

OpenPLC Studio: a scalable application for developing and evaluating Audio Packet Loss Concealment Algorithms

Filippo Daniotti and Luca Turchet

*Department of Information Engineering and Computer Science
University of Trento*

filippo.daniotti@unitn.it, luca.turchet@unitn.it

Abstract—Audio streaming applications over IP require low end-to-end latency and are thus subject to buffer under-run at the receiver side. When this happens, Packet Loss Concealment (PLC) algorithms are employed to reconstruct the empty buffer. Developing and evaluating PLC algorithms for audio signals (including speech and music) requires many operations, including simulating packet loss traces, reconstructing the audio, and computing evaluation metrics. This multi-stage process can be difficult to standardize and reproduce across experiments. To date, the only publicly available attempt to streamline this workflow is the PLC Testbench tool. While it presented the value of a modular tool for PLC development, it was developed to support a small-scale use and lacked important features for advanced PLC algorithms, such as crossfade support. To solve this, in this work we present OpenPLC Studio, a complete redesign of PLC Testbench, addressing its limitations and providing the community with a flexible, open-source research toolkit. The proposed system combines a modular Python core package and a scalable web application, featuring flexible and reproducible pipelines, scalable batch processing, and a plug-in system for easier integration of custom PLC algorithms and evaluation modules. The system bridges the gap between rapid prototyping and large-scale experimentation, providing a flexible and reliable application for the development, evaluation, and comparison of PLC algorithms for real-time audio applications, thereby fostering further advances in the field.

Index Terms—Networked Music Performance, Packet Loss Concealment, Internet of Sounds

I. INTRODUCTION

Real-time audio streaming over IP has become ubiquitous in modern applications involving speech and music and continues to advance rapidly in both performance and reliability. Among the various challenges that must be addressed, resilience to packet loss is particularly important; this is typically handled by error recovery schemes, known in the audio domain as packet loss concealment (PLC) techniques. PLC is a topic of growing interest among the scientific and industrial research

community, which prompted the realization of challenges targeting PLC algorithms for both speech [1], [2] and music [3], [4] signals. PLC is a non-trivial problem to address, as it involves a number of operations which are typically hard to automate. An audio file must first be processed to simulate packet loss, either using a realistic model or a simplified scenario, such as constant-interval drops. The degraded audio is then reconstructed according to the loss trace. Finally, the result is either evaluated by a human expert or analyzed using objective metrics. Despite the extensive literature on PLC algorithms, there has been, for a long time, the lack of a framework that standardizes their evaluation workflow. In practice, researchers rely on ad-hoc scripts to generate loss traces, run algorithms, and compute metrics on the reconstructed tracks, which makes experiments difficult to reproduce and results hard to compare across studies.

To the best of our knowledge, a first attempt to solve this problem was made with the PLC Testbench 1.0 (PLCTv1) application developed in 2023 and presented in [5]. It provided an open-source tool for PLC algorithm development and evaluation, combining loss trace generation, algorithm execution, and metric computation in a single pipeline abstracted as a computational tree. While it demonstrated the value of an integrated tool for the PLC development and evaluation pipeline, it was designed primarily for small-scale, interactive use: the computational tree implementation and DSP logic were tightly coupled to the GUI, and extensibility required modifying the codebase. Moreover, it lacked support for crossfading between PLC prediction buffer and incoming valid buffer, which is a standard feature in real-world deployments of PLC algorithms to ensure continuity at packet boundaries and minimize playback artifacts. Additionally, it included a limited set of built-in algorithms, which no longer reflect the current state of the art as of today. These limitations are particularly constraining given that, in the absence of alternative tools, the PLCTv1 effectively acts as the de facto reference tool for experimental PLC research.

In this work, we present a complete re-implementation of PLCTv1, with the goal of turning what is currently the only publicly available PLC development and evaluation tool

This work was supported by the MUSMET project funded by the EIC Pathfinder Open scheme of the European Union (grant agreement n. 101184379). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Innovation Council. Neither the European Union nor the European Innovation Council can be held responsible for them.

into a robust research toolkit for the community. To address its limitations, we re-engineered the application architecture using a domain-centered approach: we separated a core toolkit, *PLC Testbench (PTB)*, from *OpenPLC Studio (OPS)*, a GUI application that acts as a wrapper web application on top of PTB. PTB is distributed as a standard Python package with full backwards compatibility with PLCTv1 and provides a full-fledged programmatic API, primarily intended for use within a Jupyter Notebook environment. We expanded the DSP features, most notably by including support for crossfade, and we implemented recent state-of-the-art PLC algorithms and evaluation metrics. As a GUI, OPS has been rebuilt from scratch, building upon the GUI of PLCTv1, and is distributed as a stack of containerized applications. To better suit the deployment scenario of the server of a small lab, it is developed as a full-stack application comprising a web UI and a server backend wrapping PTB. Additionally, the OPS includes a plugin system that enables users to create custom modules without modifying the application.

The core contributions of this paper are as follows:

- A complete re-engineering of PLCTv1 into two decoupled software components: PTB, which encapsulates the computational graph and DSP logic, and OPS, which handles the user interface, persistence, visualization, and job management, as well as a plugin-based architecture to facilitate the development and integration of new PLC algorithms;
- Enhanced tooling within PTB. While preserving the original programming model of PLCTv1, the API has been refined to improve flexibility and extensibility. In particular, explicit support for crossfading in PLC algorithms has been introduced, enabling more realistic and reliable performance evaluation;
- New built-in modules in PTB, aligning the tool with recent advancements in the state of the art and covering both music-oriented and speech-oriented PLC approaches.

The remainder of this paper is organized as follows. Section II surveys the relevant literature and provides the necessary background on PLC algorithms. Section III reviews the PLCTv1 application and highlights its main limitations. Section IV presents the design rationale underlying the development of PTB from PLCTv1, along with its key features and improvements. Section V describes the architectural choices and design of OPS. Finally, Section VI provides a discussion, concludes the paper and outlines directions for future work.

