UNIVERSITÀ DI TRENTO

Department of Information Engineering and Computer Science

Bachelor’s Degree in
COMPUTER SCIENCE

FINAL DISSERTATION

A WEB GRAPHICAL USER INTERFACE FOR THE PACKET-LOSS-CONCEALMENT TESTBENCH TOOL

Supervisor
Prof. Luca Turchet

Co-Supervisor
Ing. Luca Vignati

Student
Stefano Dallona

Academic year 22/23

Acknowledgements

*To **my wife** and to **my daughters***
who constantly supported me during this journey.

*Thanks to **Prof. Luca Turchet** and **Ing. Luca Vignati***
who provided guidance and precious feedback during the entire thesis work.

INDEX

Abstract

The purpose of this thesis was implementing a graphical user interface for the PLC (Packet Loss Concealment) Testbench Tool developed by the Ph.D. student Luca Vignati, thus making the tool easier to use while at the same time empowering the interpretation and analysis of the tool's outputs. PLC algorithms are typically needed in real-time data transmission contexts, and their purpose is to mask ("conceal") any packet loss as transparently as possible to the application consuming the data stream. The PLC Testbench Tool aims at providing a modular framework to support a qualitative and quantitative comparison of different PLC algorithms' performance, by applying them to a set of input files. The qualitative analysis is supported through several visual representations of the audio signals' features (lost packets masks, waveforms, spectrograms) and by the possibility to playback arbitrary portions of the reconstructed audio tracks to evaluate their perceived quality. The quantitative analysis of the PLC algorithm outcome is supported through multiple metrics produced by the tool for each of the reconstructed audio files, like the Mean Squared Error (MSE), the Mean Absolute Error (MAE), the Spectral Energy, the PEAQ (Perceptual Evaluation of Audio Quality). Being conceived as a framework, the tool can be easily extended by providing additional implementations of the different object types (PLS algorithm, PLC algorithm, output analysis metric).

Before the implementation of this user interface, the only way to interact with the tool was by a Jupyter Notebook, where inputs had to be provided by directly editing the code. This required the user to have a fully functional development environment, with all the complexity implied and it also required the user to be familiar with Python programming. Moreover, there was no aid in the interpretation of results, so the only option was to consult the raw output audio files and images. This type of interaction forced the user to consider one element at a time, precluding the possibility of increasing the quality of the analysis by considering multiple aspects at once. During the design phase of the GUI particular attention was paid to modularity and to the capability of the GUI to automatically adapt to testbench extensions, especially in terms of algorithms and settings, with the purpose of making maintenance and future enhancements easier. On top of this, different types of GUI were evaluated, and their pros and cons were carefully weighted up. Finally, the decision was made to go for the development of a web application because of the many advantages offered by the web technology, like a wide range of deployment modes (ranging from a local standalone environment to big, distributed infrastructures), the portability of the application on different platforms, the ease of installation. To make the distribution of the software as easy as possible, despite the very large number of dependencies in terms of required libraries, it was decided to leverage containers technology. The development of the application presented several challenges, especially because of the large amount of data inherently involved by audio processing. Even short audio tracks are composed of millions of samples that needed to be represented visually as waveforms and/or streamed as sound in real-time, so to reach the desired performance subsampling techniques and multiple optimizations had to be adopted. Another challenging aspect was the wide spectrum of technologies and programming languages to be learnt in a relatively short period of time.

1 Introduction

Nowadays an increasing number of applications exploit the public Internet to implement remote interactions with users or with other systems and those dealing with digital audio make no exception. They range from streaming applications to audio editing programs, to sophisticated solutions in the domain of the so-called Internet of Sounds. In most cases the Internet is perceived by the users as a quite reliable infrastructure because transmission protocols like TCP build transport reliability on top of IP protocol, which by itself does not provide any guarantee on transport. The drawback of this reliability is however an increased latency caused by retransmission of lost packets. While this can be acceptable for most applications, it becomes a severe problem in applications working with video and audio media, which do not tolerate very well high latency. Actually, most of these applications prefer to live with a not reliable protocol like UDP in exchange for reduced latency, but this implies that packets can, and typically will be lost. The rate of this loss is strictly related to the congestion or the presence of malfunctioning or broken devices in the network between the two communication endpoints. In video streams packet loss causes “freezing” or image quality degradation, while in audio they produce annoying pops and crackles.

As the acronym suggests, PLC (Packet Loss Concealing) algorithms aim at reducing this problem by reconstructing the lost parts of the audio signal using different techniques which all try to “predict” the missing signal based on previously received packets and mitigate the discontinuity created by the impulsive zeroing created by the frame’s losses.

The domain where PLC techniques were applied throughout this work it that of non-speech audio signals, as speech reconstruction requires peculiar techniques that were out of scope for the research from which this thesis originated.

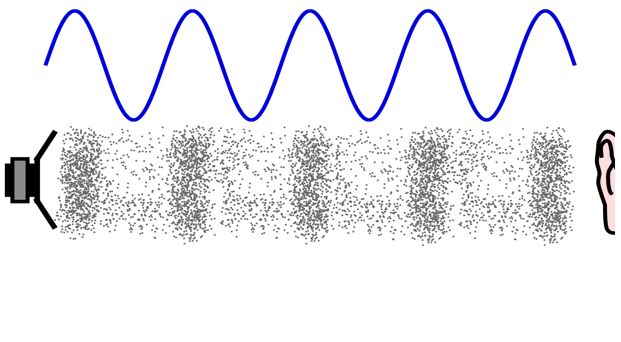
Currently only WAV audio format is supported but extension to other formats can easily be achieved by pre-processing the input files and normalizing them into WAV format.

1.1 Digital audio fundamental concepts

Sound transmission is a physical phenomenon consisting of a pressure wave propagating across a gaseous, liquid, or solid medium. The frequency, amplitude and phase of this wave can change over time, originating a frequency spectrum. Sound can be perceived by the human ear only if any of the frequencies in the spectrum falls within a certain range, called human auditory field.

Digital audio is a representation of sound in a digital form. Typically, the pressure wave amplitude is sampled at regular intervals and its value is mapped to a discrete range of values, thus originating a stream of numbers.

According to the Nyquist–Shannon sampling theorem, in order to sample all frequencies included in the human auditory field an audio signal must be sampled at a rate of 44.1 kHz.



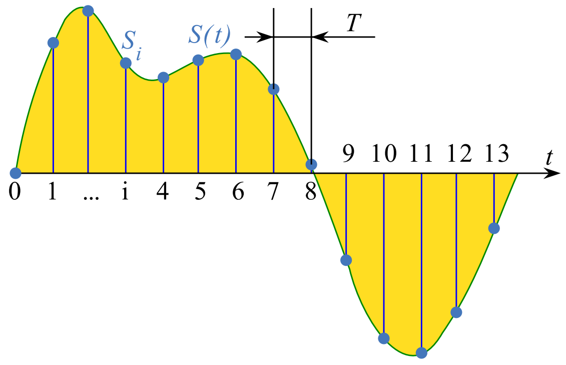


Figure 1 – ***a)*** *Wave pressure affecting air density [1]* ***b)*** *example of wave sampling [2]*

1.2 Packet loss simulation

The packet loss can be simulated by randomly dropping some packets from the original audio stream based on a given probability distribution. The random variable represents a packet transmission whose possible outcomes are successful transmission or packet loss.

1.2.1 Binomial probability distribution

In this model the probability of the random variable to take the value corresponding to a lost packet follows a binomial probability distribution, which represents the sum of a series of multiple independent and identically distributed Bernoulli trials. A Bernoulli trial is a **random experiment** with exactly two possible outcomes (success/failure), whose probabilities are p [0, 1] and q = 1 – p [0, 1].

The resulting probability to get **k** success/failure events in **n** trials is described by the following formula based on the binomial coefficients.

$$f(k,n,p) = \Pr(k;n,p) = \Pr(X=k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 0, 1, 2, \ldots, n$, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Figure 2 – Probability to get k success/failure events in n trials.

1.2.2 Gilbert-Elliot loss model

Compared to models based on a single probability distribution, Gilbert-Elliot model has the big advantage of keeping into account the correlation between loss events. While in the binomial distribution loss model each loss event is unrelated to the previous one and therefore its probability is independent, in the Gilbert-Elliot model the probability of each loss event is conditioned by the previous one.

This behaviour is in line with the assumptions of the Markov's chains and in fact Gilbert-Elliot model can be considered a two-state Markov's chain whose graph is the one depicted below.

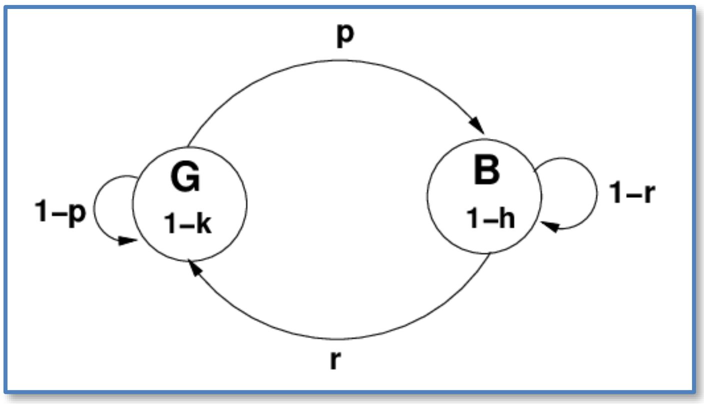


Figure 3 – Gilbert-Elliot's model of bursty packet loss as a Markov chain [3]

This model is much more accurate compared to single distribution models, because in the real-world packet loss is usually caused by failures or congestion in the transmission network and therefore tends to happen in bursts.

So, when a first loss is experienced, it is more likely that subsequent packets are lost compared to a transmission success scenario, because the system has usually transitioned from a working to a non-working state. In the same way, when the last packet has been transmitted successfully this is an indication that the system is working properly and therefore the probability that the next packet will not be lost is higher than when the system is in a non-working state.

1.3 Packet Loss Concealment (PLC) algorithms

PLC algorithms are typically needed in real-time data transmission contexts, and their job is to mask ("conceal") any loss that may occur to one or more packets in a manner that is transparent to the application consuming the data stream. Any PLC algorithm needs to satisfy the following requirements:

- Its execution time needs to be smaller than the period between the consumption of packets.
- Its output doesn't need to match exactly the content of the corresponding packet. However, when put back in place of the lost packet, it needs to be perceptually indistinguishable from the original audio stream.

There are many ways to perform PLC on audio signals. From simpler waveform-based solutions such as waveform repetition and PSOLA to more advanced autoregressive models. Recently, Deep Learning based PLC has begun to arise in the literature with some noticeable examples both in speech and music.

Regardless of the type, they all use the audio data received immediately before the lost packet as the input to the algorithm, which will output the number of samples contained in the lost packet.

1.3.1 Zeros PLC

The lost samples are replaced by zeros. This algorithm is used to generate the audio file representing the result of the loss simulation. It can be useful as a benchmark for the files reconstructed using other PLC algorithms.

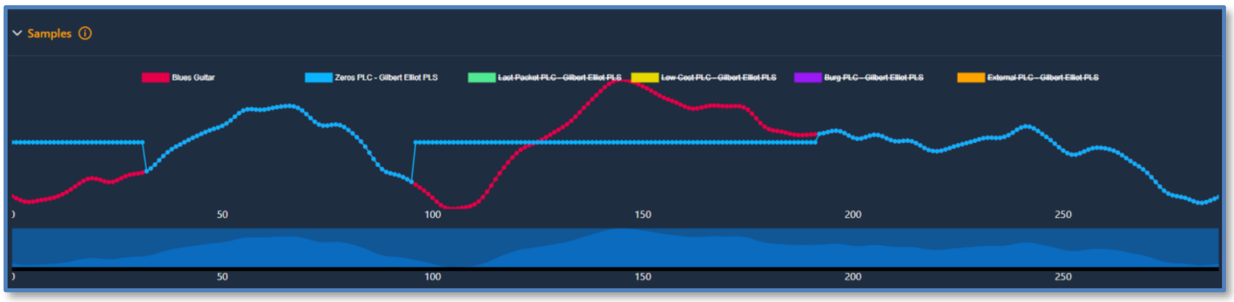


Figure 4 – Example of Zeros PLC algorithm. The algorithm simply produces an audio file corresponding to the loss simulation where for any lost packet the signal is zeroed.

1.3.2 Last Packet PLC

The lost samples are replaced by the last received packet. This is the easiest PLC technique.

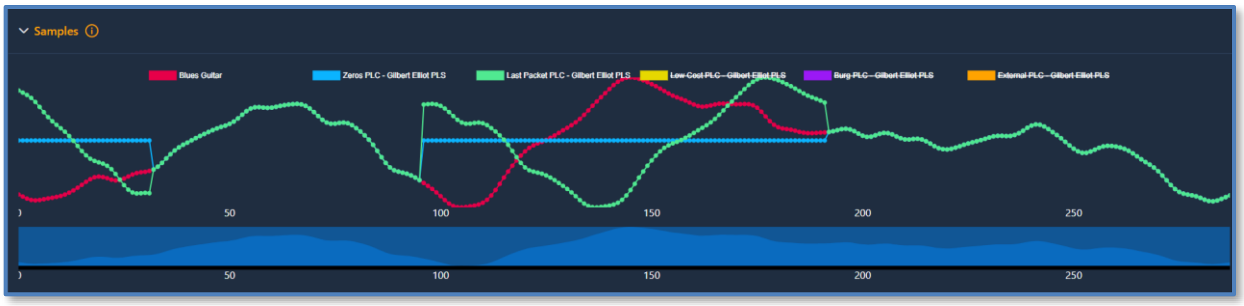


Figure 5 – Example of the Last Packet PLC algorithm. As clearly shown by this picture (green waveform), the algorithm uses the signal from the last packets preceding the loss to reconstruct the missing portion of the file. This is particularly evident in the range of samples going from sample 70 to sample 140 of the zoomed region.

1.3.3 Low Cost PLC

This module is an implementation of the algorithm proposed in [3].

