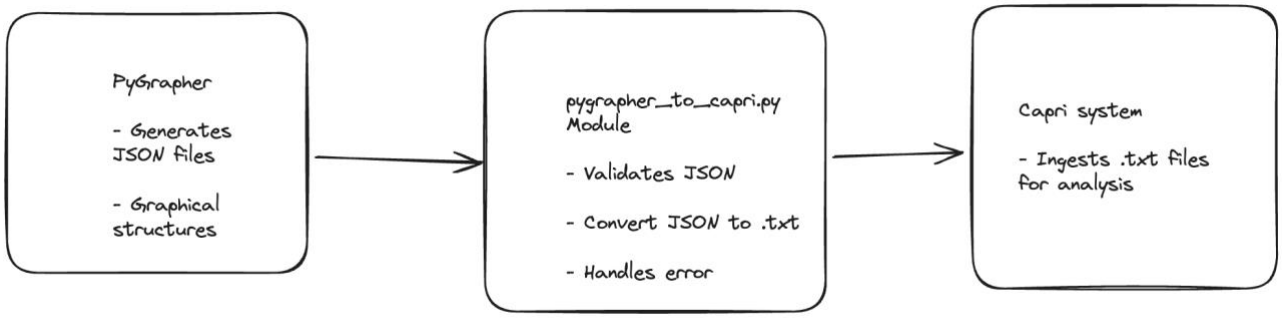


Figure 10: Depicts the integration of our module with the CAPRI system.

Testing and Validation

The **pygrapher_to_capri.py** performs extensive testing on our system making sure to catch any bug before it goes into wider release.

Key aspects of testing involved:

- **Unit Testing**: Each function of the module was tested extensively to verify our module functionality. For E2E testing the system was fed with mocked JSON files including edge cases.
- **Automated Regression Testing**: A suite of regression tests was developed to automatically run after each update to the module, ensuring that new changes did not introduce errors or disrupt existing functionality.

User Guide and Documentation

A detail readme is there with the pygrapher_to_capri.py module, providing user with clear installation instruction including solution to common installation error. It includes guide to installation process, system requirements, error handling.

4.3 File Converter

The file converter is an essential component for the data transformation process that is designed to run in line with the overall architecture of the PyGrapherConnect. It is mainly responsible for the converting the data generated by pygrapher_to_capri module to.txt file format acceptable by the CAPRI system. This is the conclusive step to save the integrity and the relevancy of the underlying graph data made by the researchers using PyGrapher frontend from the point of its creation to the next phase of analysis in the CAPRI.

The File Converter also integrates the features of the Error Handling and logging module. This integration is crucial for preemptively identifying and mitigating potential discrepancies during the conversion process. By ensuring that data errors are captured and addressed, the module aligns with best practices in software development for data integrity.

4.4 Error Handling and Logging

The error handling and component makes our whole system reliable and is the only subsystem that integrated with all the different modules that collectively makes out PyGrapherConnect system.

As soon as an error is found, the module records the incident in a quick and accurate manner. In the system, logging is done proactively that takes into account the complete picture of the state of error rather than passive recording of the events. This proactive measure includes steps such as capturing detailed information's like timestamp of error occurrence, thorough description of the

error situation, particular stage, and context of the workflow where the error occurred. These detailed logs of information are very helpful in diagnosing the error and prevent its future occurrence and makes the whole PyGrapherConnect system robust and reliable.

Furthermore, the scope of customization of this module is very high which enables it to cater to various error scenarios, ranging from simple JSON syntax error to a complicated data mapping error. This capability is of great importance as it allows it to handle any situation irrespective of its complexity and ensures accurate validation and efficient communication to user on a consistent basis.

In summary, the Error Handling and Logging design component of PyGrapherConnect is of fundamental importance to the system that helps to ensure data integrity, quick error resolution, reliable user experience. It also aids developers in analyzing complex graph data at the time of transforming data for the CAPRI system.