II. RELATED WORK

A. Packet Loss Concealment

When dealing with streaming audio content on a packet-switched network, we typically refer to error recovery algorithms as PLC algorithms. Many of them have been proposed over the years, mostly developed for and tested on speech audio. Some of them are based on the simple idea of performing interpolation by finding pitch patterns on the buffer of audio

packets surrounding the loss, in a class that is often referred to as waveform substitution [6]–[9]. By contrast, other methods are model-based, where a model is fitted on a context window of past packets and used to perform a prediction for the current missing packet. The model is typically autoregressive [10], [11], although other types of models have been proposed [12]. In recent years, many algorithms based on deep learning models have been proposed [13], [14], and the growing interest in this approach is reflected in community-wide initiatives such as the *INTERSPEECH 2022 Audio Deep PLC Challenge* [1] and the *ICASSP 2024 Audio Deep PLC Grand Challenge* [2].

When it comes to PLC algorithms tailored to musical audio signals, there is significantly less research. As in the case of speech audio signals, naïve PLC algorithms such as zero-filling and last-packet repetition can be employed [6]. Authors in [15] presented a heuristic-based waveform substitution approach that exploits additional audio features such as zero-crossing analysis to reconstruct missing audio. Recently, scientists have tried to use deep learning-based methods for PLC. A first attempt went for a purely deep learning-based approach and proposed a CNN architecture extracting a latent representation of the spectrogram of a past buffer and combined it with the last packet in the time domain in a downstream fully-connected neural network to get concealed packets [16]. The study reported in [3] presented PARCnet, a hybrid model featuring an autoregressive branch and a parallel neural network to predict the residual of the autoregressive model. The research community’s interest in music PLC has recently increased, culminating in two editions of the IEEE-IS² Music PLC Challenge, held in 2024 [17] and 2025 [4], which attracted numerous novel methods [18]–[22].

B. Evaluation of PLC algorithms

Simple metrics such as mean absolute error (MAE) or mean squared error (MSE) can be used to evaluate PLC algorithms by comparing the reconstructed audio against the reference signal. However, these measures often exhibit poor correlation with perceived audio quality. The primary objective of PLC is not to achieve a sample-accurate reconstruction of lost frames, but rather to render losses imperceptible or to minimize their impact on the listening experience. To this end, perceptually motivated metrics are commonly employed. In particular, perceptual evaluation of audio quality (PEAQ), originally developed for broadband audio coding [23], is often used, alongside related measures such as perceptual evaluation of speech quality (PESQ) [24], which is widely adopted in speech coding. Nevertheless, their effectiveness in the context of PLC remains an open question [25].

To date, the most accurate procedure for assessing the performance of a PLC algorithm is to perform a user study involving listening tests with human subjects; examples of standardized tests are MOS [26], ITU-T P.804 [27] and MUSHRA [28]. Despite a few attempts to streamline the realization of such tests [29], they remain expensive, time-consuming, hard to scale, and intrinsically error-prone. This has led to increasing interest in the development of evaluation

metrics specifically tailored to PLC, including data-driven approaches for speech, such as PLCMOS [30], and model-based approaches for music [31].

III. LEGACY PLC TESTBENCH 1.0

PLC Testbench 1.0 (PLCTv1) was distributed as a single Docker image, packaging a Python component—responsible for core algorithms implementation, execution, and data persistence—within a web application that enabled users to configure PLC experiments and visualize their results.

PLCTv1 defined four types of Modules: AudioTrack (wrapping an audio file), PacketLossSimulators (representing algorithms generating packet loss traces), PLCAAlgorithm (representing PLC algorithms), and OutputAnalyser (representing metrics). The user specified a number greater than or equal to one for each Module type. For each AudioTrack, a separate computational tree was then generated. Each tree had depth 4, and its size was $n_{\text{simulators}} \times n_{\text{algorithms}} \times n_{\text{metrics}}$: the root corresponded to the AudioTrack, while the OutputAnalysers formed the leaves. The actual algorithms were implemented as subclasses of the base class of the corresponding Module, primarily by overriding the `run()` method, which defines the behavior of the Module when instantiated as a node in the computational tree and run during an execution thereof. Refer to [5] for a complete list of the implemented modules.

Additionally, PLCTv1 was responsible for the persistence layer: nodes of the computational tree were stored in a MongoDB database, which was required for the application to function. While caching provided performance benefits, this design introduced undesirable coupling between system components, with the computational tree lifecycle and data persistence living within the same software component.

The GUI application was unreliable and difficult to maintain, with frequent failures and poor performance. The codebase was poorly structured, exhibiting a high degree of intra-component coupling, which obscured execution paths and introduced numerous side effects, as well as inter-component coupling, as the implementation of algorithms in the core business logic was not fully independent of the UI.

The first issue to address was coupling. To resolve this, we separated the system into two distinct projects: PTB, a Python package that exposes the modules and computational tree functionality through a programmatic API and does not require additional components to run in its basic form, and the OPS, a web application that serves as a GUI for PTB and manages persistence, scheduling, and the configuration and analysis workflows. This decoupled architecture separates domain business logic (which is kept within PTB) from the presentation, persistence and job orchestration layer (situated in OPS), while allowing backwards compatibility with PLCTv1.

IV. NEW PLC TESTBENCH PACKAGE

The core rationale behind the redesign of PLCTv1 into PTB was to provide a comprehensive toolkit and programming model for engineers developing PLC algorithms, independent

of any specific GUI or infrastructure technology. It is distributed as a Python package, and it can be installed from the GitHub repository¹.

A. Architecture changes

We eliminated the conflation of responsibilities between the UI layer and the computational tree abstraction. To achieve this separation, the core of PLCTv1 was decoupled from the persistence layer. The previous MongoDB-based caching and persistence mechanism was replaced with TinyDB and abstracted through a `DatabaseManager` interface, enabling interchangeable backend implementations via a factory design pattern.

At present, two database backends are supported: TinyDB, which serves as the default, and MongoDB, retained for backward compatibility. This redesign allows the toolkit to be used as a standalone Python package without requiring a MongoDB installation or external services, thereby simplifying deployment.