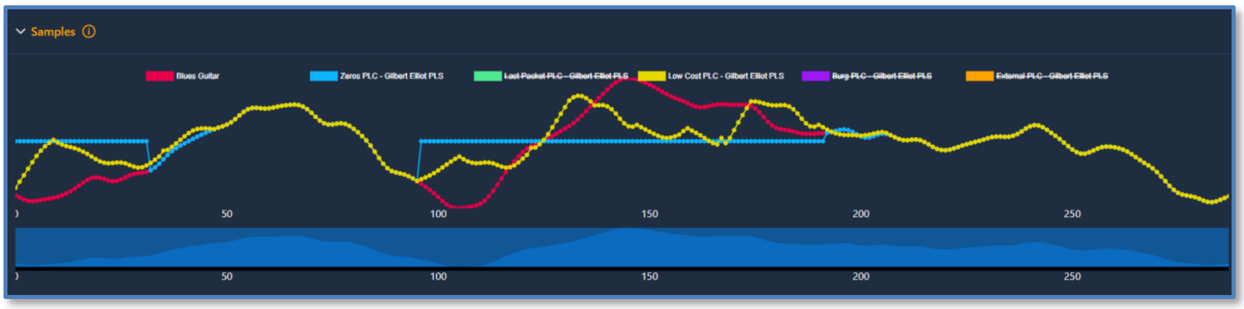


Figure 6 *– Example of a reconstruction of the signal obtained by applying the low-cost algorithm. Despite not being extremely accurate, the discontinuities at the edges of the loss burst are smoothed and the peeks of the reconstructed signal match those of the original one.*

1.3.4 Burg PLC

Python bindings for the C++ implementation of the Burg method [4].

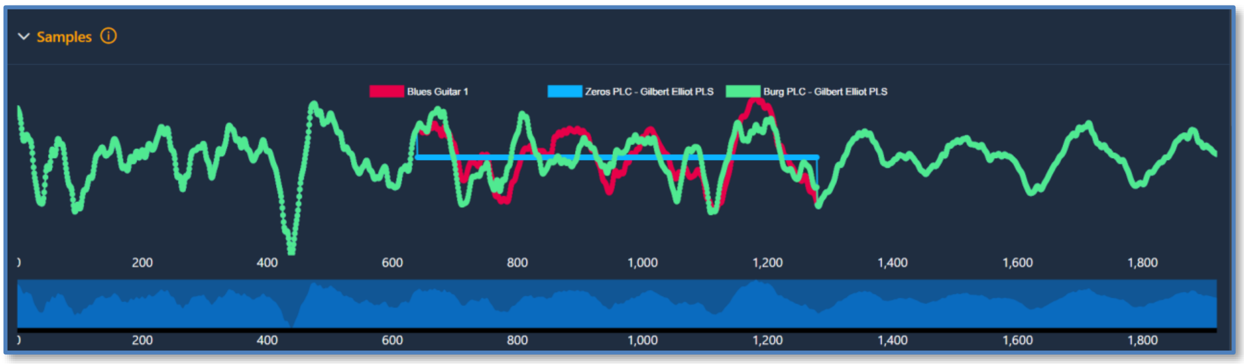


Figure 7 *– Example of a very good quality reconstruction of the signal obtained by Burg's algorithm with train size=1024 and order=512; in red the original file waveform, in green the one generated by Burg's algorithm, in blue the signal with the lost packets.*

1.3.5 Deep Learning PLC

Implementation of the algorithm proposed in [4]. Compared to the other algorithms it is more expensive due to the complex processing performed.

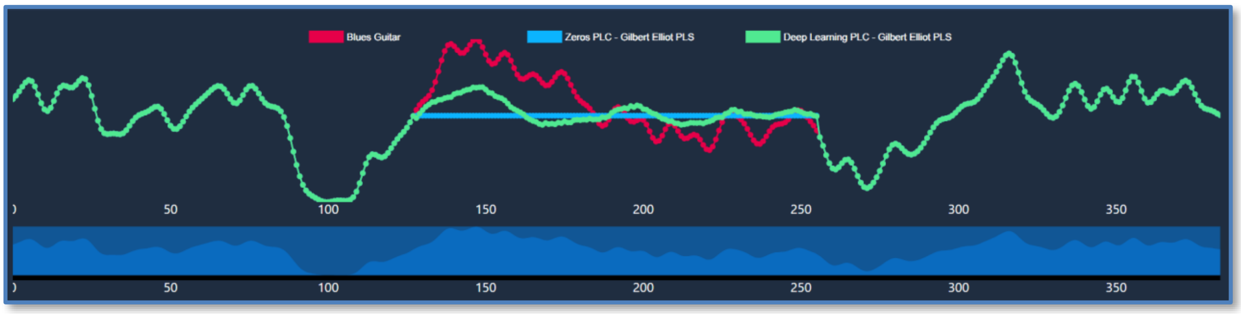


Figure 8 *– Example of Deep Learning PLC reconstruction.*

1.3.6 External PLC

Python bindings for C++ to simplify the integration of existing algorithms.

1.4 Output analysis metrics

The metrics implemented by the last version PLC Testbench tool at the time of writing this thesis are:

Mean Squared Error (MSE): is a measure representing the mean value of the squared differences between paired observations of the same phenomenon. The observation pairs can be represented by an estimated value versus an actual value or by measures of the same parameter collected at different times, or any other pair of values that make sense in the context of the statistical analysis being performed. It is computed as:

$$\boxed{\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2.}$$

Mean Absolute Error (MAE): same as above, but instead of the squared error the absolute value is used.

$$\boxed{\mathrm{MAE} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}}$$

Spectral Energy (SE):
Represents a magnitude difference signal calculated from the DFT (Discrete Fourier Transform) energies of the reference and original signals obtained by using a given window and hop size.

Perceptual Evaluation of Audio Quality: is a standardized algorithm for objectively measuring perceived audio quality. Among all the metrics implemented out-of-the-box in the PLC Testbench tool, this is certainly the most specialized and most effective one for the domain of audio signals processing because it is based on a psychoacoustic model of the human ear and brain. The model takes into consideration human auditory field and its limits and compares a reference signal with a test signal, ignoring differences that are not considered as perceptible according to the model's rules. PEAQ transforms the inputs by applying filter banks to the Discrete Fourier Transform (DFT) of the signals, producing an intermediate output in the form of a set of variables, each one capturing a different psychoacoustic dimension. Finally, the output variables are used to feed a neural network that simulates the cognitive processes happening inside the human brain, which translates the input variables into an overall quality score named Objective Difference Grade (ODG).

The algorithm comes in two different versions: the basic one, less computationally expensive, and the advanced one, more accurate but computationally heavier and therefore not always applicable to real-time signals.

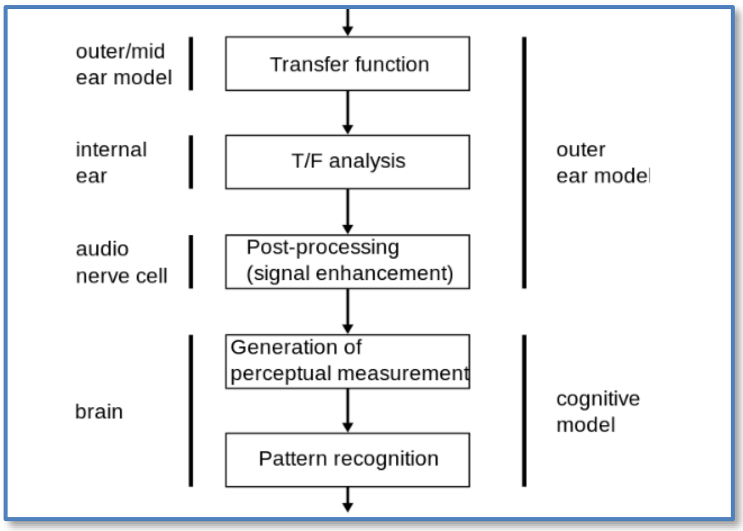


Figure 9 – Perceptual Evaluation of Audio Quality. [5]

2 PLC Testbench UI

2.1 Web GUI’s main functions

The purpose of the user interface for the PLC Testbench is to make the tool easier to use, reduce the time required to consult results and increase the quantity and quality of information obtained from the analysis of results.

Before the implementation of this user interface, the only available interaction took place directly in a Jupyter Notebook, where inputs had to be provided directly by modifying the source code. Moreover, there was no aid in the interpretation of results, so the only option was consulting the raw output audio files and images. This type of interaction is suboptimal, especially in terms of consulting the results, as the user is forced to consider one element at a time, precluding the possibility of increasing the quality of the analysis by considering multiple aspects at once.

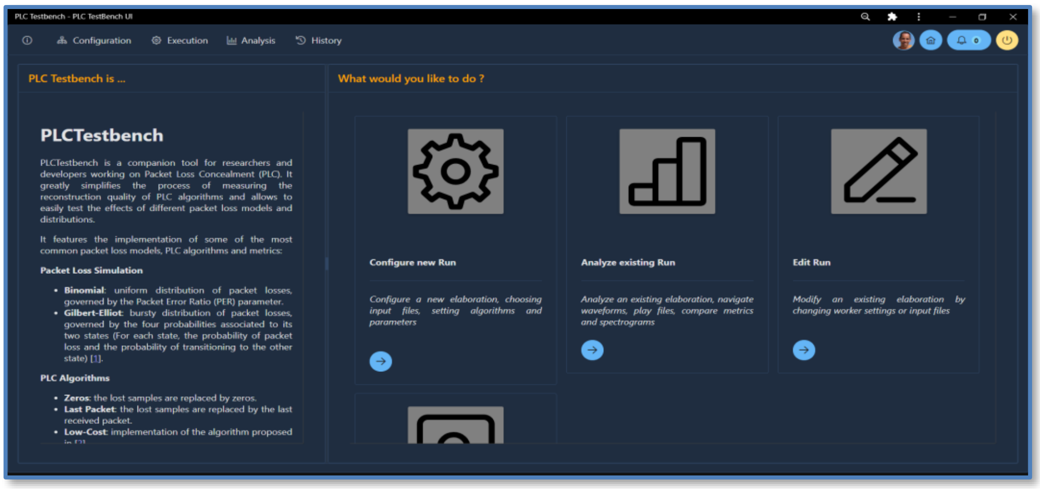
The user interface exposes all the functions provided by the underlying PLC Testbench and can be conceptually divided into five modules:

1. Landing page
2. Configuration of the input data
3. Progress monitoring
4. Results analysis
5. Elaborations search and inquiry

2.1.1 Landing page

The landing page of the application is divided into two sections: one contains a document providing a general overview of the PLC Testbench tool, the other contains a list of the main tasks supported by the application.

In the context of the PLC Testbench UI, the purpose of the landing page is helping the user to develop a mental model of the application as accurate as possible and putting the user in condition to operate on the application as quickly as possible. The help section of the page addresses the first objective, while the tasks list addresses the second one.



PLC Testbench is ...

PLCTestbench

PLCTestbench is a companion tool for researchers and developers working on Packet Loss Concealment (PLC). It greatly simplifies the process of measuring the reconstruction quality of PLC algorithms and allows to easily test the effects of different packet loss models and distributions.

It features the implementation of some of the most common packet loss models, PLC algorithms and metrics:

Packet Loss Simulation

- **Binomial**: uniform distribution of packet losses, governed by the Packet Error Ratio (PER) parameter.
- **Gilbert-Elliot**: bursty distribution of packet losses, governed by the four probabilities associated to its two states (For each state, the probability of packet loss and the probability of transitioning to the other state) [1].

PLC Algorithms

- **Zeros**: the lost samples are replaced by zeros.
- **Last Packet**: the lost samples are replaced by the last received packet.
- **Low-Cost**: implementation of the algorithm proposed

What would you like to do ?

Configure new Run

Configure a new elaboration, choosing input files, setting algorithms and parameters

Analyze existing Run

Analyze an existing elaboration, navigate waveforms, play files, compare metrics and spectrograms

Edit Run

Modify an existing elaboration by changing worker settings or input files

***Figure 10** – Landing page of the application. On the left side an overview of the PLC Testbench tool is presented. On the right side the user can find an entry point to the main use cases supported by the application.*

2.1.2 Configuration of the input data

The purpose of this module of the GUI is to allow the selection of the files to be processed by a run of the

PLC Testbench tool and to allow the configuration of the sets of packet loss, reconstruction and evaluation algorithms to be applied to each input file with the corresponding parametrizations.

Given the multi-step nature of the task and the high number of selections and choices involved in the process supported by this part of the interface, while designing it the focus was put on making the interaction as simple and intuitive as possible.

In order to achieve this goal, it was decided to model the main steps of the process through a wizard-based interaction, where each step has a clear and small purpose, and the user goes through the sequence of steps in a logical manner. Splitting a complex task into multiple simpler tasks has several advantages like decreasing the likelihood of mistakes, giving a clearer context for error reporting, making it easier for the user to develop a more accurate mental model of the software, giving the users an update on their progress in the procedure, reducing the cognitive load. At each step the coherence of the data is enforced reporting any error to the user before the next step is undertaken. This way any wrong or missing information is pointed out in a limited scope and therefore the problem becomes easier to understand and fix.

From a technical side, this was identified from the very beginning as the part of the GUI which is more likely to be impacted by extensions or future developments, so it looked crucial to make it as flexible and auto-adaptive as possible. This meant that the interface had to be able to deal with the addition of new algorithms with arbitrary parameters in a transparent way, without requiring any code adjustment. The fulfillment of this requirement was obtained by exploiting the powerful introspective functions of python programming language, that allow to analyze code structure at runtime. Through these APIs it is possible to discover relationships between software components.

In the PLC Testbench UI, it is enough for any implementation of a new algorithm to inherit from the base class of if its own category, be it a loss simulation or a packet loss concealing or an output analysis one, for it to get automatically exposed in the corresponding screens with all its properties.