V. RESULTS

This report section explains the outcome of the data transformation process performed by PyGrapherConnect. The data is processed by the pygrapher_to_capri.py module inside the PyGrapherConnect system and output file is discussed here. The process is illustrated using a sample JSON input file fed into the PyGrapher and further explains the .txt result file which is generated in a format that is suited to the CAPRI system.

5.1 Sample Gallery Json File

Below is a sample JSON input representing a graph structure for a hypothetical project called "Sample_Gallery". This JSON file includes a field "graphType" with value "Gallery", which decides the type of .txt file to be generated by the module. Notice the custom properties and attributes present in our JSON file, this highlights the customization options available to users using our PyGrapher frontend tool.

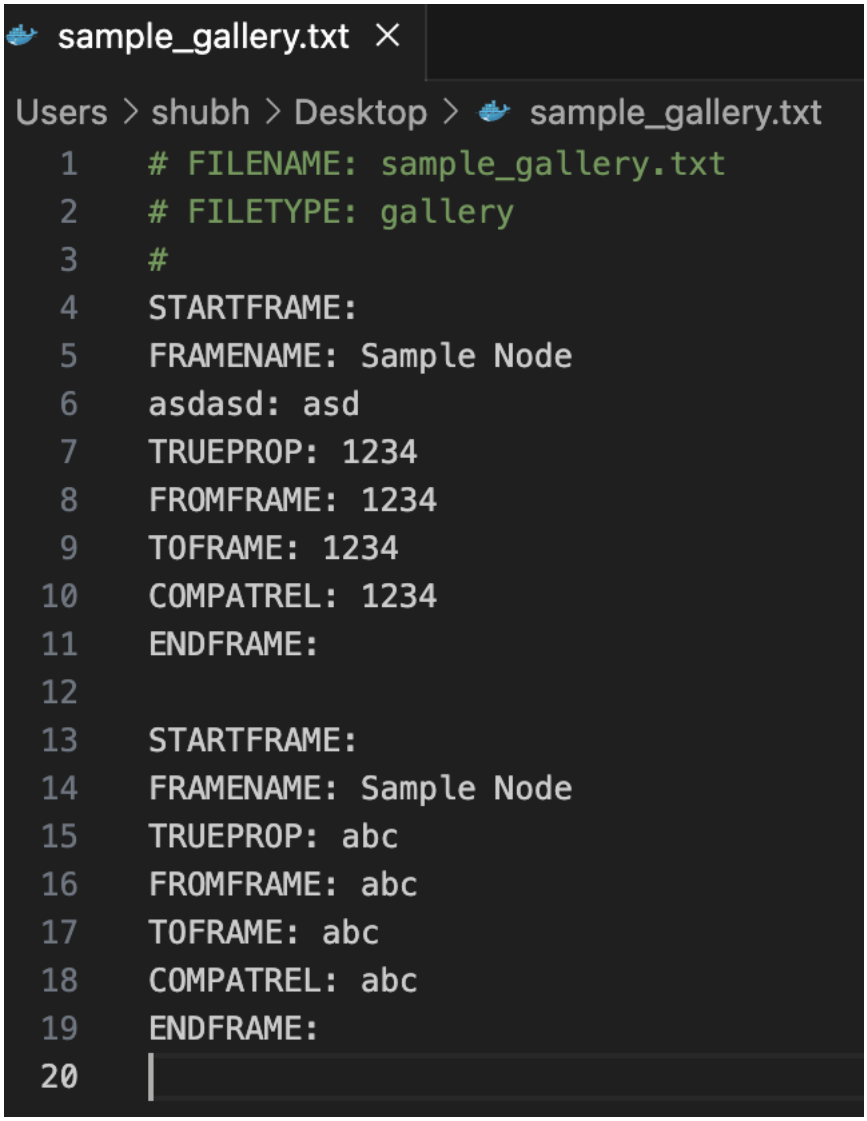
```
{
    "graphType": "GALLERY",
    "graphName": "sample",
    "graphProjectName": "Sample_Gallery",
    "nodes": [
        {
            "id": "db3ae7f0-29dd-4954-9fde-4c2a39d6461a",
            "size": 150,
            "label": "Sample Node\nasdasd: asd",
            "properties": [
                {
                    "key": "COMPATREL",
                    "value": "asd"

                },
                {
                    "key": "TRUEPROP",
                    "value": "1234"
                },
                {
                    "key": "FROMFRAME",
                    "value": "1234"
                },
                {
                    "key": "TOFRAME",
                    "value": "1234"
                },
                {
                    "key": "COMPATREL",
                    "value": "1234"
                }

            ],
            "color": "#7ed321",
            "shape": "triangle",
            "font": {
                "face": "monospace",
                "align": "left"
            }
        }
    ]
}
```