B. PLCAAlgorithm API

The `run()` method of the PLCAAlgorithm Module has been redesigned to incorporate a set of extensible hooks that support the newly introduced DSP capabilities while providing finer-grained control to algorithm developers. These hooks enable structured intervention points within the processing pipeline, including dedicated support for crossfade operations as well as general-purpose *a priori* and *a posteriori* processing stages. A high-level overview of the PLCAAlgorithm lifecycle is provided in Alg. 1.

C. DSP features

Crossfade support: One of the major additions is crossfade support. When PLC algorithms are used in the real world, they are often accompanied by crossfading, as predicted packets may not terminate smoothly, which can otherwise introduce audible glitches. The implementation supports both fade-in and crossfade. Fade-in applies a ramp over a configurable duration, while crossfade applies an outgoing crossfade to the prediction buffer. Crossfading can also be multiband: the signal is split into bands, and separate crossfade modules can be configured for each band. Band splitting is performed with Linkwitz-Riley crossover IIR filters.

This functionality is implemented via a stateful `Crossfade` class, representing a new type of Module in the application. The `Crossfade` class, along with its counterpart `MultiBandCrossfade`, maintains the experiment state internally, tracking the current packet index and updating it at each call. Unlike other modules, it is not subclassed directly; instead, different configurations are defined by subclassing the corresponding `CrossfadeSettings` class.

The `CrossfadeSettings` accept four parameters: `length`, `CrossfadeFunction`, `CrossfadeType`, `exponent`.

- `length`: number in samples of the fade

¹PTB: <https://github.com/CIMIL/plc-testbench>

Algorithm 1: High-level view of a PLCAAlgorithm life-cycle, in a pseudocode excerpt from the `run()` method. Monospaced snake-case denotes the hooks available to the developer, while italic and camel-case denotes abstractions of plumbing operations.

Input: Original track x , lost sample indices $\mathcal{L}$

Output: Reconstructed track $\hat{x}$

Post-initialization, pre-loop execution

`_prepare_to_play()`

for each packet index i in $\mathcal{L}$ **do**

 # Get current packet buffer and trace

$b \leftarrow \text{extractBuffer}(i, \text{packetSize})$

$is_valid \leftarrow \text{isPacketValid}(i)$

 # Packet pre-processing hook

$b' \leftarrow \text{_a_priori}(b, is_valid)$

if is_valid **then**

 # Crossfade hook

$\hat{b} \leftarrow \text{_crossfade}(b')$

else

 # Packet prediction hook

$p \leftarrow \text{_predict}(b')$

 # Fade-in hook

$\hat{b} \leftarrow \text{_fade_in}(p)$

 # Prepare crossfade for next packet

`crossfade.start()`

 # Packet post-processing hook;

 # shifts context window to include current buffer

$\hat{b} \leftarrow \text{_a_posteriori}(\hat{b}, is_valid)$

$\hat{x} \leftarrow \text{injectBuffer}(\hat{b})$

return $\hat{x}$

- **CrossfadeFunction:** shape of the fade-out envelope applied to the outbound reconstructed buffer (implemented: power, sinusoidal, hann).
- **exponent:** exponent to use when the crossfade function is power.
- **CrossfadeType:** shape of the envelope applied to the current valid buffer (implemented: amplitude (equal-gain) and power (equal-power)).

Five preset **CrossfadeSettings** subclasses are provided for convenience:

- **NoCrossfadeSettings:** dummy passthrough configuration, used as default
- **LinearCrossfadeSettings:** uses power Cross-

fadeFunction, exponent 1

- **QuadraticCrossfadeSettings:** uses power CrossfadeFunction, exponent 2
- **CubicCrossfadeSettings:** uses power CrossfadeFunction, exponent 3
- **SinusoidalCrossfadeSettings:** uses sinusoidal-CrossfadeFunction
- **ManualCrossfadeSettings:** Generic class, used to manually set the parameters

Crossfades are provided as configuration parameters to a PLCAAlgorithm Module. Each PLCAAlgorithmSettings instance can accept either a single **CrossfadeSettings** object or a list of **CrossfadeSettings** objects for crossfade, and a separate **CrossfadeSettings** object for fade-in. If a list is used for crossfade, a corresponding list of `crossfade_frequencies` (int) and a `crossover_order` (int) must also be provided; the frequency and crossover order lists must have exactly `len(crossfade) - 1` elements.

Spatial audio support: A Mid/Side Codec has been implemented as part of the toolchain offered by the PTB. The implementation applies a symmetric scaling during encoding and a compensatory gain during decoding, ensuring numerical stability while preserving signal energy and invertibility. Currently, this is available as an option within the AdvancedPLC meta-Module.

D. Modules changelog

A total of eight new built-in Modules have been added, compared to PLCTv1. The reasons why they were added vary from convenience, to updating the tool to the most recent developments of the state-of-the-art. A summary is reported in Table I, where bold denotes new modules.

PacketLossSimulators:

- **MetronomePLS:** Introduces a deterministic periodic packet loss model generating loss bursts. The simulator drops packets in fixed-length bursts (according to a duration parameter) occurring at regular intervals (period), with an optional temporal offset to shift the loss pattern. This model is particularly useful for controlled evaluation of packet loss concealment algorithms under repeatable burst-loss conditions.
- **CustomMaskPLS:** Adds support for user-defined packet loss trace via a binary mask. Each element of the mask specifies whether a packet is lost or received, enabling fully deterministic and reproducible loss scenarios. An optional inversion flag allows flipping the mask semantics, and the simulator gracefully defaults to no-loss behavior once the mask sequence is exhausted.

PLCAAlgorithm:

- **PARCnetPLC:** an implementation of the PLC algorithm proposed in [3]. Specifically, the implementation is adapted from the PARCnet revision used as baseline for the IEEE-IS² MPLC Challenge 2025 [4]. The method

features a hybrid approach, where a linear predictive coding (LPC) branch and a neural network (NN) branch predicting the residual of the LPC run in parallel.