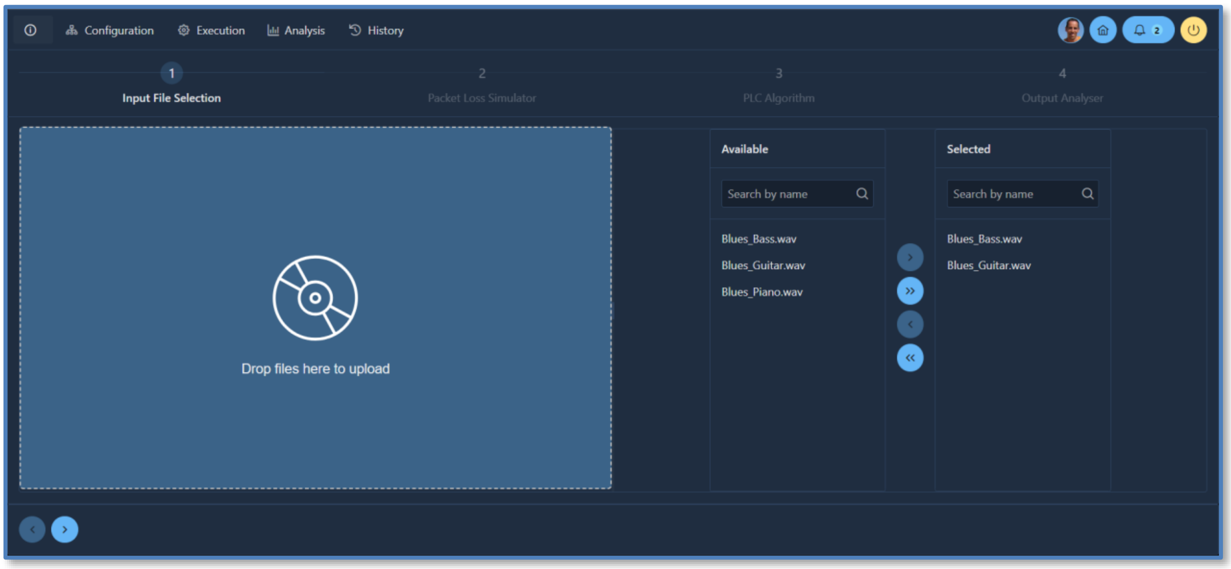
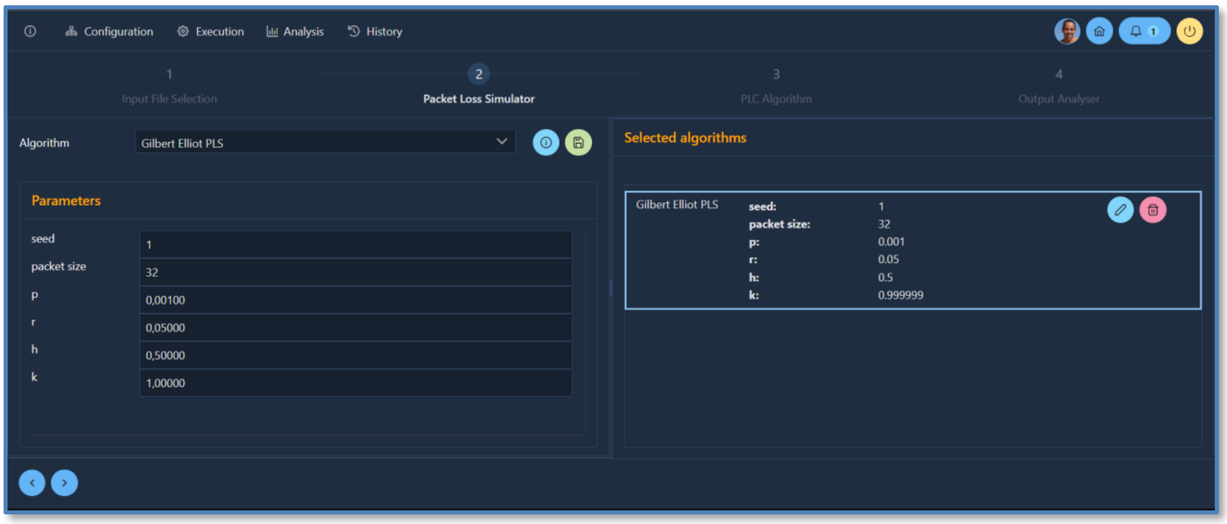


Figure 11 *– Application screen to configure an elaboration: input file selection phase.*



Algorithm: Gilbert Elliot PLS

seed	1
packet size	32
p	0,00100
r	0,05000
h	0,50000
k	1,00000

Gilbert Elliot PLS

seed:	1
packet size:	32
p:	0.001
r:	0.05
h:	0.5
k:	0.999999

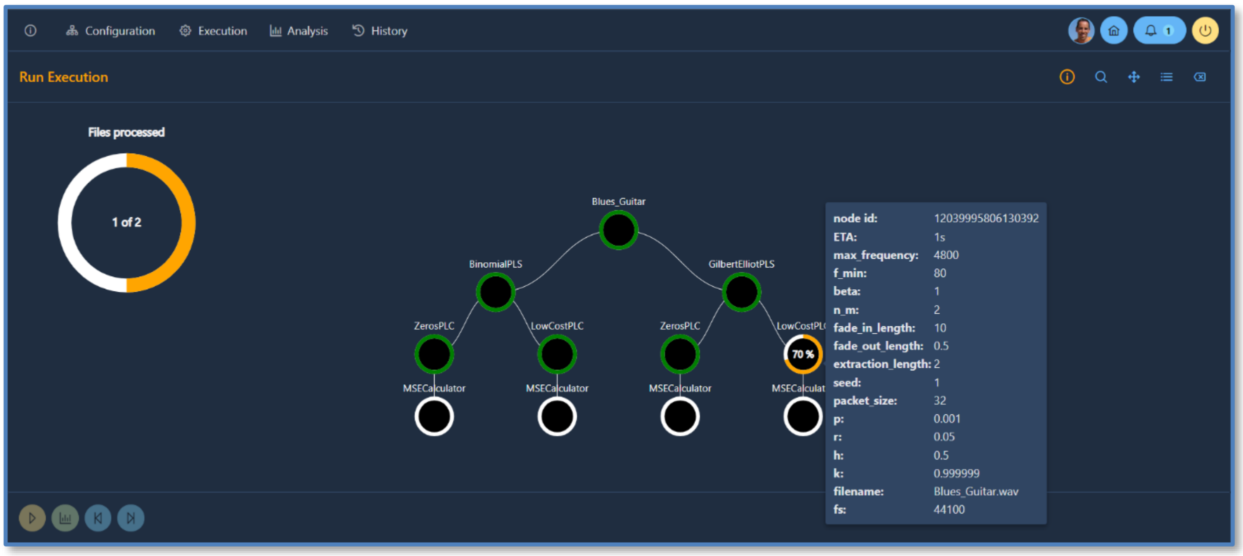
Figure 12 – Application screen to configure an elaboration: module's parameters setting. The interface is self-adapting to the underlying PLC Testbench code so that adding new modules or new module parameters can be handled automatically.

2.1.3 Execution management and monitoring

This module of the GUI is intended to allow the user to monitor the state of an in-progress elaboration, providing some sort of progress bar and information like the ETA (Estimated Time of Arrival) for the sub-steps.

Each elaboration of the PLC Testbench tool is named a "Run" and it contains a snapshot of all the input data and all the generated output results, like the loss simulation masks, the reconstructed files and the metrics evaluated on each reconstructed file. Each of the elaboration's steps can be mapped to a tree node where the root node represents an input file. The overall Run is therefore a forest of trees.

Since the elaboration is identical for all the input files in terms of the algorithms that are executed on the original track and it is sequential, displaying the entire structure of a Run all the time would imply cluttering the interface with redundant information. It was therefore decided to represent progress at two different levels, that is within each input file tree and globally.



node id:	12039995806130392
ETA:	1s
max_frequency:	4800
f_min:	80
beta:	1
n_m:	2
fade_in_length:	10
fade_out_length:	0.5
extraction_length:	2
seed:	1
packet_size:	32
p:	0.001
r:	0.05
h:	0.5
k:	0.999999
filename:	Blues_Guitar.wav
fs:	44100

Figure 13 – Application screen to manage and monitor elaborations

2.1.4 Analysis of the execution's results

This module was considered from the very beginning as the most important one, so a great part of the time spent in the analysis and development of the PLC Testbench UI was dedicated to it. In this case the focus was on providing an easy way to analyze the results from both qualitative and quantitative perspectives.

Qualitative analysis is mainly represented by browsing and visually comparing the waveforms in the regions where packets have been lost and by listening to the audio files generated from the PLC algorithms.

Quantitative analysis is mainly represented by the numerical comparison of the performance achieved by the PLC algorithms according to a given metric like the Mean Squared Error, the Mean Absolute Error, the Spectral Energy, the Perceptual Evaluation of Audio Quality (PEAQ) with respect to the specified input parameters.

Another extremely important aspect was providing the capability to integrate and correlate as much as possible the different types of information to get deeper insights into the behavior of the algorithms being tested.

The main requirements for this use case were:

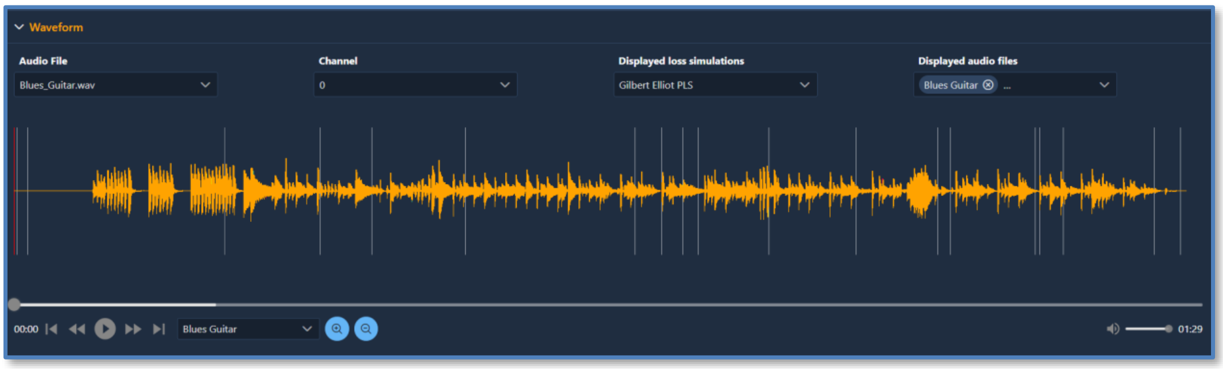
- clearly show the lost portions of the original file;
- allow easy comparison of the waveforms ranging from very high level to sample level detail;
- give the possibility to display every combination of waveforms in the comparison;
- give the possibility to play any audio track, original or reconstructed setting the playhead at any position by simply clicking on the waveform;
- support a very wide range of zoom levels including a sample level one;
- display visually the metrics calculated on the reconstructed audio files;
- display a spectrogram of the audio file, be it the original or a reconstructed one.

The need to display multiple information at the same time made a dashboard interface look more appropriate and appealing in this case. The dashboard was organized in four sections where the first one is focused on representing the waveforms of all the audio versions of an input file and providing navigation across the files and zoom functions. Lost samples regions are displayed together with the waveform and when clicked the corresponding detailed data are loaded in the section below.

The second section provides the ability to inspect the waveform at sample level in any region where some samples were lost. This is extremely useful to evaluate from a qualitative point of view the behaviour of the algorithm and to cross check the correctness of the algorithm's implementation, as visual representation makes any misbehaviour much more apparent.

The third section is intended to display the metrics calculated on the reconstructed versions of the analysed file and is composed of two different charts, one containing the figures from metrics producing time series, the other representing the figures from metrics producing single values.

Finally, the fourth section displays the spectrogram of the current audio file, representing the distribution of the frequencies composing the signal and their relative and absolute intensity in the form of a colour map.



Waveform

Audio File: Blues_Guitar.wav

Channel: 0

Displayed loss simulations: Gilbert Elliot PLS

Displayed audio files: Blues Guitar ...

Figure 14 – Waveforms and packet loss visualization.

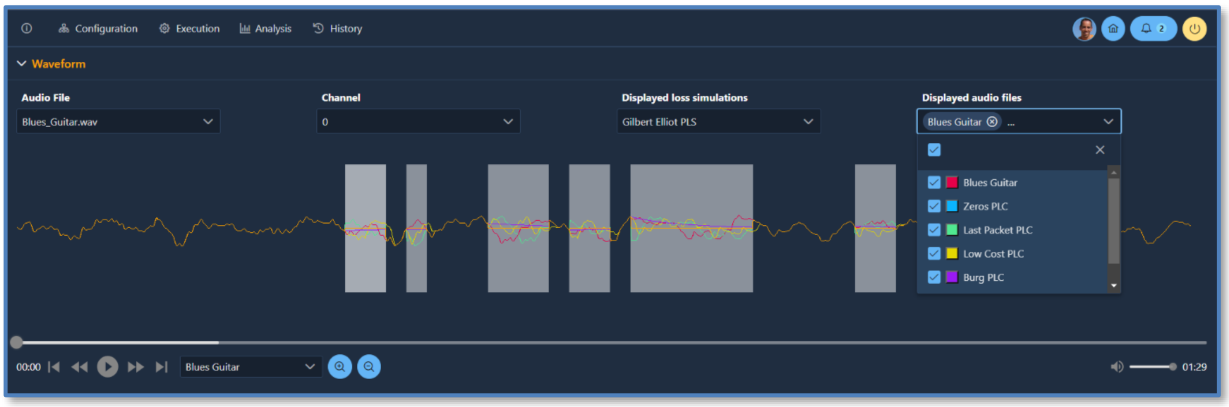


Figure 15 – *Zoomed waveforms and packet loss regions.*

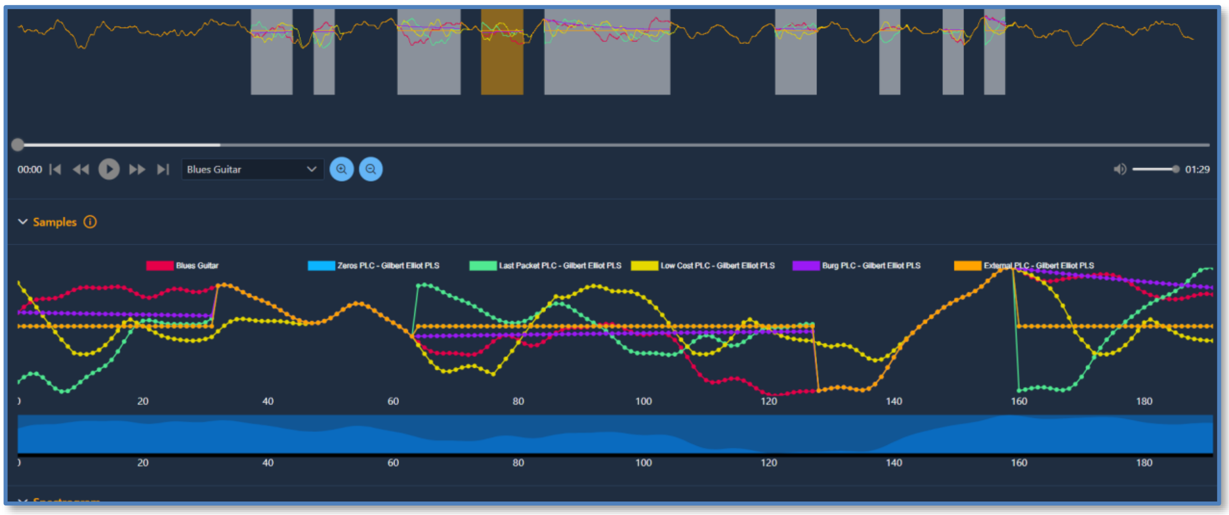

00:00
Blues Guitar
01:29
Samples
Blues Guitar
Zeros PLC - Gilbert Elliot PLS
Last Packet PLC - Gilbert Elliot PLS
Low Cost PLC - Gilbert Elliot PLS
Burg PLC - Gilbert Elliot PLS
External PLC - Gilbert Elliot PLS
20
40
60
80
100
120
140
160
180