Generated .txt File Output:

On the basis of the sample JSON input, the pygrapher_to_capri.py module will generate a .txt file called as "sample_gallery.txt" as shown below:



The generated file is correct and need no further preprocessing before feeding it to the CAPRI system. It is exactly as per the predefined schema of the CAPRI system and can be used instantly for further analytical processing.

5.2 Sample Analysis Json File

Attached below is a sample JSON input presenting a graph structure for a hypothetical project called "Sample_Analysis". The below JSON file includes a "Graph Type" of "Analysis", which decides the type of .txt file to be generated by the module.

Generated .txt File Output:

On basis of the sample JSON input, the pygrapher_to_capri.py module creates the following .txt file called as "sample_analysis.txt":

```
# FILENAME: sample_analysis.txt
# FILETYPE: analysis
#
STARTANALYSISNODE:
ANALYSISNODENAME: AnalysisNode1
ANALYSISNODETYPE: Input
FRAMENAME: Node 1
FROMANALYSISNODES: prev_node
TOANALYSISNODES: next_node
ENDANALYSISNODE:
```

As observed in the sample output illustrated above, the conversion of JSON file to .txt format is completed at high level of accuracy. The graph's properties and structure are reflected correctly in the .txt file, being in accordance with the input specifications with the translation of node attributes and relationships in line with the JSON input. The result output thus confirms that the pygrapher_to_capri.py module is effective in preserving the integrity and intent of the graph data during the conversion process.

Therefore, the results confirm that pygrapher_to_capri.py module is capable of converting complex JSON graph data to a .txt file which is compatible with the CAPRI system, in an accurate and efficient manner. The sample transformation example shows above thus proved that the process is reliable and helps facilitate complex graph analysis with the CAPRI framework for researchers.

VI. Use Case - Assessing Pancreatic Cancer Risk

This section will demonstrate with real world example how significantly PyGrapherConnect eases the work of researchers using CAPRI and PyGrapher. We will see how seamlessly it integrates the frontend and backend system.

In this example scenario a biomedical researcher is making the decision about the probability of patient having pancreatic cancer or not. The researcher has complete access to patient medical history. Based on the patient's medical data the researcher will process the necessary evidence required for him to reach a conclusion. To keep it simple let's, say the researcher picks the Blood test, Imaging reports, Genetic test Evidence to create Evidential Reasoning models for better understanding. They use PyGrapher to create the graphical view as shown in Figure. 11.