- *AdvancedPLC*: this can be considered a meta-algorithm. It is a blueprint to streamline the design of PLC algorithms with complex behavior. It enables the definition of separate algorithms for left and right channels, supports Mid/Side coding, and allows the implementation of band-split PLC strategies. It should be noted that certain PLC algorithms, such as autoregressive methods, can be sensitive to band-splitting: if they are fitted in a full-band buffer of past, valid audio and then applied in a band-split setting, their behavior may be suboptimal. Domain knowledge is therefore essential.

OutputAnalyser:

- *WindowedPEAQCalculator*: a variant of the already implemented PEAQCalculator. It chunks the reconstructed track around intornos for every lost packet, computes PEAQ for every intorno and aggregates it in a single value.
- *PerceptualCalculator*: an implementation of the metric presented in [31]. It leverages constant q transform (CQT) and has been proven to have higher correlation with human judgement compared to other metrics on a simple experimental scenario.
- *PESQCalculator*: an implementation of PESQ [24]; specifically, the implementation leverages the `pesqc2` Python package from Audiolabs [32], which implements wideband PESQ according to the P.862.2 with Corrigendum 2 specification [33].
- *PLCMOSCalculator*: the official implementation of PLCMOS from [30]. It is a NN classifier trained on an ITU p.800.1 mean opinion score (MOS) dataset assigned by human subjects. The model is primarily non-intrusive — i.e., it does not compare the reconstructed signal with a reference.

V. OPENPLC STUDIO

The legacy UI of PLCTv1 has been rebuilt from scratch as the current OPS. The decision to declare the previous GUI end-of-life was motivated by the high degree of coupling between major architectural components, indicative of inadequate design and resulting in limited maintainability and poor extensibility. This rewrite aims to provide a maintainable, efficient, and forward-compatible system.

The application is designed for deployment on a small lab server. Consequently, several features typical of production-grade software-as-a-service (SaaS) systems, such as multi-user support and access-control lists (ACLs), were not included.

A. System architecture

OPS is designed as a full-stack web application, an architecture well suited for deployment on a small lab server and access by lab members. The system comprises a Frontend application, a Backend application handling inter-process

communication (IPC) between components, a database for data persistence, and a message broker for task scheduling and message exchange. A high-level view of the system architecture is presented in Fig. 1.

Both Frontend² and Backend³ applications are open-source and available on GitHub, along with user and developer documentation.

It is distributed as two Docker images hosted on a public registry, one image for the Frontend⁴ and one for the Backend⁵. For convenience of usage, a Docker Compose YAML file is provided⁶, with all the relevant instructions on how to deploy the application on a local server.

B. Frontend

The frontend application provides the graphical user interface and is implemented as an Angular application. It leverages several libraries, including PrimeNG⁷ for general UI components, wavesurfer.js⁸ for audio visualization, and chart.js⁹ for plotting and data visualization.

The application is organized into three main components. The *backlog* component offers an overview of all executed PLC experiments. The *run-configurator* component enables users to define and launch new PLC experiments. The *analyser* component provides tools for visualizing and evaluating the results of completed PLC experiments. Screenshots for the *run-configurator* component (Fig. 2) and the *analyser* component (Fig. 3) are provided.

C. Backend

The backend role is manifold. It exposes a REST API as the primary interface and manages data persistence through interaction with the database. It also wraps and exposes PTB, scheduling PLC experiments as asynchronous jobs via a queuing system. In addition, it supports live updates through a WebSocket endpoint.

The backend is implemented as a Python server based on FastAPI¹⁰, which provides a simple and flexible implementation of the ASGI web server specification¹¹. The choice of Python is motivated by the need to streamline integration with PTB, which is itself written in Python. MongoDB is used as the database due to its ease of setup and robustness; specifically, the Motor driver¹² is employed to ensure efficient integration with FastAPI's `asyncio`-based architecture.

From an architectural perspective, the system follows a custom structure inspired by domain-driven design [34] and hexagonal architecture [35]. In particular, it separates domain