Figure 16 – *Sample visualizer: when a specific packet loss is selected by clicking on it, the sample visualizer displays a view of the signal that can be zoomed at single sample level for each version of the audio file, original or reconstructed.*

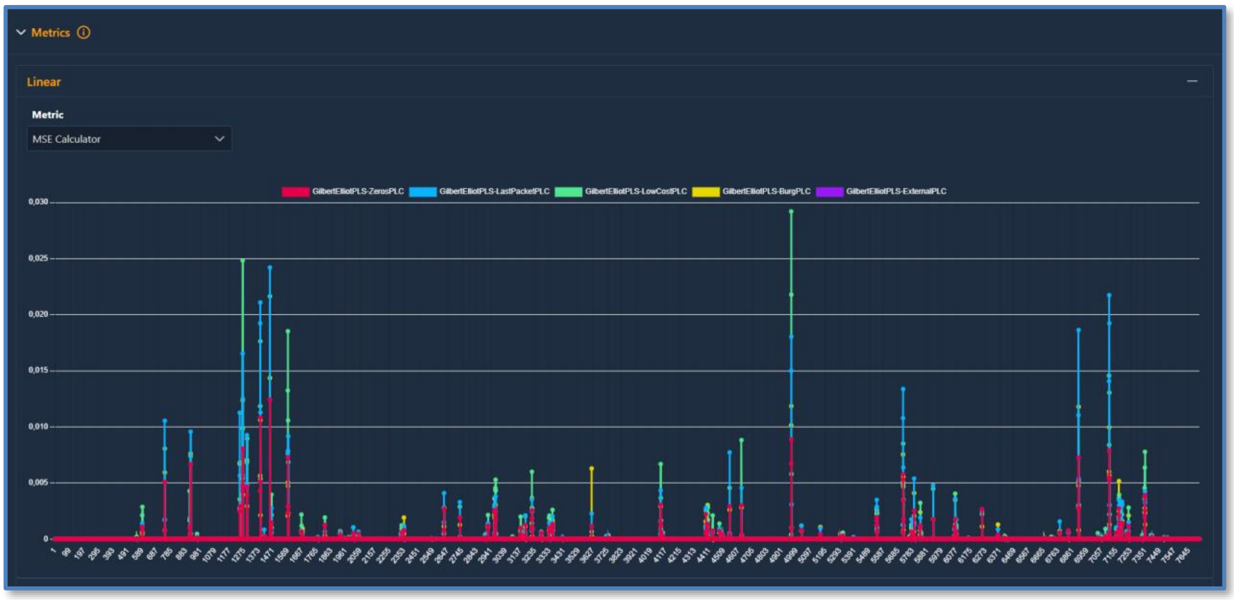

Metrics
Linear
Metric
MSE Calculator
GilbertElliotPLS-ZerosPLC
GilbertElliotPLS-LastPacketPLC
GilbertElliotPLS-LowCostPLC
GilbertElliotPLS-BurgPLC
GilbertElliotPLS-ExternalPLC
0,030
0,025
0,020
0,015
0,010
0,005
0

Figure 17 – *Metrics visualizer: linear metric.*

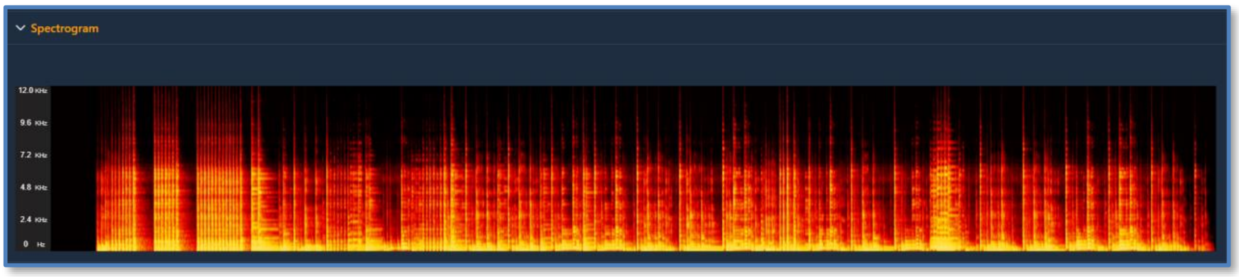


Figure 18 – Spectrogram: a view presenting a spectrogram of the selected portion of the file in the form of a heat map

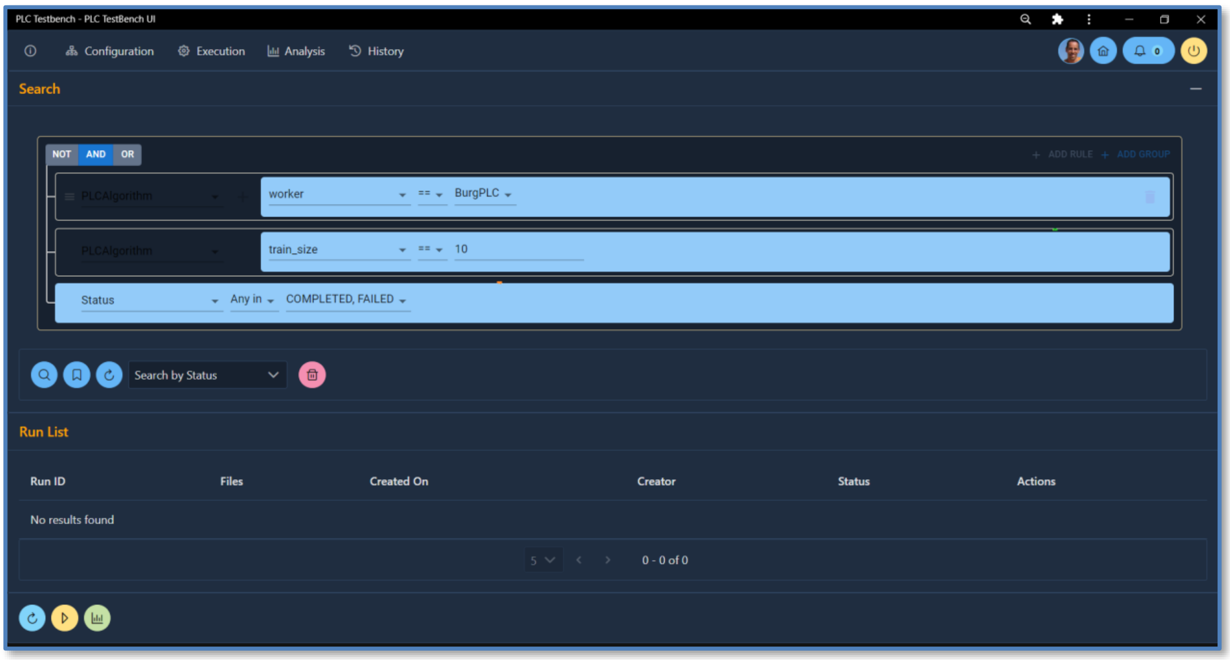
2.1.5 Elaboration search and inquiry

After the concept of “Run” emerged as a way to organize input and output of a specific elaboration, the need to design some search function became more and more clear. The purpose of the inquiry function was not only to allow recalling previous elaborations, but also to allow finding out similar elaborations for comparison purpose.

In order to make searching as easy as possible but at the same time keep it very powerful, flexible, and resilient to code changes or extensions, a query builder component has been exploited. The component allows to graphically create a filter by adding multiple conditions and connecting them with the logical and/or operators. It is also possible to parenthesize the conditions by nesting them in groups. All the relevant fields, both at run level and at configuration level can be referenced in each condition and they can be matched against specific values by using a wide set of operators, depending on the type of the field.

Given the high number of parameters involved in the configuration of a PLC Testbench run, queries can quickly become quite complex and writing them from scratch each time would be frustrating and time-consuming. To address this problem filter saving functions have been provided, allowing to store each filter persistently in the application database, while associating a meaningful name to them.

Finally, the results are presented in the bottom section of the page.



PLCAlgorithm | worker == BurgPLC

PLCAlgorithm | train_size == 10

Status Any in COMPLETED, FAILED

Search by Status

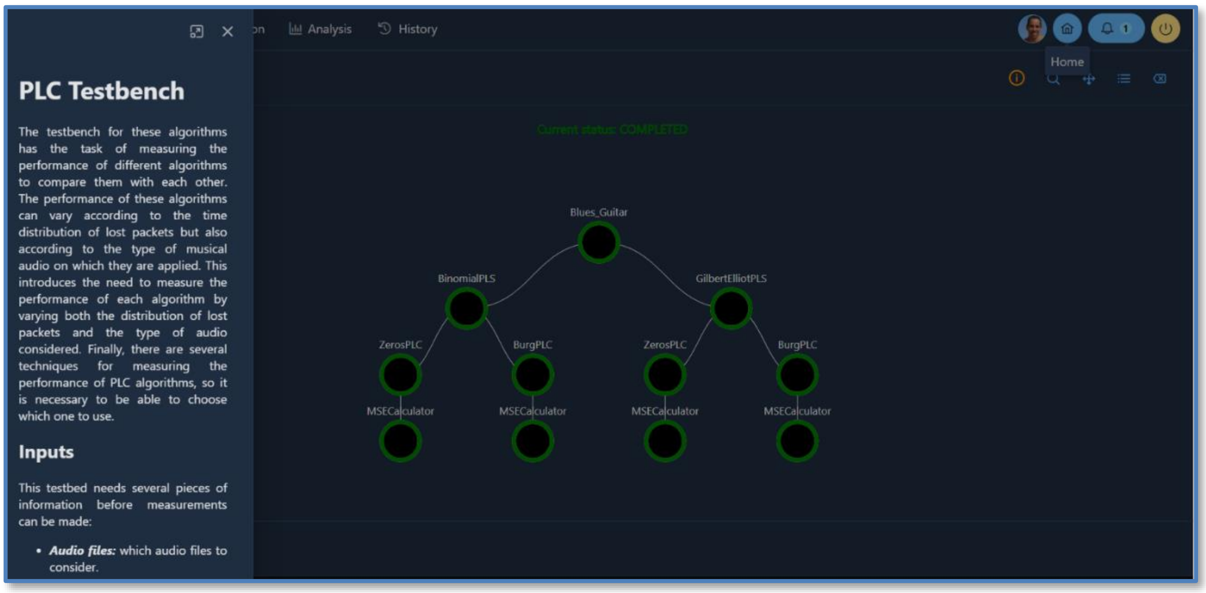
Run ID	Files	Created On	Creator	Status	Actions
No results found					

Figure 19 – Run History page. In the search section custom queries can be built with the help of a query composer. In the Run list section the results of the search operation are displayed.

2.1.6 Utility functionalities

Several utility functionalities have been developed to keep the user constantly informed about relevant information and events. For example, by clicking on the info button in the top menu bar it is possible to display an help page about the currently selected functions.

As each elaboration can take a significant amount of time, depending on the number of input files, the duration of each audio track and the parameters of the simulated loss, reconstruction algorithm and output metric evaluation, a notification menu has been added to the top menu bar. When an elaboration is complete a notification is dropped in the user "mailbox" and the number of unread notifications is displayed on the component. This is particularly helpful because the results of an elaboration can be analyzed only once the elaboration is completed, therefore a push notification mechanism can free the users from having to regularly check the elaboration status while carrying out other activities in the application.



PLC Testbench

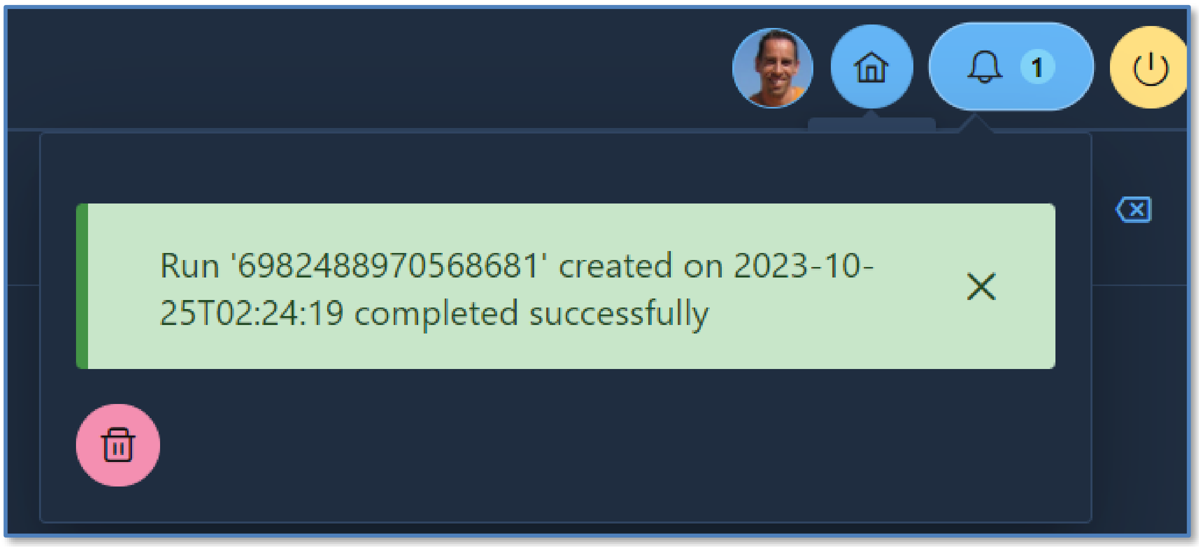
The testbench for these algorithms has the task of measuring the performance of different algorithms to compare them with each other. The performance of these algorithms can vary according to the time distribution of lost packets but also according to the type of musical audio on which they are applied. This introduces the need to measure the performance of each algorithm by varying both the distribution of lost packets and the type of audio considered. Finally, there are several techniques for measuring the performance of PLC algorithms, so it is necessary to be able to choose which one to use.

