pyBeamline Designer: A No-Code Platform for Streaming Process Mining Pipelines

Arturo Cortes[1], Sotero Romero[1] and Andrea Burattin[1]

[1]*DTU Compute, Technical University of Denmark, Kgs. Lyngby, Denmark*

Abstract

This paper presents *pyBeamline Designer*, a web-based no-code platform developed for the *construction*, *validation*, and *execution* of streaming process mining pipelines. Built as a front-end application atop the pyBeamline library, it enables users to visually define streaming process mining workflows through an intuitive drag-and-drop interface. The platform automatically generates valid Python code for these pipelines and supports built-in live execution and export functionalities. Our contribution enhances the usability, accessibility, and prototyping of streaming process mining applications, especially for non-expert users, and represents a novel integration of low-code development principles within the process mining domain. The tool is open-source, thoroughly evaluated in a user study, and freely available online.

Keywords

Streaming Process Mining, No-Code Development, Visual Programming, pyBeamline

Metadata description	Value
Tool name	pyBeamline Designer
Current version	1.0
Legal code license	Apache 2.0
Languages, tools and services used	TypeScript
Supported operating environment	Any browser
Application URL	`https://beamline.github.io/pybeamline-designer/`
Documentation URL	`https://beamline.github.io/pybeamline-designer/`
Source code repository	`https://github.com/beamline/pybeamline-designer/`
Screencast video	`https://youtu.be/eCkfo1CTgog`

1. Introduction

Streaming process mining [1] is an increasingly important research direction, allowing for the analysis of process execution data in real time. Frameworks such as Beamline [2] and its Python counterpart, pyBeamline [2], offer a flexible environment for developing streaming pipelines using asynchronous event processing. However, despite their expressiveness, these frameworks impose a considerable entry barrier, particularly for users unfamiliar with programming or the functional reactive paradigm [3].

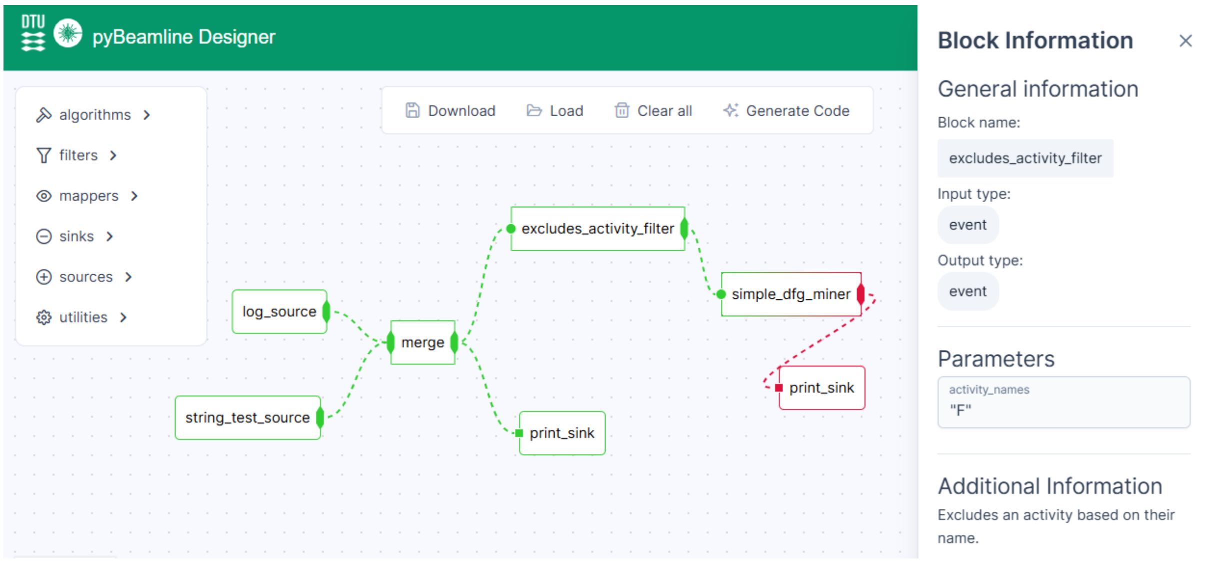
To address this gap, we developed pyBeamline Designer, an interactive, no-code [4] platform that allows users to visually construct streaming process mining pipelines. Our tool removes the need for manual scripting by translating diagrammatic representations into executable Python code. The platform includes features such as real-time error detection, runtime execution, and extendability through user-defined operators. It is fully implemented as a stateless web application, requiring no backend infrastructure, intended for broader accessibility to streaming process mining techniques.