²<https://github.com/CIMIL/openplc-studio-frontend>

³<https://github.com/CIMIL/openplc-studio-backend>

⁴<https://hub.docker.com/r/cimil/openplc-studio-frontend>

⁵<https://hub.docker.com/r/cimil/openplc-studio-backend>

⁶<https://github.com/CIMIL/openplc-studio>

⁷PrimeNG: <https://primeng.org/>

⁸wavesurfer.js: <https://wavesurfer.xyz/>

⁹chart.js: <https://www.chartjs.org/>

¹⁰FastAPI: <https://fastapi.tiangolo.com/>

¹¹ASGI specifications: <https://asgi.readthedocs.io/en/latest/specs/main.html>

¹²<https://motor.readthedocs.io/en/stable/>

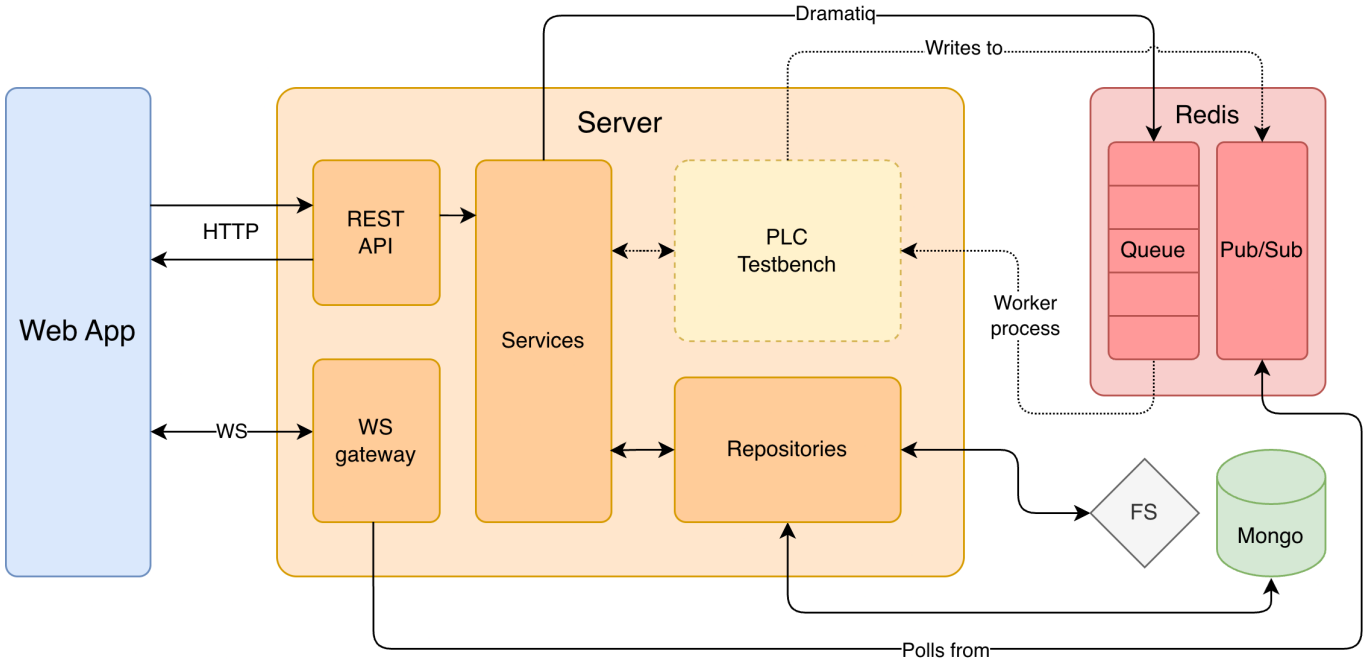


Fig. 1. High-level view of OPS system architecture. Solid black lines denote in-process communication; dotted lines denote IPC. The PLC Testbench block within the Server block represent PTB and is displayed with dotted stroke to denote that it is always executed in a separate worker process.

logic (encapsulated in software components named *services*), data access and infrastructure concerns (handled by software components named *repositories*), and interface layers (REST and WebSocket APIs).

API layer: There are two separate API layers: REST and WS. The REST layer exposes four main endpoints: `/runs`, `/modules` and `/assets`

- `/runs`: the most important REST endpoint, exposes operations regarding runs, which are the main entity in the data model of the application;
- `/modules`: used primarily to read metadata and information about the Modules from the PLC Testbench package;
- `/assets`: used to handle upload of audio tracks and fetching of artifacts produced by the run;

The WebSocket API sends two main types of messages: `RunCompletion` and `RunProgress`, the former delivering information about whether a run has completed and with which status, and the latter providing a polled snapshot of the progress, useful for live progress feed purposes.

Persistence layer and data model: Persistence is managed using a NoSQL database; specifically, we use MongoDB accessed via the asynchronous Motor driver. The data model comprises two primary entities: runs and modules. A run is characterized by an identifier, creation and update timestamps, a name, a reference to the corresponding internal node identifier within the PLC Testbench package, a status, a collection of tracks, and a module configuration. The module configuration is represented as a dictionary that maps each module type—where types correspond to those defined in the PTB—to a list of associated modules.

In OPS, a Module is an interoperability construct that bridges the backend data model, the frontend, and the PTB data model. Its design follows a schema-driven UI approach: rather than transmitting only runtime values, the backend also exposes metadata that enables the frontend to render and validate user inputs dynamically. Each Module consists of a name and a set of settings, where each setting is a `ModuleParameter` modeled as a name-value pair. For each parameter and module, a corresponding Spec representation is provided, which exposes descriptive metadata such as the value type, default value, allowed range, and the set of valid values for enumerated types. This metadata enables the frontend to generate the appropriate user interface components and validation logic without hardcoding parameter-specific rules, thereby ensuring consistency between the backend data model and the frontend representation.

Job scheduling: A PLC experiment is a CPU-bound task; therefore, offloading computation is necessary to maintain UI responsiveness. To achieve this, we employ an asynchronous job queue with Redis as the message broker. When a user creates a run, the REST API persists the run metadata to the database and issues a corresponding job in Redis. Once a worker becomes available, the job is consumed and executed asynchronously.

Worker management and message handling are implemented using the Dramatiq Python library, which provides a concise and flexible API based on decorated Python functions (actors) that act as message consumers. These actors are not executed within the main application process; instead, they are launched as separate processes via the Dramatiq CLI. Within a Docker-based deployment, this architecture is realized by