Inputs

This testbed needs several pieces of information before measurements can be made:

- ***Audio files:*** which audio files to consider.

Figure 20 – *In each context a relevant help page is displayed in the side bar.*



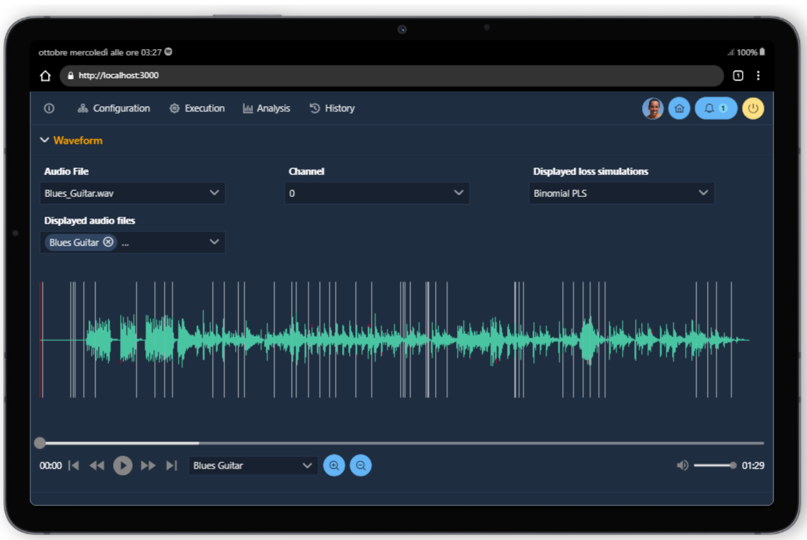
Run '6982488970568681' created on 2023-10-25T02:24:19 completed successfully

Figure 21 – *Push notifications: when relevant events happen (e.g., an elaboration is completed) a notification is displayed in this area.*

2.1.7 Responsive layout and PWA (Progressive Web Application)

The application has a responsive layout that can adapt gracefully to most screen dimensions, ranging from big workstation monitors to tablets and smartphones screens.

The application can be installed as a Progressive Web Application giving the user a "desktop-like" experience, where the application can be run by just clicking on a desktop icon and screen space is not wasted by the browser toolbars.



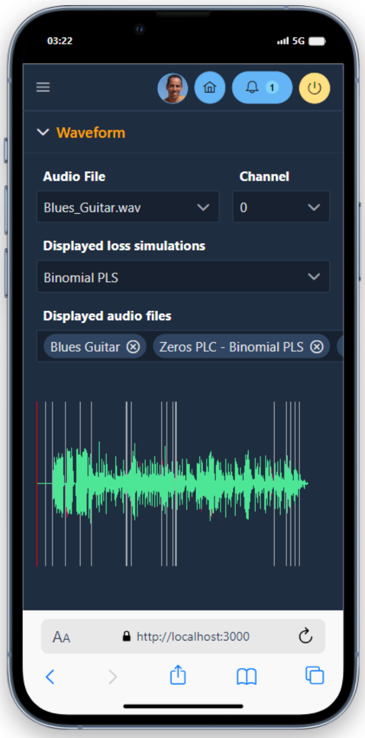


Figure 22 – a) Tablet layout. *b) Smartphone layout.*

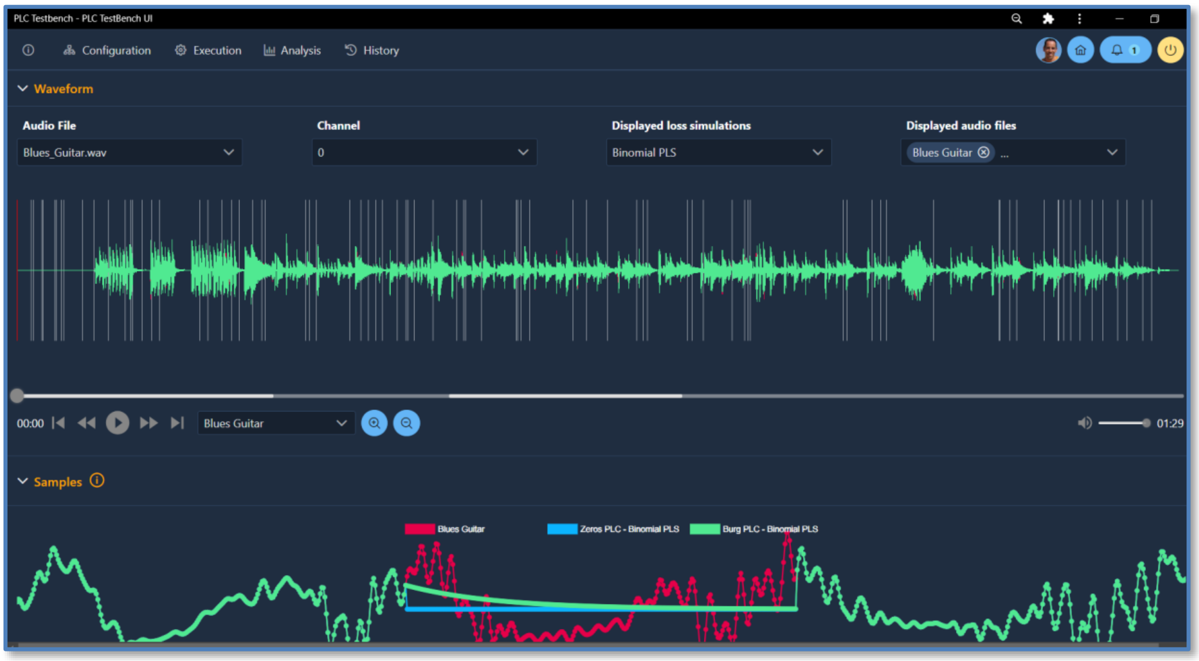


Figure 23 – Progressive Web App layout.

Figure 24 – Progressive Web App launch shortcut.

2.2 Development Methodology

The development of the PLC Testbench spanned over a period of several months and implied a lot of technical and architectural choices, the main of which are summarized in the following chapters. The application was developed iteratively and during the process several major refactoring were made, to leverage the insights gradually gained on the domain and to accommodate the PLC testbench evolution.

2.2.1 Technical choices

During the design phase different types of GUI were evaluated and their pros and cons were carefully weighted up. Finally, the decision was made to go for the development of a web application because of the many advantages offered by the web technology, like a wide range of deployment modes (ranging from a local standalone environment to big, distributed infrastructures), the portability of the application on different platforms, the ease of installation.

2.2.2 Project architecture

The user interface application is made of a Web application composed of two layers: one managing the user interface representation and the interaction with the user, the other providing the backend services to be consumed by the graphical components.

The application therefore can be used both as a standalone application by deploying it on the user machine or in a multi-user environment where the application is deployed on one or more remote servers.

The two layers exchange information via a RESTful API which, if needed, can be exploited also by third-party services. The API is secured by using OAuth2 protocol.

In the presentation layer, modularity has been achieved through the creation of specialized but highly customizable components that can easily be reused across different pages.

In the backend layer, modularity is provided by the extensibility of the RESTful API where new functions can be “plugged in” by simply adding new endpoints. This layer is also hiding the details of the interaction with the PLC Testbench from the presentation layer, providing an abstract and stable view of the library functions.

Persistence is based on a NoSQL database, where the data are stored in JSON format in order to provide higher flexibility to accommodate future evolution of the schema. Both on-premises and cloud databases are supported.

The GUI adapts automatically to the PLC Testbench library by using introspection to retrieve the list of the algorithms used for packet loss, PLC, and output analysis. This way any extension made to the underlying tool does not require any manual change to the UI code.

The application architecture has been designed to be as stateless as possible and thus supports very well horizontal scalability, which means it is possible to split the workload among as many instances of the application as it is need by providing proper load balancing in front of them.

Scalability together with containerized distribution makes the application suitable for deployment in an orchestrated environment.

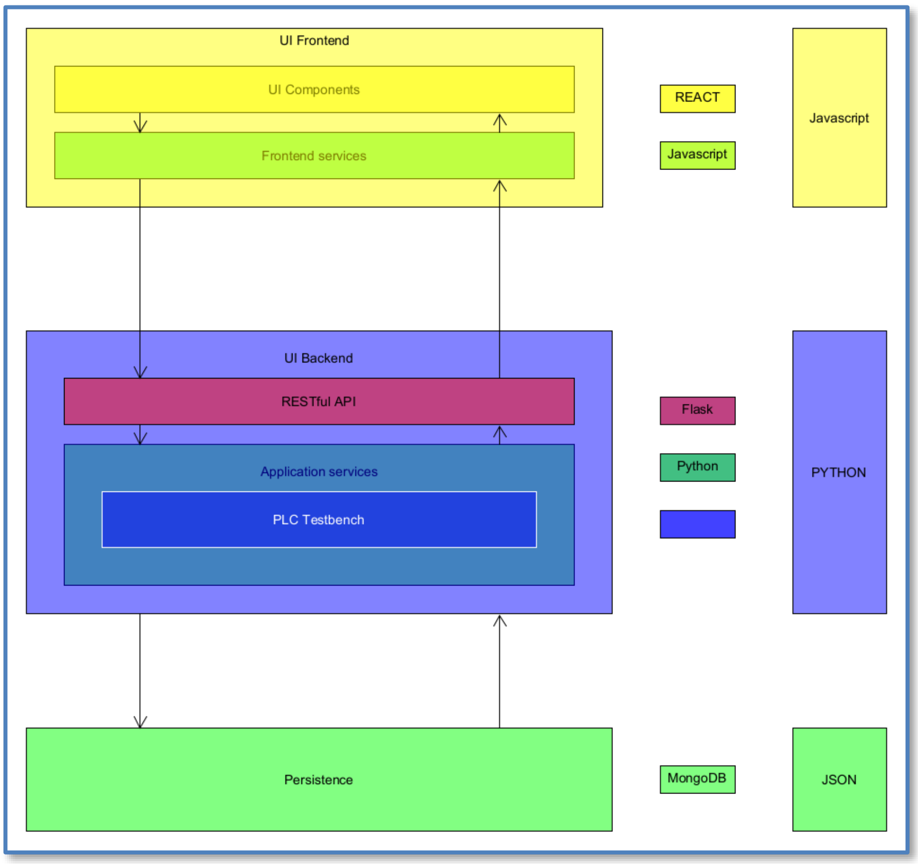


***Figure 25** – Application's layered architecture and layer mapping onto corresponding frameworks and languages*

2.2.3 Adopted programming languages

The criteria that drove the selection of the programming languages to be used in the project were the following in order of importance:

1) more portable language are to be preferred over less portable language;
2) more largely adopted and supported language are to be preferred over less widespread language;
3) the languages already used by the PLC Testbench are to be preferred in order to reduce the project maintenance complexity;
4) languages with better support for scalability are to be preferred over less scalable languages.

Based on this criteria JavaScript popped up as the best choice for the core UI layer because of its out-of-the-box integration with all browsers, its being the de-facto standard for the web and the huge number of GUI widgets libraries freely available. On the other side Python seemed like the most natural choice for the Rest API and services' layer, given its tighter integration with the PLC Testbench tool and the availability of several libraries and frameworks inherently designed for that purpose.

Adopting the same language used by the PLC Testbench tool helped a lot in keeping the project complexity as low as possible, both in terms of languages and in terms of final artifact's distribution.

2.2.4 Frameworks

Although frameworks in some cases tend to increase the complexity of software projects, in this case it was judged to be worth adopting some. In fact, leveraging available solutions to common problems was seen in this case not only as a way to avoid "reinventing the wheel" and to speed up the project delivery, but also as a means to standardize the code organization and incorporate some important best practices and design-patterns.

The frameworks used in the PLC Testbench UI codebase are the following:

Framework	Description
React	Probably the most important one; the library makes declarative, component-based development of GUIs much easier, by providing support for components' lifecycle management, state management, UI rendering management and optimization.
PrimeReact	A rich library of powerful react graphical widgets with a very nice look and feel which allowed fast and reliable implementation of the interface and supported easy refactoring and layout rearrangement.
Node.js	Used essentially to manage JavaScript dependencies through the npm package manager.
D3.js	Used for the highly specialized tasks related to visual representation of data; in most cases the library was used behind the scene by wrapping the D3.js code into brand new custom React components.
Waves.js	A flexible graphical library to represent layered audio waveforms by SVG (Scalable Vector Graphics); custom visualizations are supported and based on stacking specialized layers one on top of the other, like cursor layer, waveform layers, segment layers, datapoints layers.
Flask	A lightweight framework providing all the functionalities needed by a REST API based web application.
MongoDB	Persistence of the data was achieved by means of MongoDB, which is one of the most widespread opensource no-SQL databases. The choice of a no SQL database was mainly influenced by the fact that this technology does not impose a rigid schema on the data and therefore looked more flexible and adaptable in case of future extensions of the software
PyMongo	A library for Python programming language that makes interacting with MongoDB much more convenient by providing higher level abstractions over the data layer.
Wavesurfer.js	Library to display audio waveforms, supporting a rich set of plugins, one of which was exploited to implement spectrogram visualization for PLC Testbench UI.

One of the most complex parts of the graphical user interface both from design and implementation perspective is certainly the component displaying audio files' waveform. Despite the existence of several libraries accomplishing a similar function, none of them perfectly matched the specific needs of the project, as none of them could be natively integrated into React and each one did not cover all the identified use case scenarios out of the box.

In some cases, the mismatch was mainly concerning the visual functions, like not being able to present multiple graphs on the same timeline, in other cases it was more related to data handling.

Peaks.js for example is a very nice-looking framework developed at BBC, mainly intended to support audio files annotation, and editing, but in PLC testbench's context it presented some severe limitations in zoom management. In PLC analysis being able to span from full track to extremely detailed waveform visualization is an essential requirement, but the approach to zoom taken from the library is discrete, supporting only a set of precalculated zoom levels. With this regard the choice made in PLC Testbench UI project was dramatically different as it was decided to manage zoom in a continuous way rather than in a discrete one. In order to reach adequate performance several optimizations had to be applied like subsetting and downsampling techniques. This way the system can deal only with the portion of the data actually displayed at the minimal resolution actually needed.

Nevertheless, after a preliminary evaluation, one of the libraries seemed to be flexible enough to work as good starting point for the implementation of the needed set of React components. The Waves.js library, thanks to its approach based on composable layers, could support one of the main requirements for the waveform visualization components, which was displaying multiple overlapped waveforms and lost segments in the same view.