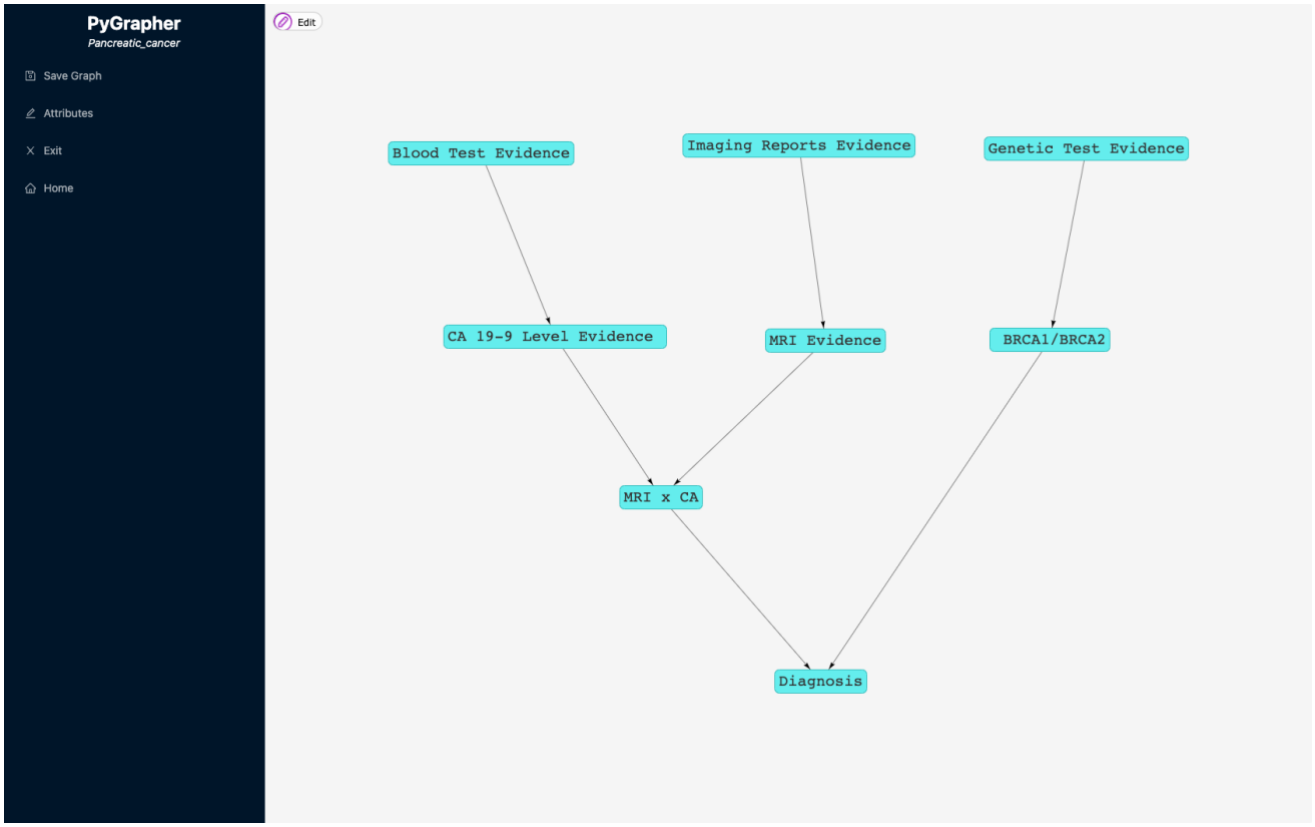


Figure 11: Evidential Reasoning models visualization in PyGrapher

After visualization, to arrive at a decision analytical processing in CAPRI is required. But the graphical data generated in PyGrapher is in JSON format and CAPRI only accepts txt files with its predefined schema. To convert this graph data into a format acceptable by CAPRI PyGrapherConnect is utilized.

PyGrapherConnect will parse through the whole intricate graphical data and take the evidence present in the JSON file. It will store these values in the text file as per the schema and make it into a structured text file acceptable by CAPRI system. After the conversion is completed, researcher feed the generated text file to CAPRI, which performs its Dempster-Shafer combination and do final belief assessment in its analysis as per the graph type and provides the analytical results. Based on the analytical result generated by the CAPRI researchers reliably makes the final decision whether the patient is at risk of pancreatic cancer or not.

Figure 12 depicts the complete journey of the researcher doing pancreatic assessment with the evidential reasoning model approach.