2. Motivation and Contribution

The motivation for this work stems from the need to lower the technical barriers associated with streaming process mining. While pyBeamline simplifies prototyping relative to Beamline, it still

ICPM Doctoral Consortium and Demo Track 2025, October 20-24, 2025, Montevideo, Uruguay

s225187@student.dtu.dk (A. Cortes); s225145@student.dtu.dk (S. Romero); andbur@dtu.dk (A. Burattin)



Block Information

General information

Block name: excludes_activity_filter

Input type: event

Output type: event

Parameters

activity_names: "F"

Additional Information

Excludes an activity based on their name.

Figure 1: An example pipeline in pyBeamline Designer with the configuration panel for the `excludes_activity_filter` operator.

demands a strong grasp of asynchronous programming and stream composition. This complexity can hinder rapid experimentation and reduce adoption among non-technical stakeholders. Our contribution is the design and implementation of a web-based no-code platform that complements pyBeamline. By enabling users to create pipelines visually, inspect their logical flow, and receive immediate feedback on potential errors, pyBeamline Designer improves both the efficiency and the approachability of pipeline development. Furthermore, the ability to export pipelines as fully functioning Python scripts allows for integration with existing workflows, making the platform suitable for both teaching and applied research.

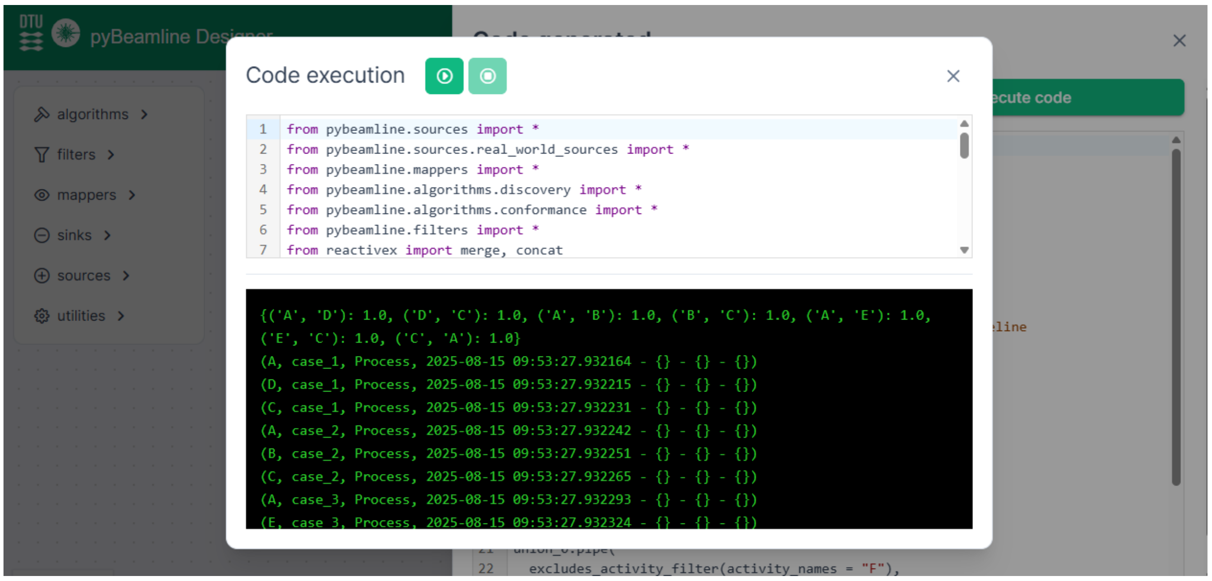
3. Tool Description

pyBeamline Designer is implemented as a fully client-side web application using modern front-end technologies, including Vue.js (`https://vuejs.org`) for the underlying framework, TypeScript for static typing and maintainability, and PrimeVue (`https://primevue.org`) and VueFlow (`https://vueflow.dev`) for user interface and diagramming capabilities. The platform enables users to design streaming process mining pipelines using a visual flow-based model, where blocks representing operators are connected via edges to form a directed acyclic graph.

Each block in the designer corresponds to a concrete pyBeamline operator [2], such as a *log source*, *transformation operator*, *filter*, *miner*, or *sink*. The user selects blocks from a categorized component menu, which mirrors the structure of the pyBeamline operator schemas. These blocks can be dragged into a central whiteboard area, where they can be connected to other blocks to define the data flow (cf. Fig. 1). Each connection corresponds to the propagation of an event stream in the pyBeamline execution model.

Under the hood, each block is backed by a JSON schema that describes its structural characteristics, including its expected input and output types, required parameters, valid values, and pyBeamline function mapping. The tool leverages the AJV (`https://ajv.js.org`) validation library to dynamically check that user-constructed pipelines conform to these constraints[1]. Type mismatches, missing parameters, and invalid block configurations are detected in real time and highlighted directly into the pipeline, guiding users toward a valid and executable pipeline.