2.2.5 Packaging and distribution

With regard to packaging and distribution, the objective was a zero-effort installation of the tool and the widest possible portability in terms of supported platforms, and the proper technology to achieve this was considered to be Docker containers.

The PLC Testbench UI is available as a docker public image built from the master branch of the publicly available GitHub repository of the project. The only prerequisite to get the tool running in standalone mode is having a Docker installation on the target machine. This also means that the PLC Testbench UI can run on any platform where Docker can run, like all Linux distributions, Windows and MacOS.

In order to start the application two containers must be created, one to host a MongoDB instance and one to host the Flask web application. The template for command lines to launch each container are provided in the README.md file of the PLC Testbench UI's GitHub project and must be customized in the parts which are necessarily specific to each user's environment, like volumes' path or SSL certificates setup.

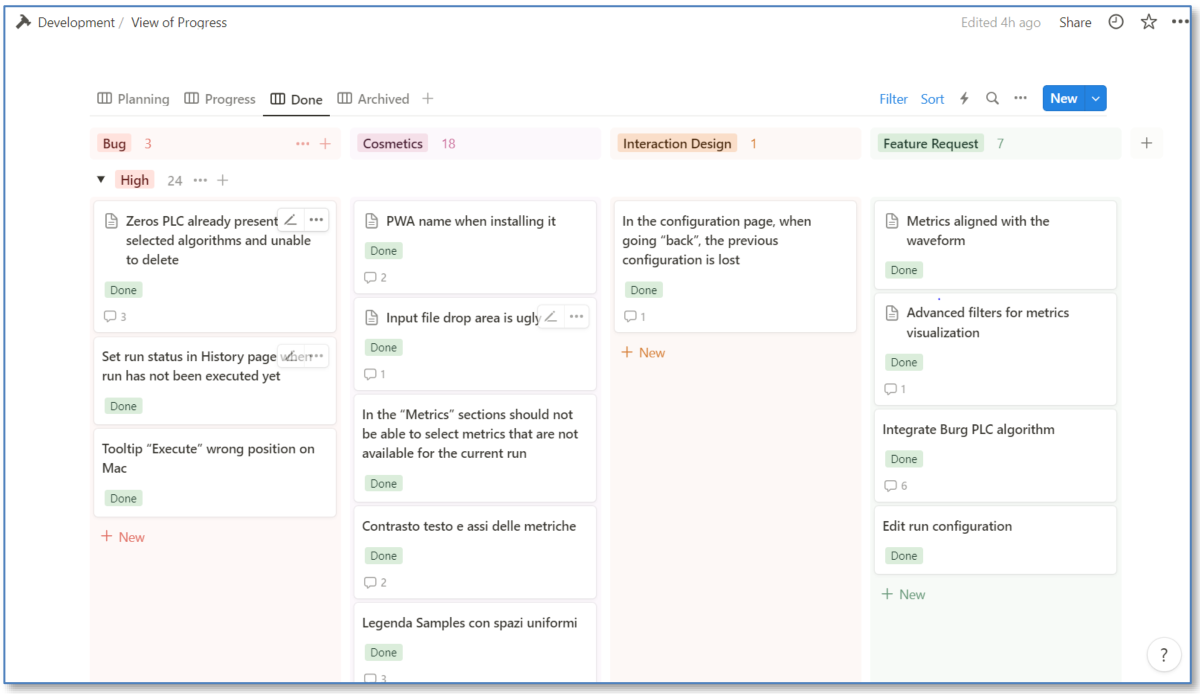
The Docker containerization, together with the packages installer tool from Python also helped in managing the huge number of transitive dependencies brought into the project by specialized libraries for audio manipulation like gstreamer, librosa or by others for more general purposes like tensorflow. It also addressed the complexity lying behind the installation of the libraries, which was often considerable.

2.2.6 Software Development Process

At the beginning of the thesis work my knowledge about the domain of digital audio signals processing was extremely limited and my knowledge on the specific subdomain of packet loss concealing was close to zero. Therefore, a major concern was keeping software delivery iterations as short as possible in order to avoid wasting time in developing useless features and in order to allow fast feedback on the implemented features.

A software development process based on Kanban methodology seemed to be a perfect match for this agile way of working because of its simplicity compared to other agile models like Scrum, which made it more appropriate and effective for a project managed by a single developer. Enforcing the strict role separation prescribed by Scrum or applying advanced Scrum techniques like velocity estimation and burn down charts would probably represent an overkill in this case.

The process was enforced through a combined use of Notion and GitHub to manage issues backlog. Notion proved to be an excellent tool to quickly capture any type of request about the application, be it a bug fixing request or a new requirement to be addressed or a new feature to be developed. On the other side GitHub proved to be an extremely powerful tool to track the correlation between the change requests and their implementation in the code base, combining at the same time simplicity, expressiveness, and tight integration with the source code management system.



Bug 3

High 24

Zeros PLC already present selected algorithms and unable to delete
Done
3

Set run status in History page when run has not been executed yet
Done

Tooltip "Execute" wrong position on Mac
Done

+ New

Cosmetics 18

PWA name when installing it
Done
2

Input file drop area is ugly
Done
1

In the "Metrics" sections should not be able to select metrics that are not available for the current run
Done

Contrasto testo e assi delle metriche
Done
2

Legenda Samples con spazi uniformi
Done
3

Interaction Design 1

In the configuration page, when going "back", the previous configuration is lost
Done
1

+ New

Feature Request 7

Metrics aligned with the waveform
Done

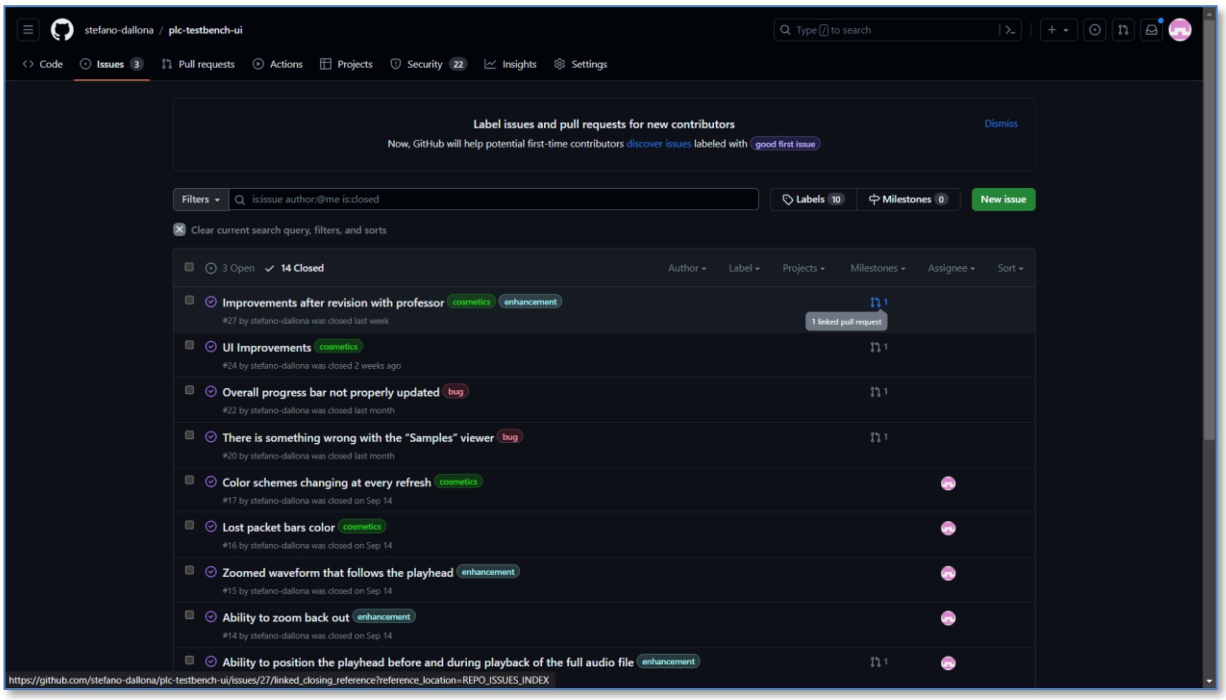
Advanced filters for metrics visualization
Done
1

Integrate Burg PLC algorithm
Done
6

Edit run configuration
Done

+ New

Figure 26 – Notion board organizing issues according to their priority and type



Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with good first issue

is:issue author:@me is:closed

Clear current search query, filters, and sorts

3 Open ✓ 14 Closed

Improvements after revision with professor cosmetics enhancement
#27 by stefano-dallona was closed last week

UI Improvements cosmetics
#24 by stefano-dallona was closed 2 weeks ago

Overall progress bar not properly updated bug
#22 by stefano-dallona was closed last month

There is something wrong with the "Samples" viewer bug
#20 by stefano-dallona was closed last month

Color schemes changing at every refresh cosmetics
#17 by stefano-dallona was closed on Sep 14

Lost packet bars color cosmetics
#16 by stefano-dallona was closed on Sep 14

Zoomed waveform that follows the playhead enhancement
#15 by stefano-dallona was closed on Sep 14

Ability to zoom back out enhancement
#14 by stefano-dallona was closed on Sep 14

Ability to position the playhead before and during playback of the full audio file enhancement

https://github.com/stefano-dallona/plc-testbench-ui/issues/27/linked_closing_reference?reference_location=REPO_ISSUES_INDEX

Figure 27 – GitHub's issue management

The current status of the GUI is the result of an iterative development process during which initial ideas where prototyped and reviewed in close collaboration with prof. Luca Turchet and Ing. Luca Vignati. Their precious feedback was crucial in recognizing new emerging use case, which led to the expansion of the application functionalities, and made it possible to gradually improve both the appearance and the ease of use of the application.

A thorough evaluation of the GUI usability was not carried out as it was far beyond the purpose of this thesis, given the considerable effort already implied by the software development activity and the bachelor thesis's nature of the present work, but it could represent an opportunity for a future dedicated work.

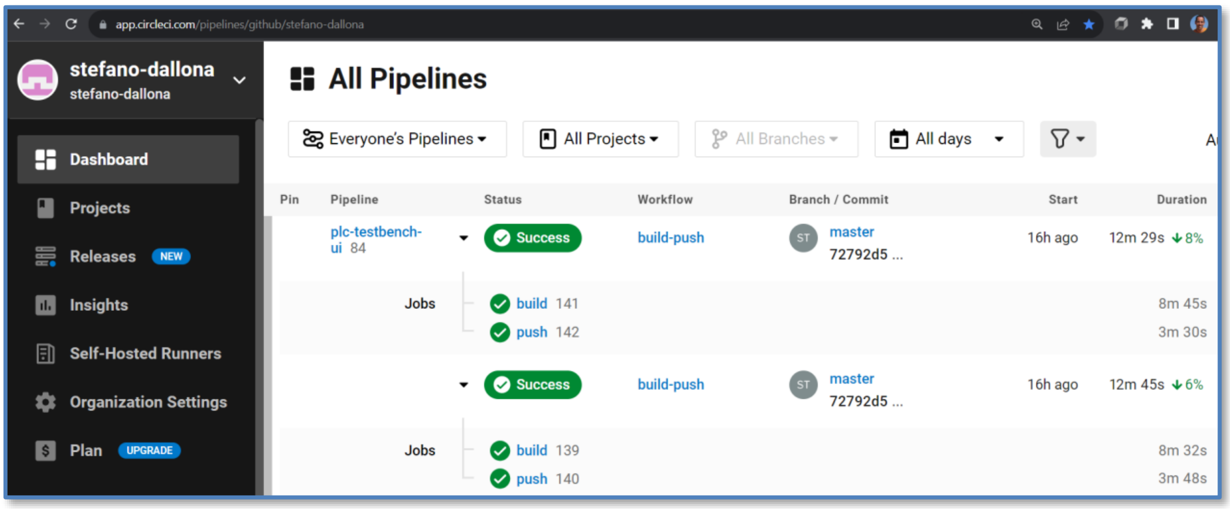
Nevertheless, a lot of attention was paid to making the interface as clear as possible by adopting a minimalistic design, standard conventions, and consistency across the entire application, making interaction flow as smooth as possible, providing the user with all and only the relevant information in the current context, detecting errors as early as possible and providing meaningful error description and instructions on how to solve or avoid the problem.

Some preliminary very positive feedback on the GUI has been received from a team of experts during a demo session carried out at the 4th International Symposium on the Internet of Sounds, which was held in Pisa on the 26th and 27th of October 2023 and where a paper on the PLC Testbench and on this thesis work was presented by Ing. Luca Vignati.

The need to release code frequently dictated by the agile development process adopted, also led to the decision to setup a Continuous Integration/Continuous Delivery infrastructure to automate the release of new features in the test environment. CI/CD pipelines are a very powerful tool as they allow describing the build and deploy process in the shape of declarative files that can be automatically parsed and executed by CI/CD platforms. The tools can also execute different test suites to address unit testing, regression testing, integration and load/stress testing before actually deploying the artifacts, thus ensuring a proper quality of the deployed releases. On top of all the usual advantages coming from automation, like increased reliability, higher standardization, increased traceability of the process, being simple text files, pipelines descriptors can be managed like any other source file in the project. This means they can be stored in the source code repository, thus getting the benefits of versioning and more reliable management in collaborative environments where multiple developers work on the same project. Automated deployment can also be automatically triggered when source code is committed to the repository, so that any code commit that is successfully built and tested can be immediately propagated to the test environment.

Circle CI cloud was selected as the CI/CD platform for the PLC Testbench UI project because it is very powerful and it is freely available for small teams and non-commercial use.

Having a CI/CD infrastructure in place helped a lot in shortening the release cycles' duration and in preventing deployment mistakes related to the differences between the local development environment located on a Windows laptop and the test environment running on a Mac OS server.



app.circleci.com/pipelines/github/stefano-dallona

stefano-dallona
stefano-dallona

- Dashboard
- Projects
- Releases NEW
- Insights
- Self-Hosted Runners
- Organization Settings
- Plan UPGRADE

All Pipelines

Everyone's Pipelines ▾ | All Projects ▾ | All Branches ▾ | All days ▾

Pin	Pipeline		Status	Workflow	Branch / Commit	Start	Duration
	plc-testbench-ui 84	▾	Success	build-push	ST master 72792d5 ...	16h ago	12m 29s ↓8%
	Jobs		✓ build 141				8m 45s
			✓ push 142				3m 30s
		▾	Success	build-push	ST master 72792d5 ...	16h ago	12m 45s ↓6%
	Jobs		✓ build 139				8m 32s
			✓ push 140				3m 48s

***Figure 28** – Circle CI pipeline automatically building and pushing a new Docker image to Docker Hub whenever any change is committed to the master branch of the application's GitHub repository.*

3 Results

The most important results achieved through the PLC Testbench UI development can be summarized as follows:

- Support to the validation and debugging of the PLC Testbench tool.
- Highlight of improvement areas for the PLC Testbench tool.
- Extension of the tool usability.
- Empowering of PLC Testbench tool's result analysis.
- Installation of the PLC Testbench made easier and more stable through containerization.

