MHz - Tomoscopy

D2.2

Fast Data Interface (Software)

Project information

Project full title	MHz rate mulTiple prOjection X-ray MicrOSCOPY
Project acronym	MHz-TOMOSCOPY
Grant agreement no.	101046448
Instrument	EIC Pathfinder Open
Duration	42 months
Website	https://tomoscopy.eu/

Deliverable information

Deliverable no.	Deliverable 2.2
Deliverable title	Fast Data Interface
Deliverable responsible	Ing. Peter Szeles
Related Work-Package/Task	Task 2.2
Type (e.g. report; other)	Report, Software
Author(s)	Ing. Peter Szeles
Dissemination level	Public
Document Version	1.0.0

MHz - Tomoscopy

Date	31st of May 2024
Download page	https://git.xfel.eu/pszeles/fatra

Document information

Version no.	Date	Author(s)	Comment
1.0.0	10.05.2024	Peter Szeles	First full version

Abstract

FATRA (FAst Train Review Aplication) empowers researchers by providing near real-time visualization of scientific camera data streams. It tackles challenges associated with manual data access from Shimadzu cameras at the MHz-Tomoscopy project. FATRA leverages the European XFEL's Karabo Framework and PyQT for a user-friendly interface. Key functionalities include real-time monitoring, past data review, convenient playback, data export, and offline analysis. While a 15-30 second latency exists, FATRA strives to minimize delays while offering a streamlined user experience.

Executive Summary

FATRA addresses the need for efficient data visualization in MHz-Tomoscopy experiments using Shimadzu cameras. It replaces a cumbersome manual process with a user-friendly application for real-time and offline analysis of scientific camera data streams. FATRA offers significant improvements over its predecessor through a modular design, multithreading, a streamlined interface, dedicated detector view, and offline analysis capabilities. It successfully facilitated the Megahertz imaging experiment at the European XFEL.

MHz - Tomoscopy

Table of Contents

1. FATRA (FAst Train Review Aplication)

The MHz-Tomoscopy project utilizes Shimadzu cameras to capture sample activity during experiments in multiprojectional way. Ideally, researchers would see these camera feeds **almost instantaneously** within the control room. However, achieving true simultaneity presents technical challenges.

1.1. Challenges of Manual Data Access

Recorded frames by Shimadzu high speed cameras are transferred to the XFEL's Online cluster. This approach caused delays due to:

- Camera speed limitations and data transfer SW API which causes more then 15 seconds delay between the creation of data and finished transfer of data to the online cluster.
- User needs to manually access the data.
- Multiple data streams requiring constant monitoring across separate windows.

This cumbersome process led to information overload and hindered efficient analysis.

FATRA (Fast Train Review Application) goes beyond fast data review. It empowers researchers in the MHz-Tomoscopy project, and any experiment which uses shimadzu, Jungfrau or similar cameras with **near real-time visualization** of scientific camera data streams, nicknamed "trains."

FATRA leverages the European XFEL's Karabo Framework for seamless integration and utilizes the PyQT library for a user-friendly graphical user interface (GUI).

1.2. Main features and functionalities of FATRA

- **Real-time monitoring**: Users can monitor the live output from multiple cameras simultaneously within the FATRA interface.
- **Past data review**: FATRA allows users to review previously captured data trains saved in local cache, comprising collections of images.
- **Convenient playback**: Past data trains can be conveniently replayed in a video format for further analysis or visualization purposes.
- **Data export**: FATRA offers functionalities to export the captured videos, facilitating their seamless download for archival or in-depth analysis outside the control room environment.

- **Offline analysis**: The user (vith valid access rights) can connect to the Online clusters proposal folder where every data mesured by the experiment is placed and view the past data in similar window like the online view.

1.3. Latency Considerations:

It is important to acknowledge that the term "live" may not be entirely accurate. Processing delays exist due to:

- Camera processing time
- Data transfer through an intermediary PC
- Network communication limitations

These factors contribute to a latency of approximately 15-30 seconds between image capture and display on the screen. The application strives to minimize this delay while providing a user-friendly, streamlined interface for near real-time visualization.