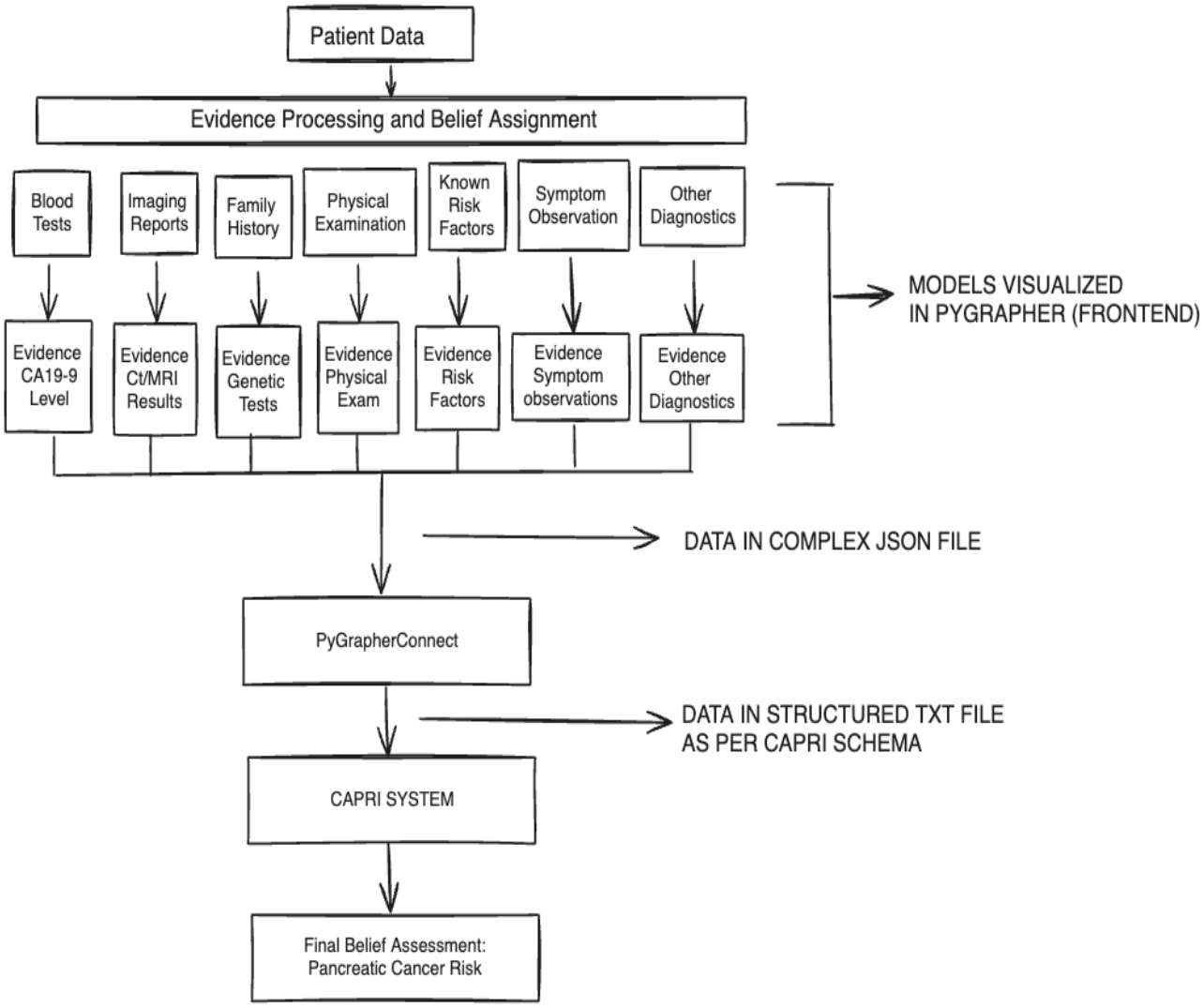


Figure 12: Pancreatic risk assessment using Evidential Reasoning approach.

This use case underscores the requirement of PyGrapherConnect and its importance in helping the researcher perform cancer risk assessment. Acting as reliable bridge between frontend and backend it not only simplifies the whole process making it more accessible but also reliable with its accurate conversion.

VII. FUTURE WORK

.

In the final release of our project work, PyGrapherConnect has been realized to be an effective means to transform complex graphs generated by the PyGrapher in to formats compatible with CAPRI platform. However, there are several additional features which can be added to PyGrapherConnect to make it the best free tool for academia to do their research in Graph visualization. The goal of adding these features in the future work is to further expand the capabilities and utility of the module, to keep up with the fast-paced development in the graph visualization and data analytic field.