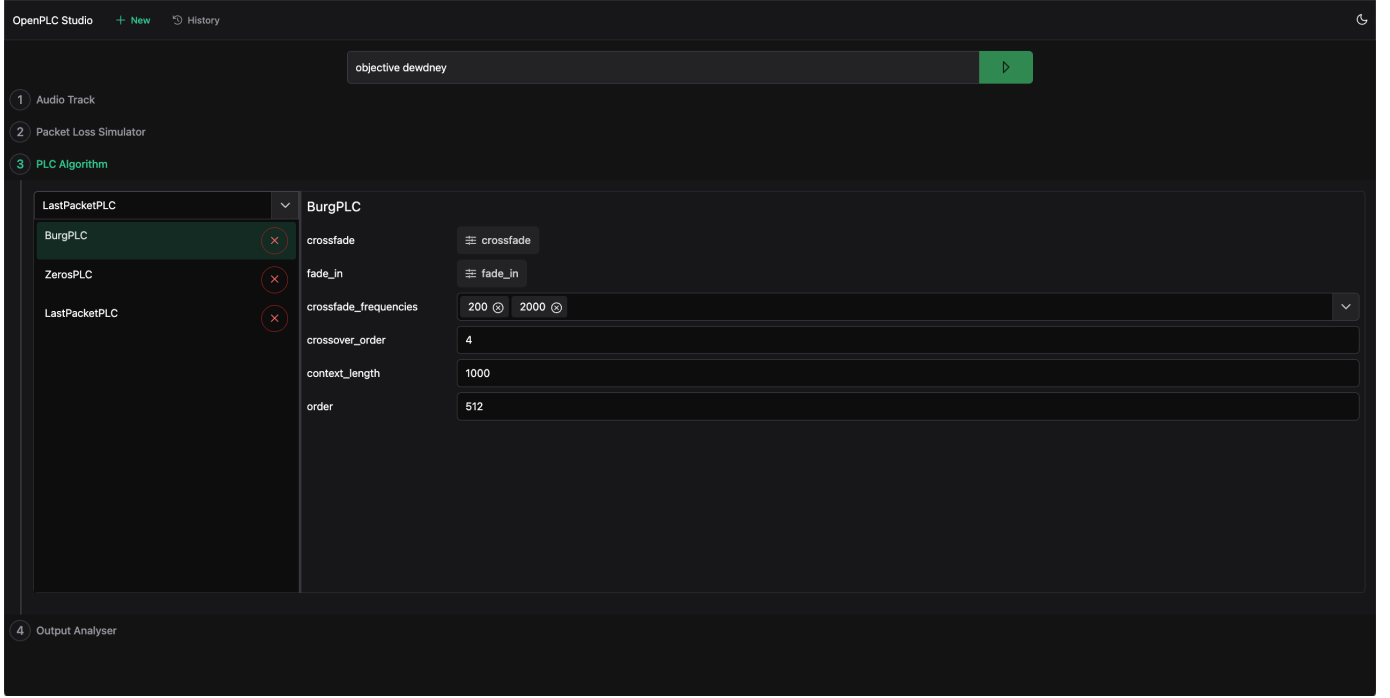


Fig. 2. Screenshot of a section of the *run-configurator* component of OPS.

instantiating two containers from the same backend image: one for the main backend service and another dedicated to Dramatiq workers.

Additionally, since the workers operate in a separate address space and do not expose the REST or WebSocket communication endpoints, we employ Redis as a message bus to enable IPC. For convenience, the same Redis instance used as the message broker is also used for this purpose. Workers publish run progress updates to a Redis pub/sub channel, while the WebSocket gateway in the main process polls this channel and forwards updates to clients via WebSocket messages as they become available.

Plug-in system: We provide a plug-in system that enables users of OPS to define custom PLC algorithms without modifying or forking PTB, which would require a customized deployment of the app. In OPS, a *plug-in* is a Python module located in a user-specified directory and conforming to a predefined API based on naming conventions. Each plug-in module is required to include the following components:

- A top-level docstring specifying the plug-in name and parameters, formatted as a YAML object.
- A Python class implementing the algorithm logic. The class may have an arbitrary name but must inherit from the `PLCAlgorithm` base class. This class exposes several extensible hooks to facilitate the implementation of complex PLC behaviors; see Alg. 1 for a high-level description of the execution lifecycle.
- A Python class defining the algorithm configuration. This class must inherit from `PLCSettings` and follow the naming convention

TABLE I
SUMMARY OF THE MODULES AVAILABLE IN OPS.

Module name	Type	Notes
BinomialPLS	PacketLossSimulator	[36]
GilbertElliotPLS	PacketLossSimulator	
MetronomePLS	PacketLossSimulator	
CustomMaskPLS	PacketLossSimulator	
ZerosPLC	PLCAlgorithm	[15]
LastPacketPLC	PLCAlgorithm	
LowCostPLC	PLCAlgorithm	
BurgPLC	PLCAlgorithm	
ExternalPLC	PLCAlgorithm	[16] ^a
VermaPLC	PLCAlgorithm	
PARCnetPLC	PLCAlgorithm	
AdvancedPLC	PLCAlgorithm	
MAECalculator	OutputAnalyser	[23]
MSECalculator	OutputAnalyser	
PEAQCalculator	OutputAnalyser	
PESQCalculator	OutputAnalyser	
PLCMOSCalculator	OutputAnalyser	
PerceptualCalculator	OutputAnalyser	

^a Name change with respect to PLCTv1: *DeepLearning-PLC* → *VermaPLC*.

`<AlgorithmClassName>Settings`. It must implement an `__init__()` method through which the algorithm parameters are provided.

At present, the plug-in system supports only modules of type `PLCAlgorithm`; other module types, such as `PacketLossSimulator` and `OutputAnalyser`, are not yet supported. Full details of the plug-in API are provided in the Backend repository.