1.4. The development process and lifecycle

The development of FATRA began with a prototype application called the "fast data interface." Here's a breakdown of the key stages:

1. Initial Planning and Environment Setup:

- **Requirements Gathering:** We started by gathering detailed software requirements to understand the functionalities needed for the final application.
- **Environment Analysis:** The environment where FATRA would be hosted and used was carefully analyzed to ensure compatibility and efficient operation.
- **Version Control and Task Management:** A dedicated repository was created on XFEL's GitLab (https://git.xfel.eu/pszeles/) to manage code versions and track development tasks.

2. Backend Development and Test Environment Creation:

- **Backend Focus:** Initial development focused on building the core functionalities of the application (backend).
- **Data Stream Simulation:** Since XFEL access wasn't readily available, a method to simulate camera data streaming was crucial. We successfully achieved local streaming of past XFEL data using the "extra-data" Python library.

3. User Interface (GUI) Development and Demo Creation:

- **GUI Design and Implementation:** With a functional backend, the development of the graphical user interface (GUI) commenced.
- **Demo Completion and Testing:** By the end of 2023, a working demo version with a simulated environment and GUI was completed.

4. Demo Testing, Redesign, and FATRA is Born:

- **Testing and Evaluation:** January 2024 saw the demo version tested at the XFEL online cluster. While functional, numerous bugs were identified.
- **Refined Architecture and Redesign:** Leveraging the demo's insights, a completely new architecture and design structure were created (refer to the figure 1. for details).
- **FATRA Emerges:** This major overhaul led to the application being christened FATRA with a new GitLab repository established.

5. FATRA's Feature Implementation:

The new architecture introduced significant improvements:

- **Modular Design:** Separation of backend, frontend, and simulator into distinct applications.
- **Multithreading:** Blocking processes were multithreaded for enhanced performance.
- **Streamlined Interface:** A single main window allows adding backend connections for data streaming.
- **Dedicated Detector View:** Users can open a separate window to visualize and analyze detector data.
- **Offline Analysis:** FATRA facilitates offline analysis by connecting to past data folders on the online cluster.
- **Real-time and Offline Visualization:** Both real-time and offline data can be visualized in various formats, including train view, waterfall view, and intensity plots.
- **Data Navigation:** A tree view helps users navigate the key-value structure of incoming data (Python dictionary type) to select the relevant detector data.

6. Deployment and First Experiment:

Following successful implementation, FATRA's backend and GUI were installed at XFEL, paving the way for its first real-world application:

- **Experiment Integration:** FATRA was successfully used in the Megahertz imaging experiment ([5157] Megahertz imaging of liquid-bubble-particle multi-time scale interaction dynamics in ultrasonic fields, Main proposer: Prof. Dr. Jiawei Mi, April 2024)

1.5. The structure and architecture of FATRA app

A well-defined software architecture plays a critical role in application performance. After identifying bottlenecks in the demo version, a key focus was establishing efficient communication channels between the backend, frontend, and separate online windows.

Figure 1 illustrates the communication architecture:

- **Main Communication Channel:** A central channel exists between the frontend and backend, utilizing ZeroMQ's "Request-Reply" pattern.
- **Adding a Backend:** When a user adds a backend component, the frontend sends a "Add Backend" request to the backend.
- **Backend Processing:** The backend receives the request, creates a new detector Karabo client instance, and establishes two separate communication channels:
    - **ZMQ Reply Channel:** This dedicated channel facilitates communication back to the frontend with processed data.
    - **ZMQ Publish Channel:** This channel allows newly created detector windows on the frontend to subscribe and receive streamed data.
- **Data Flow:** Upon receiving data, the detector Karabo backend streams it via the ZMQ publish channel. The frontend retrieves the data, processes it, and visualizes it accordingly.