7.1 Expansion of Supported Graph Types

The current module only supports “Gallery” and “Analysis” graph types, while in future PyGrapherConnect scope can be expanded to include other graph types such as “Mass distribution” or others that may be identified to be important for researchers in their graph analysis.

7.2 Direct integration with the Frontend System

As shown in Figure 13 our backend system can also directly integrate in the frontend instead of manually running the script separately. From the user interface only, we can provide a button clicking on which will directly covert the on-screen graph models to txt file as per the standards accepted by CAPRI. This will significantly improve user experience of our tool and make performing analysis on CAPRI easier and accessible.

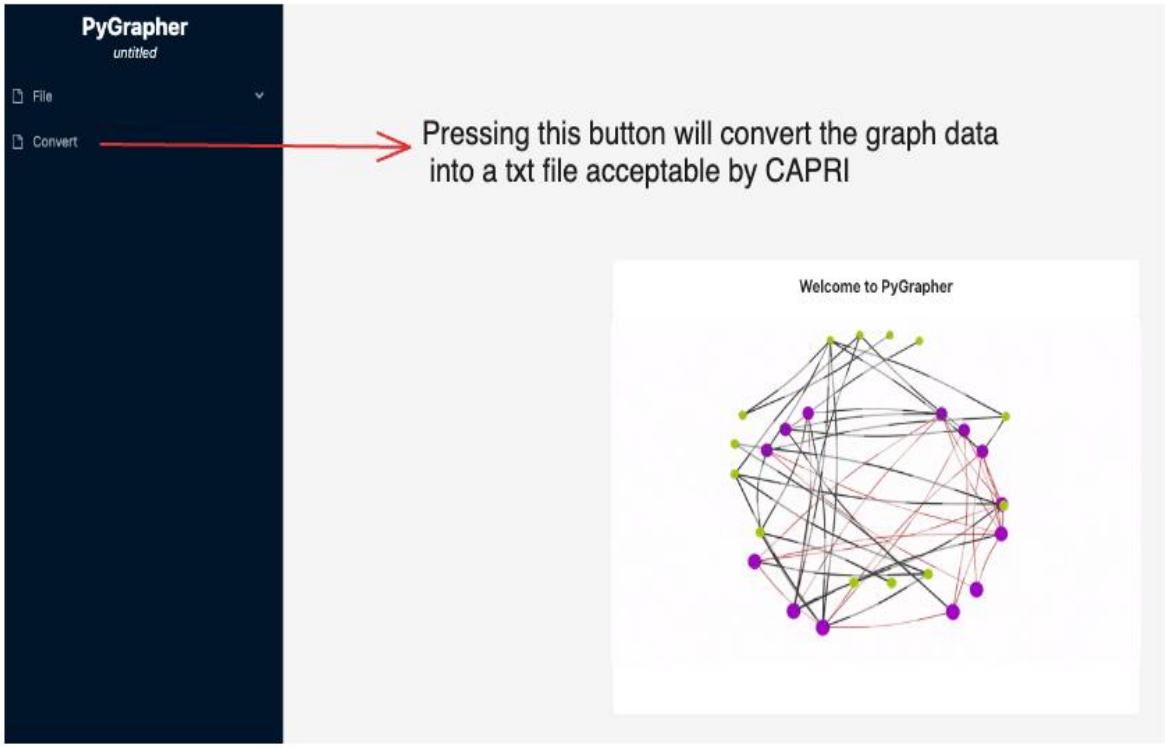


Figure 13: Convert button in PyGrapher.

7.3 Enhanced Error Handling Mechanisms

The current error handling mechanism can be improved further with the introduction of machine learning to include techniques like prediction and mitigation of error before they could occur. This step might improve the data processing time and accuracy by significant degree.

7.4 Advanced Customization Features

More customization options can be provided to users like freedom to set custom conversion rules that might cater to different graph structures.

7.5 Performance Optimization

Efforts can be taken to reduce the conversion time taken in generating text files from the JSON file containing complex graph data and optimizing module to handle larger and more complex data in less time and higher accuracy.