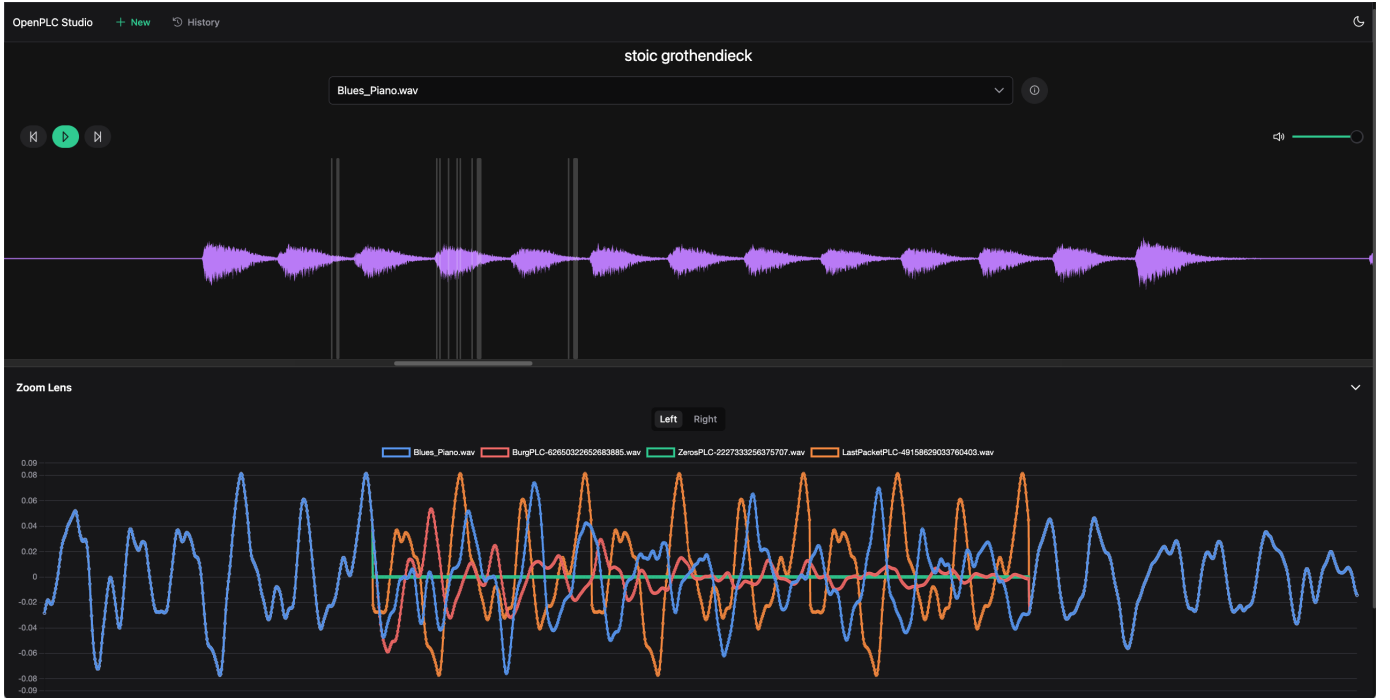


Fig. 3. Screenshot of a section of the *analyser* component of OPS.

VI. DISCUSSION AND CONCLUSIONS

This paper presented OPS, a novel iteration of the original PLCTv1 into a modular and extensible research toolkit for the development, evaluation, and comparison of audio packet loss concealment algorithms. We aimed to improve its reliability and convenience by re-engineering it into separate software components, PTB and OPS, independently packaged and distributed. Essential features in PLC algorithm development were also added to PTB, most importantly crossfading between PLC in the predicted buffer and the valid incoming buffer. Additionally, new built-in modules were implemented, with the goal of updating the tool to the most recent advancements of PLC algorithm development and evaluation.

Importantly, beyond the software engineering improvements, the proposed application contributes toward establishing a common experimental infrastructure for PLC research. By standardizing the workflow from packet loss simulation to objective evaluation, while exposing extensible interfaces for algorithms, simulators, and metrics, the application streamlines the development and assessment process of PLC algorithms. Consequently, researchers can devote more effort to algorithmic innovation rather than repeatedly implementing experimental tooling, while future community initiatives, such as PLC challenges and benchmarking studies, can build upon a shared and openly available foundation.

Although the current architecture provides a robust and practical tool for the community, several extensions remain to be explored. When it comes to features specific to the domain of PLC algorithm analysis, future work may include support for real-time performance benchmarking, as the current system

is primarily optimized for offline analysis. The plugin system could also be extended to support the development of user-defined PacketLossSimulator and OutputAnalyser Modules. Furthermore, integrating an in-app interface for performing listening tests — potentially inspired by interfaces such as webMUSHRA — would enhance PLC algorithm evaluation capabilities. On the technical side, potential improvements include support for chunked track processing and further abstraction of file system interactions within OPS for assets management.

A limitation of the present work is that no user evaluation or usability study was conducted, so the practical impact on researchers and practitioners remains to be assessed. Nevertheless, the application was preliminarily assessed by the authors to verify that the redesigned architecture satisfies the engineering goals motivating the re-implementation of PLCTv1, namely modularity, extensibility, and compatibility. Representative PLC experiments from PLCTv1 were successfully reproduced with consistent objective scores and reconstructed audio. A custom PLC algorithm was integrated as an external plugin within OPS. The complete Docker-based stack was deployed on a standard Linux server to run multiple concurrent experiments through the web interface. These checks provide only preliminary, author-driven evidence of technical correctness and extensibility, and a systematic user study is left for future work.

By making the application openly available, the authors hope to foster reproducible experimentation, facilitate collaboration within the PLC research community, and accelerate future advances in audio PLC.

ACKNOWLEDGMENT

The corresponding author would like to thank Ivan Tonidandel for his substantial contributions to the development of OPS.