FATRA Communication Architecture

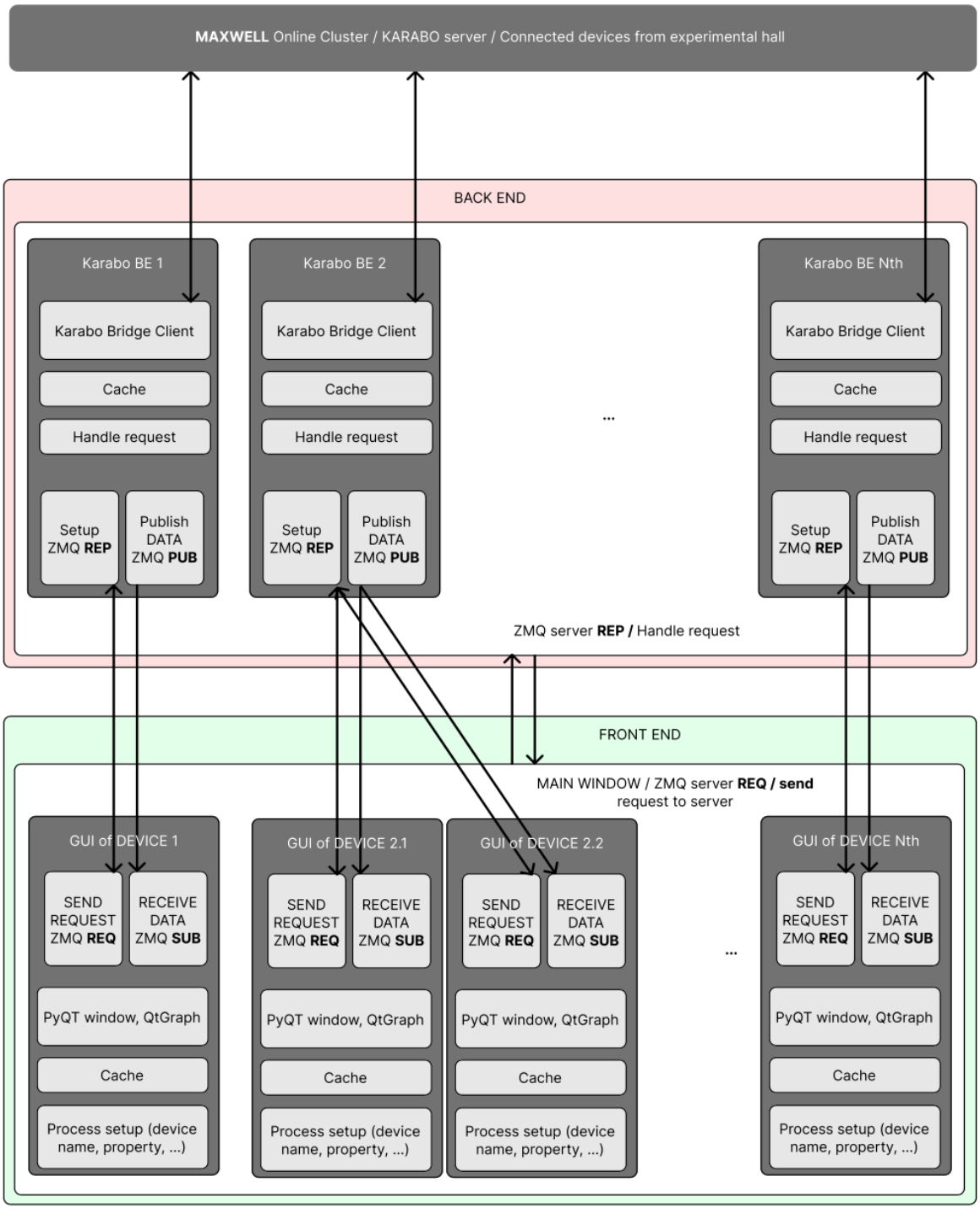

Fig. 1. The FATRA communication architecture which was used for development and is practically the base of data catching, streaming inside the fatra and visualizing in the GUI.

1.6. The end-users

An end user can be anyone who needs to review online or offline data transmitted via data streams using ZMQ. FATRA's applications extend beyond EU XFEL experiments, offering a wider range of use cases.

1.7. The name and logo

The FATRA logo, for the Fast Train Review Application developed by the University of Pavol Jozef Šafárik in Košice, Slovakia, combines its technical function with a nod to its origin. The rotated letter "F" resembles a mountain peak, reflecting Slovakia's mountainous landscape, especially the mountains called Fatra[1]. The red section represents the application's backend, and the grey edges symbolize the data stream flowing from experimental hall sensors to the data analysis system.