[1]The complete JSON schema is availble at `https://github.com/beamline/pybeamline-designer/tree/master/src/model/schemas`.



```
from pybeamline.sources import *
from pybeamline.sources.real_world_sources import *
from pybeamline.mappers import *
from pybeamline.algorithms.discovery import *
from pybeamline.algorithms.conformance import *
from pybeamline.filters import *
from reactivex import merge, concat
```

```
{('A', 'D'): 1.0, ('D', 'C'): 1.0, ('A', 'B'): 1.0, ('B', 'C'): 1.0, ('A', 'E'): 1.0,
('E', 'C'): 1.0, ('C', 'A'): 1.0}
(A, case_1, Process, 2025-08-15 09:53:27.932164 - {} - {} - {})
(D, case_1, Process, 2025-08-15 09:53:27.932215 - {} - {} - {})
(C, case_1, Process, 2025-08-15 09:53:27.932231 - {} - {} - {})
(A, case_2, Process, 2025-08-15 09:53:27.932242 - {} - {} - {})
(B, case_2, Process, 2025-08-15 09:53:27.932251 - {} - {} - {})
(C, case_2, Process, 2025-08-15 09:53:27.932265 - {} - {} - {})
(A, case_3, Process, 2025-08-15 09:53:27.932293 - {} - {} - {})
(E, case_3, Process, 2025-08-15 09:53:27.932324 - {} - {} - {})
```

```
22    excludes_activity_filter(activity_names = "F"),
```

Figure 2: The execution environment built inside of pyBeamline Designer.

When a block is selected, a parameter configuration panel is shown in a sidebar (cf. Fig. 1). This panel includes helpful documentation, input forms for each required argument, and type hints to ensure proper usage. For instance, a `excludes_activity_filter` block might require a list of activities to exclude. Parameters are specified using a combination of text boxes, dropdowns, and – where appropriate – code editors.

The tool also supports advanced usage scenarios through its implementation of "wildcard blocks." These customizable blocks allow users to embed arbitrary Python code directly within the platform. A code editor embedded in the sidebar allows the user to define new operator functions or sink behaviors, which are automatically injected into the script during the code generation phase. This enables users to prototype and test experimental operators or specialized logic without modifying the core application.

Once the pipeline is completely drawn, it is possible to generate the corresponding code. The code generation module traverses the user-defined diagram using a depth-first search strategy. The traversal respects the logical execution order of the pipeline and constructs a complete Python script that includes import statements, source and operator instantiations, and subscription logic. The generated script is presented to the user in a code viewer and can be downloaded as a standalone file.

To ensure reproducibility and continuity of work, users can save and load their pipelines as JSON files. These files capture the full state of the diagram, including node positions, parameters, and connections, making it easy to archive, share, or resume work at a later time. An action bar at the top of the interface (cf. Fig. 1) provides access to all core functions, including saving, loading, resetting, and generating the code.

The tool includes support for the execution of pipelines within the browser by connecting to a local Jupyter Kernel Gateway (`https://jupyter-kernel-gateway.readthedocs.io`). Fig. 2 shows the interface with the result of the execution of the generated pyBeamline code directly inside pyBeamline Designer. More complex pipelines can be exported and executed externally in a full Python environment where pyBeamline is installed. The ability to run code directly from within the tools fosters more rapid prototyping and faster development/test lifecycles. Visually, the tool emphasizes clarity and feedback. Each port type of the pipeline operators is color-coded and shaped according to its type and connection constraints (cf. Fig. 1). For example, a port that accepts multiple incoming connections (e.g., union operators) is visually distinct from a port that allows only a single input. Animations along the edges indicate the direction of the events flowing, and the interface dynamically updates to reflect the connection status of each node. In summary, pyBeamline Designer offers a comprehensive and user-friendly environment for constructing and experimenting with streaming process mining

```python
# Imports [omitted]

# Area reserved for user-defined parameters
# You can define variables here that will be used in the pipeline

# pyBeamline code
source_0 = log_source(log = ["ADC","ABC","AEC"])
pipe_0 = source_0.pipe()

source_1 = string_test_source(iterable = ["ABC","ADC"])
pipe_1 = source_1.pipe()

union_0 = merge(pipe_0, pipe_1)

union_0.pipe(
  excludes_activity_filter(activity_names = "F"),
  simple_dfg_miner()
).subscribe(on_next = lambda x : print(str(x)))

union_0.pipe().subscribe(on_next = lambda x : print(str(x)))
```

Listing 1: Generated pyBeamline code from the visual pipeline in Fig. 1.

pipelines. Its architecture emphasizes modularity, real-time validation, and extensibility, making it suitable for both novice users exploring process mining for the first time and advanced users seeking a rapid prototyping platform for complex pipeline configurations.