.

VIII. CONCLUSION

The development and integration of PyGrapherConnect system with the PyGrapher frontend in the PyGrapher system reflects a significant level of advancement in making complex powerful backend systems more accessible as the gap between data visualizing capability of PyGrapher and analytical capability of CAPRI system is successfully closed. The importance of PyGrapherConnect is proved as complex graph data generated by the researchers are successfully translated to .txt file for further use in their analytic purpose.

PyGrapherConnect has demonstrated high data integrity, accuracy and reliability through its sophisticated error managing and logging techniques, thereby providing consistent user experience to researchers, and gaining their confidence. It's simple design and functionality also helped in enhancing the user experience.

Going forward, adding the features described in the future work will significantly ease the user experience of researchers with PyGrapherConnect converting behind the scenes faster with high accuracy.

PyGrapherConnect also is a testament to the result achieved by a well-planned collaborative team effort under an able advisor.

REFERENCES

[1] Norman, D. A., & Draper, S. W. (1986). User Centered System Design: New Perspectives on Human-Computer Interaction. Lawrence Erlbaum Associates, Inc.

[2] “Salesforce.” https://www.salesforce.com

[3] Card, S. K., Moran, T. P., & Newell, A. (1983). The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates.

[4] Lapatas, V., Stefanidakis, M., Jimenez, R. C., Via, A., & Schneider, M. V. (2015). Data integration in biological research: an overview. Journal of Biological Research-Thessaloniki, 22(1), 9. https://doi.org/10.1186/s40709-015-0032-5

[5] Croset, S., Rupp, J., Romacker, M. (2016). Flexible data integration and curation using a graph-based approach. Bioinformatics, 32(6), 918–925.

[6] Xiong, S., Li, X., & Wang, Y. (2022). Research on General Data Layer Components Model based on Data-Driven. Journal of Physics: Conference Series, 2303(1), 012025, IOP Publishing. https://doi.org/10.1088/1742-6596/2303/1/012025

[7] lwesley/capri: Causal and approximate probabilistic reasoning and inference. https://github.com/lwesley/capri

[8] Dempster, A. P. (2008). The Dempster–Shafer calculus for statisticians. International Journal of Approximate Reasoning, 48(2), 365-377, Elsevier.

[9] Lowrance, J. D., Garvey, T. D., & Strat, T. M. (1988). A Framework for Evidential-Reasoning Systems. In Yager, R. R., & Liu, L. (Eds.), Classic Works of the Dempster-Shafer Theory of Belief Functions. Studies in Fuzziness and Soft Computing, vol 219. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-44792-4_16

[10] "Apache NiFi: Automated and Secure Data Routing." https://nifi.apache.org/

[11] "Talend: A Leader in Data Integration and Data Integrity." https://www.talend.com/.

[12] "Apache Camel: Integration Patterns Framework." https://camel.apache.org/.

[13] "Informatica PowerCenter: Data Integration for Enterprise." https://www.informatica.com/products/data-integration/powercenter.html.

[14] "ArangoDB: Multi-Model Database for Graph, Document, and Key-Value." https://www.arangodb.com/.

[15] "Amazon Neptune: Fully Managed Graph Database Service." https://aws.amazon.com/neptune/

[16] "IBM DataStage: Data Integration Tool." IBM. https://www.ibm.com/products/datastage

[17] "Microsoft SQL Server Integration Services (SSIS)." https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services.

[18] "JSON Schema: A Media Type for Describing JSON Documents." https://json-schema.org/

[19] "Python JSON: The Ultimate Guide." https://docs.python.org/3/library/json.html.

[20] Harvard Medical School. File naming conventions. Data Management. Retrieved from https://datamanagement.hms.harvard.edu/plan-design/file-naming-conventions.

[21] Wesley, L. PyGrapher to Capri Specification, Version 1. https://github.com/lwesley/pygrapher/blob/master/pygrapher-to-capri-spec/PyGrapher_to_Capri_Spec_v1.pdf.