Fig. 2. FATRA logo

1.8. Python packages and frameworks used in the software

Used open-source python libraries.

1. **numpy**: https://numpy.org/devdocs/user/index.html#user NumPy is an open-source numerical computing library for Python.[2]
2. **karabo_bridge**: Karabo Bridge is part of the European XFEL software suite, an open-source Python library for accessing saved data produced at European XFEL. Documentation[3]

[1] https://en.wikipedia.org/wiki/Ve%C4%BEk%C3%A1_Fatra_National_Park
[2] https://numpy.org/devdocs/user/index.html#user
[3] https://github.com/European-XFEL/karabo-bridge-py#how-to-use

3. **extra_data**: Extra-data is a Python library for accessing saved data produced at European XFEL.[4]
4. **pyqtgraph**: PyQtGraph is an open-source scientific graphics and GUI library for Python.[5]
5. **opencv-python**: OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library which we use for image and video processing.[6]
6. **PyQt5**: Qt is a full development framework with tools designed to streamline the creation of applications and user interfaces for desktop, embedded, and mobile platforms.[7]
7. **pyinstaller**: Pyinstaller bundles a Python application and all its dependencies into a single package. The user can run the packaged app without installing a Python interpreter or any modules.[8]
8. **coverage**: check the code coverage.[9]
9. **flake8**: Python linter for checking the code style.[10]

1.9. Installation and Usage

The installation and usage guide is placed at the project's gittlab repository in README.md[11].

1.10. Issues, limitations and areas that require improvement

For future improvement, we propose to integrate several image correction algorithms into the FATRA GUI. These algorithms could include flat-field correction, dark-field correction, and iterative reconstruction to address diffraction artifacts in the frames.

[4] https://extra-data.readthedocs.io/en/latest/index.html
[5] https://pyqtgraph.readthedocs.io/en/latest/
[6] https://docs.opencv.org/4.8.0/d1/dfb/intro.html
[7] https://doc.qt.io/qt-5/
[8] https://pyinstaller.org/en/stable/
[9] https://coverage.readthedocs.io/en/7.4.0/
[10] https://flake8.pycqa.org/en/latest/#
[11] https://git.xfel.eu/pszeles/fatra/-/blob/main/README.md

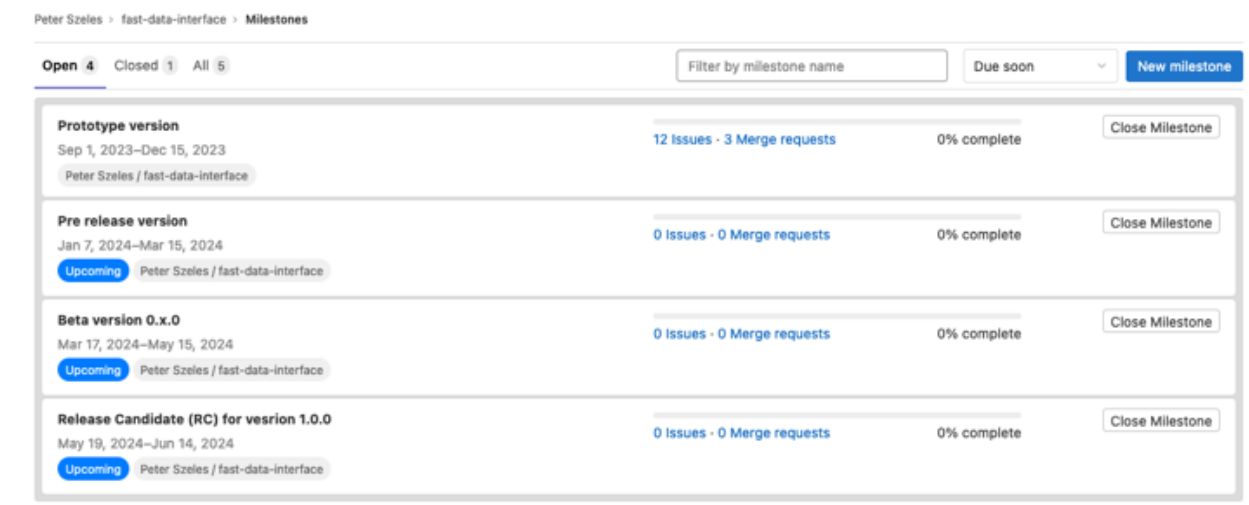

Fig. 3. Gitlab Milestones for SW Developlent.