3.1 Support to the validation and debugging of the PLC-TestBench tool

During the development of the Graphical User Interface, it has become evident how the GUI could support and speed-up the debugging of the PLC algorithm, thus acting also as a validation tool for the PLC Testbench itself. The possibility to compare the waveforms of the original files with those of the reconstructed files at different level of details and focusing only on the regions affected by packets loss made it much easier to confirm the compliance of the PLC algorithms with some essential invariants.

One of these invariants is for example that signals must be identical to the original one in every region where packets have not been lost, except for some deviations in the area immediately following a packet loss where there can be some border effects caused by a gradual fading of the PLC reconstruction to make transition smoother.

The detailed visual representation of the signals also made it easier to discover any weird behavior in the PLC algorithms. Without a visual representation more sophisticated than the raw images generated natively by the PLC Testbench tool, finding out such situations would probably have been much more time consuming or even not possible at all.

3.2 Highlight of improvement areas for the PLC Testbench tool

Before the development of the GUI the PLC Testbench tool had only a minimal persistence layer built around filesystem storage for audio file output and ***.pickle*** files for configurations and packet loss masks. Pickle is a serialization format natively supported by python which allows to persist memory objects into a file and rebuild them when needed.

The inputs and outputs of each elaboration were grouped by means of filesystem directories and to avoid collisions in the folder names naming conventions had been adopted to encode the parameters. Naming conventions, together with the elaboration logs also represented the main criteria to find out the location where the outputs had been stored.

Every elaboration was unrelated from the others and the inquiry functions were completely unsupported.

While developing the GUI, the "Run" concept slightly emerged as a container for the metadata related to an elaboration and together with it the need to move to a more sophisticated persistence layer became clear.

At the beginning a raw set of inquiry functions was built on top of the existing persistence but they were extremely limited due to the poor information available about the "Run". Finally, the decision was made to exploit a database to collect the relevant metadata of the elaborations.

Database-based persistence paved the way for the development of a much more powerful inquiry module, exploiting a query engine to dynamically build queries. It also led to further insights, like the possibility to store into persistence the hashes of the files and configuration parameters used by each elaboration and to use them to enable performance optimization in the PLC Testbench by completely removing redundant processing across different elaboration. Thanks to the introduction of the hashing feature, it was also possible to build a validation layer to detect and prevent data inconsistencies.

3.3 Extension of the tool usage

The availability of a GUI for the Testbench Tool improves the overall user experience, makes the learning curve of the tool less steep and extends its usability to users not having a programming background.

3.4 Empowering of PLC Testbench tool's result analysis

Compared to the out-of-the-box representations provided by the PLC Testbench tool which are essentially static images, the PLC Testbench UI allows real-time exploration of the data at different level of detail. The detail level can be selected interactively, depending on the needs and the data are more integrated by being displayed, when comparable, as multiple time-series on the same graph.

In order to improve the user experience the GUI has been made responsive so that it can automatically adjust to different devices (PC, tablet, mobile).

The interface has been designed to be user-friendly, intuitive, and as fast as possible, considering the large dataset inherently involved in audio processing applications and the overhead related to charting functions. Whenever possible subsampling or caching techniques have been applied to minimize latency and network bandwidth waste.

3.5 Comparison between native PLC Testbench's I/O functions and the Web GUI

Below a visual comparison is performed between the input/output formats consumed/produced out-of-the-box by the PLC Testbench and the corresponding Web GUI functions. The comparison is structured by functionality.

Elaboration configuration	
PLC Testbench native functions	Web GUI

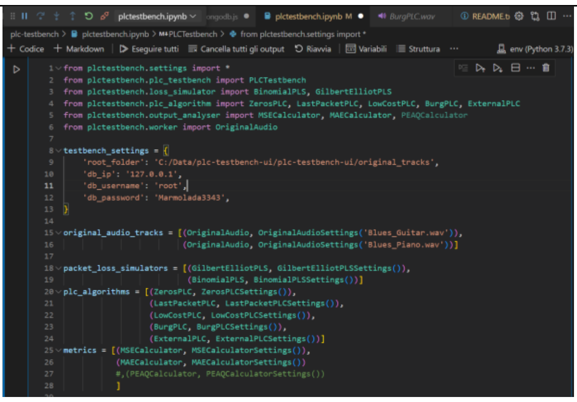
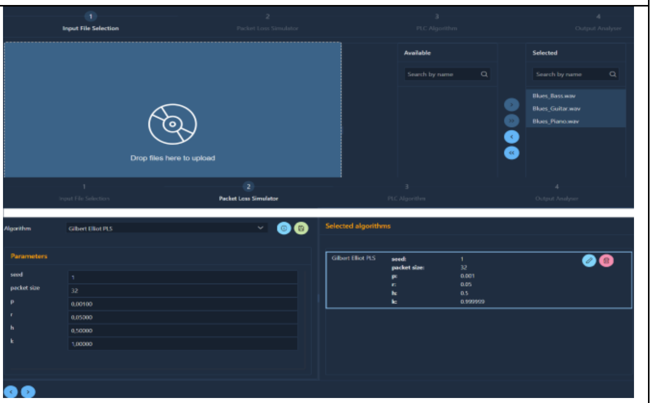

	

Figure 29 *– The configuration must be edited directly in the Integrated Development Environment (IDE), by changing the list of the objects passed as an input to the main PLC Testbench class instance. No validation is performed before the application is actually run.*	***Figure 30*** *– The configuration is performed in a visual way, going through a series of guided steps and getting immediate feedback on the its validity. In the upper screen input files can easily be uploaded and selected. In the lower screen the settings for a Gilbert-Elliot packet loss simulator instance are configured.*

```
from plctestbench.settings import *
from plctestbench.plc_testbench import PLCTestbench
from plctestbench.loss_simulator import BinomialPLS, GilbertElliotPLS
from plctestbench.plc_algorithm import ZerosPLC, LastPacketPLC, LowCostPLC, BurgPLC, ExternalPLC
from plctestbench.output_analyser import MSECalculator, MAECalculator, PEAQCalculator
from plctestbench.worker import OriginalAudio

testbench_settings = {
    'root_folder': 'C:/Data/plc-testbench-ui/plc-testbench-ui/original_tracks',
    'db_ip': '127.0.0.1',
    'db_username': 'root',
    'db_password': 'Marmolada3343',
}

original_audio_tracks = [(OriginalAudio, OriginalAudioSettings('Blues_Guitar.wav')),
                         (OriginalAudio, OriginalAudioSettings('Blues_Piano.wav'))]

packet_loss_simulators = [(GilbertElliotPLS, GilbertElliotPLSSettings()),
                          (BinomialPLS, BinomialPLSSettings())]
plc_algorithms = [(ZerosPLC, ZerosPLCSettings()),
                  (LastPacketPLC, LastPacketPLCSettings()),
                  (LowCostPLC, LowCostPLCSettings()),
                  (BurgPLC, BurgPLCSettings()),
                  (ExternalPLC, ExternalPLCSettings())]
metrics = [(MSECalculator, MSECalculatorSettings()),
           (MAECalculator, MAECalculatorSettings())
           #,(PEAQCalculator, PEAQCalculatorSettings())
           ]
```

Blues_Bass.wav
Blues_Guitar.wav
Blues_Piano.wav

Algorithm	Gilbert Elliot PLS

Parameters

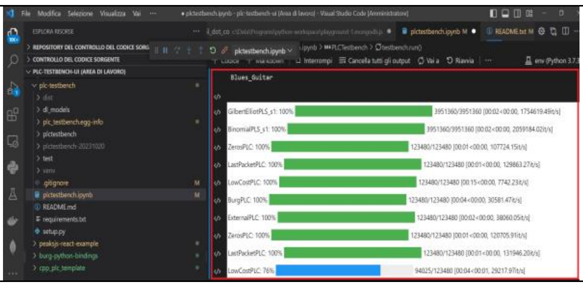
seed	1
packet size	32
p	0,00100
r	0,05000
h	0,50000
k	1,00000

Selected algorithms

Gilbert Elliot PLS	seed:	1
	packet size:	32
	p:	0.001
	r:	0.05
	h:	0.5
	k:	0.999999

Progress monitoring

PLC Testbench native functions



```
Blues_Guitar

GilbertElliotPLS_s1: 100% 3951360/3951360 [00:02<00:00, 1754619.49it/s]
BinomialPLS_s1: 100% 3951360/3951360 [00:02<00:00, 2059184.02it/s]
ZerosPLC: 100% 123480/123480 [00:01<00:00, 107724.15it/s]
LastPacketPLC: 100% 123480/123480 [00:01<00:00, 129863.27it/s]
LowCostPLC: 100% 123480/123480 [00:15<00:00, 7742.23it/s]
BurgPLC: 100% 123480/123480 [00:04<00:00, 30581.47it/s]
ExternalPLC: 100% 123480/123480 [00:02<00:00, 38060.05it/s]
ZerosPLC: 100% 123480/123480 [00:01<00:00, 120705.91it/s]
LastPacketPLC: 100% 123480/123480 [00:01<00:00, 131946.20it/s]
LowCostPLC: 76% 94025/123480 [00:04<00:01, 29217.97it/s]
```

Figure 31 – The native progress monitor does not show the hierarchical nature of the elaboration and the only information available for each step are the progress and the time to complete

Web GUI

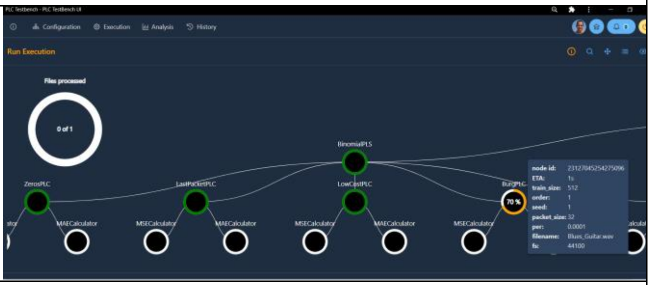


Figure 32 – The Web UI shows both the overall progress (on the left side) and the progress within the current file elaboration (on the right side). The progress information is presented in a hierarchical fashion and enriched with the specific parameters of each node (see the tooltip in the right-bottom corner of the picture).

Results analysis

PLC Testbench native functions

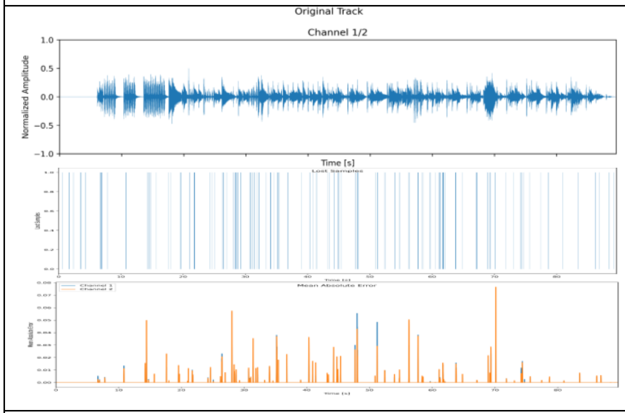


Figure 33 – Visual plots are saved as bitmap images and the entire track is represented, thus fixing resolution at creation time and making overlapping multiple waveforms unmanageable. No sample level view is available.

Web GUI

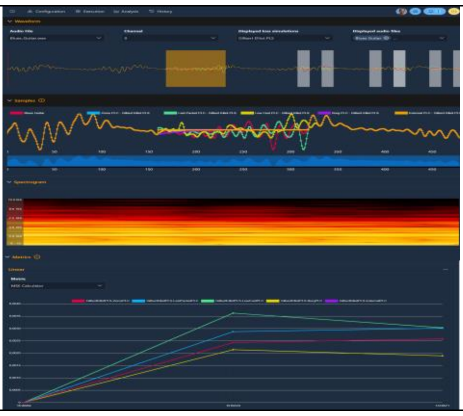


Figure 34 – In the Web UI the same information is displayed but in a more integrated and comparable way. Packet loss regions and waveforms are overlapped, a sample level zoom function and a spectrogram are available. All the views allow the selection of the dataset to be displayed and share the same zoom capability. Finally, they are scroll together with waveforms when playback is activated in zoomed mode.

Audio files playback

PLC Testbench native functions

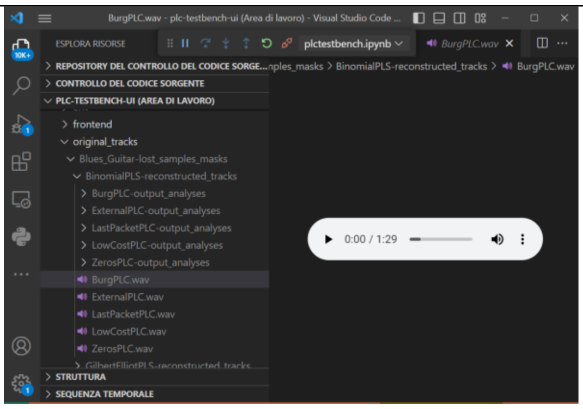

BurgPLC.wav - plc-testbench-ui (Area di lavoro) - Visual Studio Code
ESPLORA RISORSE
REPOSITORY DEL CONTROLLO DEL CODICE SORGENTE
CONTROLLO DEL CODICE SORGENTE
PLC-TESTBENCH-UI (AREA DI LAVORO)
frontend
original_tracks
Blues_Guitar-lost_samples_masks
BinomialPLS-reconstructed_tracks
BurgPLC-output_analyses
ExternalPLC-output_analyses
LastPacketPLC-output_analyses
LowCostPLC-output_analyses
ZerosPLC-output_analyses
BurgPLC.wav
ExternalPLC.wav
LastPacketPLC.wav
LowCostPLC.wav
ZerosPLC.wav
GilbertElliotPLS-reconstructed_tracks
STRUTTURA
SEQUENZA TEMPORALE
plctestbench.ipynb
BurgPLC.wav
0:00 / 1:29

Figure 35 – In order to play a specific track's version, the user has to navigate across the output folder of the application in the Integrated Development Environment (IDE, Visual Studio Code

Web GUI

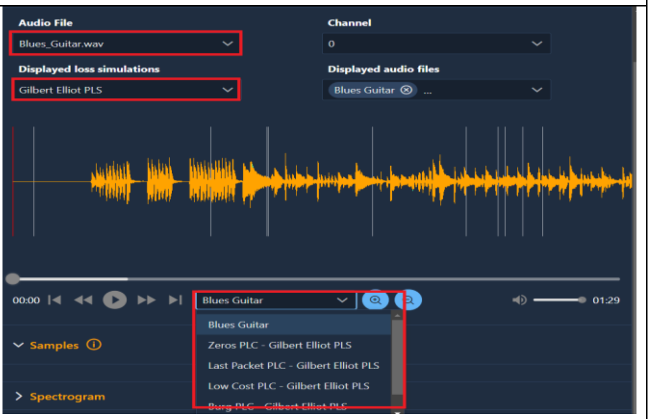

Audio File
Blues_Guitar.wav
Channel
0
Displayed loss simulations
Gilbert Elliot PLS
Displayed audio files
Blues Guitar
00:00
01:29
Blues Guitar
Zeros PLC - Gilbert Elliot PLS
Last Packet PLC - Gilbert Elliot PLS
Low Cost PLC - Gilbert Elliot PLS
Burg PLC - Gilbert Elliot PLS
Samples
Spectrogram

Figure 36 – The user can easily select what track and what version of the track to play.

in this case) and find the proper audio file by inspecting the parent folders' and files' names.	