REFERENCES

- [1] Lorenz Diener and Sten Sootla and Solomiya Branets and Ando Saabas and Robert Aichner and Ross Cutler, "INTERSPEECH 2022 Audio Deep Packet Loss Concealment Challenge," in *Interspeech 2022*, 2022, pp. 580–584.
- [2] L. Diener, S. Branets, A. Saabas, and R. Cutler, "The icassp 2024 audio deep packet loss concealment grand challenge," *IEEE Open Journal of Signal Processing*, vol. 6, pp. 231–237, 2025.
- [3] A. I. Mezza, M. Amerena, A. Bernardini, and A. Sarti, "Hybrid packet loss concealment for real-time networked music applications," *IEEE Open Journal of Signal Processing*, vol. 5, pp. 266–273, 2024.
- [4] A. I. Mezza, A. Bernardini, and C. Rinaldi, "The icassp 2024 music packet loss concealment challenge," in *2025 IEEE 6th International Symposium on the Internet of Sounds (IS2)*. IEEE, 2025, pp. 1–10.
- [5] L. Vignati, S. Dallona, and L. Turchet, "Plc testbench: a modular tool for the study and comparison of audio packet loss concealment algorithms," in *2023 4th International Symposium on the Internet of Sounds*, 2023, pp. 1–8.
- [6] C. Perkins, O. Hodson, and V. Hardman, "A survey of packet loss recovery techniques for streaming audio," *IEEE network*, vol. 12, no. 5, pp. 40–48, 1998.
- [7] D. Goodman, G. Lockhart, O. Wasem, and W.-C. Wong, "Waveform substitution techniques for recovering missing speech segments in packet voice communications," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 6, pp. 1440–1448, 1986.
- [8] O. Wasem, D. Goodman, C. Dvorak, and H. Page, "The effect of waveform substitution on the quality of pcm packet communications," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 3, pp. 342–348, 1988.
- [9] H. Sanneck, A. Stenger, K. Ben Younes, and B. Girod, "A new technique for audio packet loss concealment," in *Proceedings of GLOBECOM '96. 1996 IEEE Global Telecommunications Conference*, vol. MiniConfInternet, 1996, pp. 48–52.
- [10] E. Gunduzhan and K. Momtahan, "Linear prediction based packet loss concealment algorithm for pcm coded speech," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 8, pp. 778–785, 2001.
- [11] G. Zhang and W. B. Kleijn, "Autoregressive model-based speech packet-loss concealment," in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2008, pp. 4797–4800.
- [12] C. Rodbro, M. Murthi, S. Andersen, and S. Jensen, "Hidden markov model-based packet loss concealment for voice over ip," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1609–1623, 2006.
- [13] J.-M. Valin, A. Mustafa, C. Montgomery, T. B. Terriberry, M. Klingbeil, P. Smaragdakis, and A. Krishnaswamy, "Real-time packet loss concealment with mixed generative and predictive model," 2022.
- [14] B. Irvin, S. Yin, S. Zhang, and M. Stamenovic, "A fullband neural network for audio packet loss concealment," in *2024 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*, 2024, pp. 107–108.
- [15] M. Fink and U. Zölzer, "Low-delay error concealment with low computational overhead for audio over ip applications," in *Proceedings of the International Conference on Digital Audio Effects*, 2014, pp. 309–316.
- [16] 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.
- [17] A. I. Mezza and A. Bernardini, "Ieee-is² 2024 music packet loss concealment challenge," Internet of Sounds, Tech. Rep., oct 2024, accessed: Feb. 5, 2026. [Online]. Available: https://internetofsounds.net/public_downloads/ieee-is2-music-plc-challenge-report.pdf
- [18] C. Aironi, L. Gabrielli, S. Cornell, and S. Squartini, "Enhancing music packet loss concealment with generative spectrogram inpainting," Università Politecnica delle Marche, Tech. Rep., 2024, accessed: Feb. 5, 2026. [Online]. Available: https://internetofsounds.net/public_downloads/Aironi_tech_report.pdf
- [19] F. Daniotti, L. Vignati, and L. Turchet, "Tilt loss: a perceptual loss function to improve music packet loss concealment," *EURASIP Journal on Audio, Speech, and Music Processing*, 2026. [Online]. Available: <https://doi.org/10.1186/s13636-025-00442-1>
- [20] C. Aironi, L. Gabrielli, S. Cornell, and S. Squartini, "Restoring music integrity via predictive modeling and spectral inpainting," Università Politecnica delle Marche, Tech. Rep., 2025, accessed: Feb. 5, 2026. [Online]. Available: https://internetofsounds.net/publicdownloads/Aironi_tech_report_2025.pdf
- [21] F. Daniotti and L. Turchet, "Conditioning PARCnet with TFiLM for robust packet loss concealment," University of Trento, Tech. Rep., 2025, accessed: Feb. 5, 2026. [Online]. Available: https://internetofsounds.net/public_downloads/Daniotti_techreport2025.pdf
- [22] L. Severi and C. Rottondi, "Sparse linear prediction for packet loss concealment in networked music performances," in *2025 IEEE 6th International Symposium on the Internet of Sounds (IS2)*, 2025, pp. 1–5.
- [23] T. V. 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, pp. 3–29, 2000. [Online]. Available: <https://api.semanticscholar.org/CorpusID:16573563>
- [24] A. Rix, J. Beerends, M. Hollier, and A. 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, 2001, pp. 749–752 vol.2.
- [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] ITU, "P.800.1 : Mean opinion score (MOS) terminology." [Online]. Available: <https://itu.int/rec/T-REC-P.800.1>
- [27] ITU-T, "Subjective diagnostic test method for conversational speech quality analysis," International Telecommunication Union, Recommendation P.804, 10 2017, approved 2017-10-29, in force. [Online]. Available: <https://www.itu.int/rec/T-REC-P.804/en>
- [28] ITU, "BS.1534 : Method for the subjective assessment of intermediate quality level of audio systems." [Online]. Available: <https://www.itu.int/rec/R-REC-BS.1534/en>
- [29] M. Schoeffler, S. Bartoschek, F.-R. Stöter, M. Roess, S. Westphal, B. Edler, and J. Herre, "webmushra — a comprehensive framework for web-based listening tests," *Journal of Open Research Software*, Feb 2018.
- [30] L. Diener, M. Purin, S. Sootla, A. Saabas, R. Aichner, and R. Cutler, "Plcmos – a data-driven non-intrusive metric for the evaluation of packet loss concealment algorithms," in *Proceedings of the 23rd Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Incheon, Korea, 2022.
- [31] L. Vignati and L. Turchet, "On the lack of a perceptually-motivated evaluation metric for packet loss concealment in networked music performances," *Journal of the Audio Engineering Society*, 2025, in press.
- [32] M. Torcoli, M. M. Halimeh, and E. A. P. Habets, "Navigating pesq: Up-to-date versions and open implementations," in *Speech Communication; 16th ITG Conference*, 2025, pp. 51–55.
- [33] ITU-T, "Wideband extension to recommendation p.862 for the assessment of wideband telephone networks and speech codecs," International Telecommunication Union, Geneva, Switzerland, ITU-T Recommendation P.862.2 P.862.2, Nov. 2007, corrigendum: P.862.2 (2007) Cor. 1 (10/2017).
- [34] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.
- [35] A. Cockburn, "Hexagonal architecture," HaT Technical Report, Technical report 2005.02, Sep. 2005. [Online]. Available: <https://alistair.cockburn.us/hexagonal-architecture/>
- [36] E. O. Elliott, "Estimates of error rates for codes on burst-noise channels," *Bell System Technical Journal*, vol. 42, no. 5, pp. 1977–1997, Sep. 1963.
- [37] M. Sacchetto, C. Rottondi, and A. Bianco, "Implementation and optimization of burg's method for real-time packet loss concealment in networked music performance applications," *Personal and Ubiquitous Computing*, pp. 1–17, 2024.