4. Example Pipeline and Code Generation

To illustrate the core functionality of pyBeamline Designer, we present a minimal yet representative example of a streaming process mining pipeline constructed in the tool (cf. Fig. 1) and the Python code it generates (cf. List. 1). In the designer interface, a `log_source` and a `string_test_source` blocks are added to emit event data. The events are combined together using the `merge` operator. From here, all events are printed using a `print_sink` and they are also sent to an `exclude_activity_filter`, which removes events corresponding to activity “F”. The output of the filter is then forwarded to a `simple_dfg_miner` block, which derives a directly-follows graph (DFG) from the filtered stream and, finally, the resulting DFG is passed to a sink block that prints the result. Fig. 1 shows the pipeline. The user specifies the activity to filter (e.g., “F”) through the block’s parameter sidebar. Once the diagram is complete and validated, the tool generates the corresponding Python script shown in Listing 1. The generated code is syntactically correct and executable in any Python environment where the pyBeamline is installed.[2] The script reflects the sequence of operations defined visually in the tool, with function parameters and chaining logic correctly preserved.

This example demonstrates the platform’s core objective: enabling users to define complex streaming logic visually and obtain robust, ready-to-run Python implementations. Larger pipelines involving *joins*, *unions*, or custom user-defined functions are handled similarly, with the tool automatically managing function declarations, type validations, and connection logic.

5. Maturity and Evaluation

The tool has reached a mature state, having satisfied a set of functional and non-functional requirements [5, Ch. 6]. These include support for saving/loading pipelines, live code execution, error detection, diagram-to-code translation, and extensibility. All functionalities are provided through a self-contained front-end interface, in line with modern web deployment practices.

[2]The library can be installed with `pip install pybeamline`.

To evaluate the platform's usability and effectiveness, we conducted a user study involving 20 participants from both technical and non-technical backgrounds. Participants were tasked with validating pipelines and completing a series of guided exercises. Quantitative and qualitative feedback indicated that the platform significantly reduced the complexity of building streaming pipelines (compared to just using pyBeamline) and was perceived as intuitive and efficient. Fig. 3 shows an excerpt of the results where time and correctness were compared between pyBeamline Designer and plain pyBeamline. Participants consistently achieved similar or higher correctness with pyBeamline Designer, while also completing the tasks significantly faster. All results are available at [5].

From a development standpoint, the tool adheres to modular design principles and maintains compatibility with evolving versions of pyBeamline. The use of schema-based operator definitions also ensures that new operators can be added without modifying core logic, contributing to long-term maintainability.

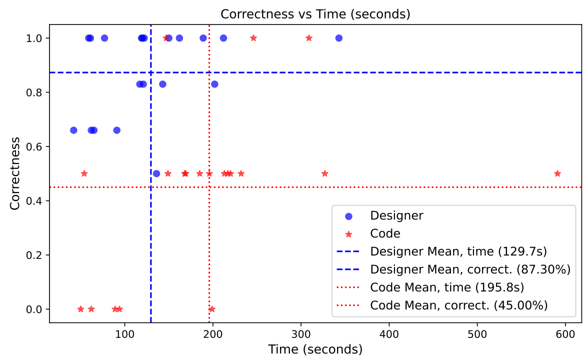


Figure 3: Results of an evaluation that compared pyBeamline Designer and plain code.

6. Conclusion

We presented pyBeamline Designer, a no-code, browser-based platform for visually creating, validating, and executing streaming process mining pipelines. By combining an intuitive interface with automatic code generation and real-time feedback, the tool reduces complexity, accelerates development, and broadens access to pyBeamline.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: Grammar and spelling check. After using this service, the authors reviewed the content and take full responsibility for the publication's content.

References

[1] A. Burattin, Streaming Process Mining, in: Process Mining Handbook, Springer, 2022, pp. 349–372. doi:10.1007/978-3-031-08848-3{_}11.

[2] A. Burattin, Beamline: A comprehensive toolkit for research and development of streaming process mining, Software Impacts 17 (2023) 100551.

[3] G. Salvaneschi, A. Margara, G. Tamburrelli, Reactive Programming: A Walkthrough, in: Proc. of ICSE, IEEE, 2015, pp. 953–954. doi:10.1109/ICSE.2015.303.

[4] G. S. Guaki, G. P. Genove, A literature review on low code and no code (LCNC) platforms in reshaping web and application development, 2025, p. 030021.

[5] A. Cortes, S. Romero, Development of an interactive, no-code process mining platform based on Beamline/pyBeamline, DTU, 2025. URL: https://findit.dtu.dk/en/catalog/686f04e2aac7a30102deea0e.