Search	
PLC Testbench native functions	Web GUI

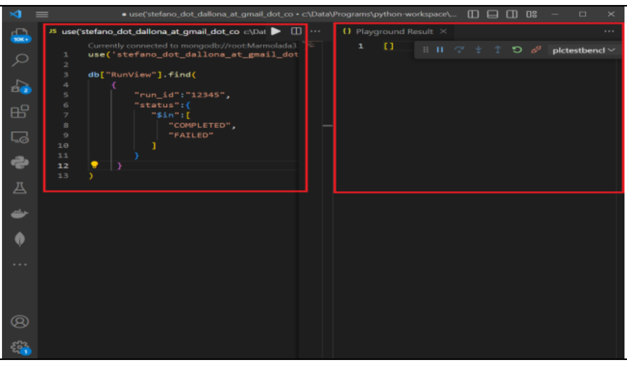
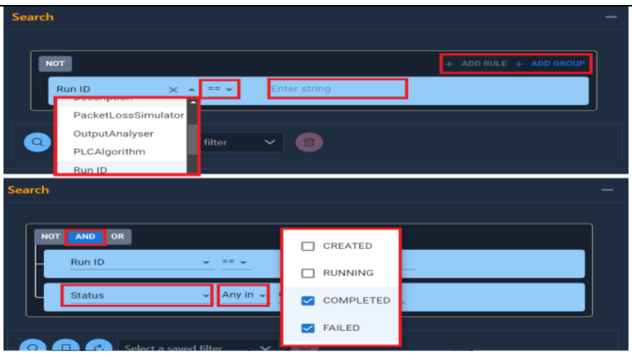

	

***Figure 37** – Only available through MongoDB clients or Visual Studio Mongo extension, by directly executing a handwritten query. The output is in JSON format.*	***Figure 38** – Queries can easily be composed through a builder by adding groups of conditions connected by logical operators.*

```
use('stefano_dot_dallona_at_gmail_dot

db["RunView"].find(
    {
        "run_id":"12345",
        "status":{
            "$in":[
                "COMPLETED",
                "FAILED"
            ]
        }
    }
)
```

```
[]
```

3.6 Future work

Despite providing an implementation for all the basic recognized use cases and being fully operational, the PLC Testbench UI has obviously still a lot of room for improvements. Future enhancements of the GUI could be dictated by the evolution of the PLC Testbench itself or could be more strictly connected to the user interaction.

The following list mentions a few features that could be developed in future releases of the application:

- extensive evaluation with users;
- extending support to additional audio file formats, like mp3, ogg, flac, etc.;
- providing a function to expose the PLC Testbench logs;
- implementing search functions on output results;
- implement different user interface themes;
- internationalize the UI by adding support for additional languages;
- implementing user collaboration use cases;
- extending supported identity providers for login (currently only Google);
- developing export functions for the most widely used formats (Excel, CSV, etc.).

4 Conclusions

PLC Testbench provided natively some basic visual outputs, but they were not sufficient to perform an effective analysis of the results produced by the different PLC algorithms. Also, the configuration phase was complex due to the need to install a fully working development environment to be able to modify the parametrization in the Jupiter notebook source code and run it.

Distribution was made difficult by the high number and complex installation of the dependencies involved in the project setup.

The main deliverable of this thesis is a Web GUI for the PLC Testbench that addresses all the mentioned pain points by:

- providing a set of interactive functions to analyze and compare the result of the different PLC algorithms at any level of detail;
- guiding the user through the configuration of each PLC Testbench module;
- making Docker the only prerequisite to run the application;
- providing a containerized bundle with all the needed dependencies.

On top of this, the development of the Web GUI based on a multi-layer architecture led to these additional benefits:

- possibility to deploy the application in a highly scalable environment;
- possibility to expose the PLC Testbench functions as services via a RESTful API;
- thorough validation/debugging of the PLC Testbench.

Bibliography

[1] “File:CPT-sound-physical-manifestation.svg,” [Online]. Available: https://commons.wikimedia.org/wiki/File:CPT-sound-physical-manifestation.svg, licensed under the Creative Commons CC0 1.0 Universal Public Domain Dedication (https://creativecommons.org/publicdomain/zero/1.0/deed.en). [Accessed 2023].

[2] “File:Signal Sampling.svg,” [Online]. Available: https://commons.wikimedia.org/wiki/File:Signal_Sampling.svg, licensed under the Creative Commons CC0 1.0 Universal Public Domain Dedication (https://creativecommons.org/publicdomain/zero/1.0/deed.en). [Accessed 2023].

[3] G. Haßlinger and O. Hohlfeld, “The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet.,” 2008, pp. 269-286.

[4] M. Sacchetto, “Implementations of various autoregressive,” 2023. [Online]. Available: https://github.com/.

[5] “Perceptual Evaluation of Audio Quality,” [Online]. Available: https://en.wikipedia.org/wiki/Perceptual_Evaluation_of_Audio_Quality. [Accessed 2023].

[6] J. Yablonski, “Design Principles for Reducing Cognitive Load,” Marvel, [Online]. Available: https://marvelapp.com/blog/design-principles-reducing-cognitive-load/. [Accessed 2023].

[7] C. S. Oh, J. N. Bailenson and G. F. Welch, “A systematic review of social presence: Definition, antecedents, and implications,” *Frontiers in Robotics and AI,* p. 114, 2018.

[8] C. Rottondi, C. Chafe, C. Allocchio and A. Sarti, “An overview on networked music performance technologies,” *IEEE Access,* vol. 4, p. 8823–8843, 2016.

[9] L. Gabrielli and S. Squartini, Wireless Networked Music Performance, Springer, 2016.

[10] L. Turchet and C. Fischione, “Elk Audio OS: an open source operating system for the Internet of Musical Things,” *ACM Transactions on the Internet of Things,* vol. 2, no. 2, p. 1–18.

[11] C. Drioli, C. Allocchio and N. Buso, “Networked performances and natural interaction via lola: Low latency high quality a/v streaming system,” in *International Conference on Information Technologies for Performing Arts, Media Access, and Entertainment,* Springer, 2013, p. 240–250.

[12] C. A. G. D. Silva and C. M. Pedroso, “Mac-layer packet loss models for wi-fi networks: A survey,” *IEEE Access,* vol. 7, p. 180 512–180 531, 2019.

[13] H. Sanneck, A. Stenger, K. B. Younes and B. Girod, “A new technique for audio packet loss concealment,” in *Proceedings of GLOBECOM’96. 1996 IEEE Global Telecommunications Conference. IEEE, 1996, pp. 48–52.*

[14] E. O. Elliott, “Estimates of error rates for codes on burst-noise channels,” *The Bell System Technical Journal,* vol. 42, no. 5, p. 1977–199742, 1963.

[15] M. Holters, “Gstpeaq - a gstreamer plugin for perceptual evaluation of audio quality (peaq),” 2015. [Online]. Available: https://github.com/HSU-ANT/gstpeaq.

[16] M. Li, M. Wu, D. Wu, L. Wang and C. Xu, “Packet loss concealment using enhanced waveform similarity overlap-and-add technique with management of gains,” in *2009 5th International Conference on Wireless,* 2009, pp. 1–4.

[17] T. Thiede, W. C. Treurniet, R. Bitto, C. Schmidmer, T. Sporer, J. G. Beerends and C. Colomes, “Peaq-the itu standard for objective measurement of perceived audio quality,” *Journal of the Audio Engineering Society,* vol. 48, no. 1/2, p. 3–29, 2000.

[18] S. Pascual, J. Serr`a and J. Pons, “Adversarial auto-encoding for packet loss concealment,” in *2021 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA). IEEE,* 2021, pp. 71–75.

[19] Q. Ji, C. Bao and Z. Cui, “Packet loss concealment based on phase correction and deep neural network,” *Applied Sciences,* vol. 12, no. 19, p. 9721, 2022.

[20] R. Lotfidereshgi and P. Gournay, "Speech prediction using an adaptive recurrent neural network with application to packet loss concealment," in *2018 IEEE International Conference on Acoustics, Speech*

and Signal, 2018, pp. 5394–5398.

[21] P. Verma, A. Mezza, C. Chafe and C. Rottondi, “A deep learning approach for low-latency packet loss concealment of audio signals in networked music performance applications,” in *2020 27th Conference of Open Innovations Association (FRUCT). IEEE*, 2020, pp. 268–275.

[22] M. Sacchetto, Y. Huang, A. Bianco and C. Rottondi, “Using autoregressive models for real-time packet loss concealment in networked music performance applications,” in *Proceedings of the International Conference Audio Mostly*, 2022, pp. 203–210.

[23] A. W. Rix, J. G. Beerends, M. P. Hollier and A. P. Hekstra, “Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs,” in *2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221), vol. 2. IEEE*, 2001, pp. 749–752.

[24] F. Vassilatos and C. Crawshaw, “Dashboard Design UX Patterns,” [Online]. Available: https://pencilandpaper.io/articles/ux-pattern-analysis-data-dashboards/. [Accessed 2023].

[25] A. F. Khalifeh, A.-K. Al-Tamimi and K. A. Darabkh, “Perceptual evaluation of audio quality under lossy networks,” in *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). IEEE*, 2017, pp. 939–943.

[26] A. Carôt, C. Hoene, H. Busse and C. Kuhr, “Results of the fast-music project—five contributions to the domain of distributed music,” *IEEE Access*, vol. 8, p. 47 925–47 951, 2020.

[27] M. Fink and U. Zölzer, “Low-delay error concealment with low computational,” in *Proceedings of the International Conference on Digital Audio Effects*, 2014, pp. 309–316.

[28] M. Fink, “Enhancements for networked music performances,” *Ph.D. dissertation, Universitätsbibliothek der HSU/UniBwH*, 2018.

[29] L. Vignati, S. Dallona and L. Turchet, “PLC Testbench: a modular tool for the study and comparison of audio Packet Loss Concealment algorithms,” in *Proceedings of the 4th International Symposium on the Internet of Sounds*, 2023, (In Press).

Attachment A Source code repository

PLC Testbench UI’s GitHub source code repository

The source code of the project is available on the following GitHub public repository:
https://github.com/cimil/plc-testbench

Attachment B Docker image

PLC Testbench UI’s Docker image

A pre-built image of the application is publicly available at the following URL:
https://hub.docker.com/r/cimil/plc-testbench-ui

Running pre-built image and building custom images from source code

If the pre-built image does not fit your needs you can build a suitable image by customizing the Dockerfile located in the root directory of the source code project on published on GitHub.

Attachment C CI/CD Pipelines

In order to make release cycles faster and more reliable a Continuous Integration/Continuous Deployment pipeline has been setup on the free cloud-based tool Circle CI.

The pipeline has been developed exploiting the pipeline as code feature of Circle CI, based on yaml language and is stored in the ***.circleci/config.yml*** file in the GitHub source code repository of the PLC Testbench UI project.

CircleCI CI/CD Pipelines

```yaml
# Use the latest 2.1 version of CircleCI pipeline process engine.
# See: https://circleci.com/docs/configuration-reference
version: 2.1

# Define a common Docker container and environment for jobs
executors:
  docker-publisher:
    # Define the image tag
    environment:
      IMAGE_TAG: stdallona/plc-testbench-ui:1.0.3
    # Use `docker:stable` as the Docker image for this executor
    docker:
      - image: docker:stable

# Define a job to be invoked later in a workflow.
# See: https://circleci.com/docs/configuration-reference/#jobs
jobs:
  build:
    # Use docker-publisher from above as the Docker container to run this job in
    executor: docker-publisher

    # Add steps to the job
    # See: https://circleci.com/docs/configuration-reference/#steps
    steps:
      - checkout

      # Set up a separate Docker environment to run `docker` commands in
      - setup_remote_docker

      - run:
          name: Build Docker image
          command: docker build --tag "${IMAGE_TAG}" .

      # Archive and persist the Docker image
      - run:
          name: Archive Docker image
          command: docker save --output image.tar "${IMAGE_TAG}"
      - persist_to_workspace:
          root: .
          paths:
              - ./image.tar

    push:
      # Use docker-publisher from above as the Docker container to run this job in
      executor: docker-publisher

      steps:
        # Set up a separate Docker environment to run `docker` commands in
        - setup_remote_docker

        # Load and un-archive the Docker image
        - attach_workspace:
            at: /tmp/workspace
        - run:
            name: Load Docker image
            command: docker load --input /tmp/workspace/image.tar

        # Log in to Docker Hub and push the image
        - run:
            name: Publish Docker image
            command: |
                        echo   "${DOCKERHUB_PASS}"   |   docker   login   --username
"${DOCKERHUB_USERNAME}" --password-stdin
              docker push "${IMAGE_TAG}"

  # Run the two different jobs as a sequenced workflow
  workflows:
    version: 2
    build-push:
      jobs:
        # Run the build first
        - build
        # Push the image second
        - push:
            # Build needs to finish first
            requires:
              - build
            # Only push images from the master branch
            filters:
              branches:
                only: master
```

Figure 39 *– Yaml descriptor of the Circle CI pipeline used to automatically build and deploy the project.*