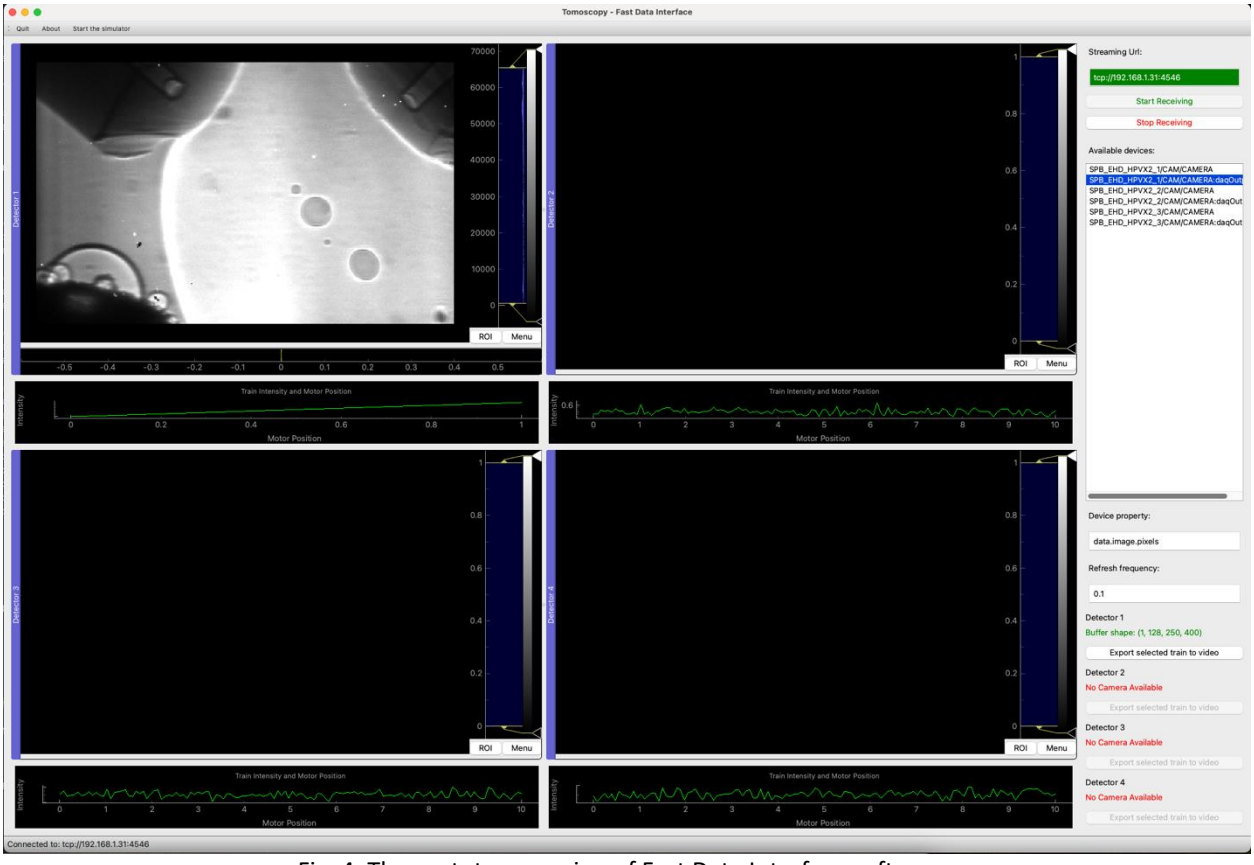

Fig. 4. The prototype version of Fast Data Interface software.

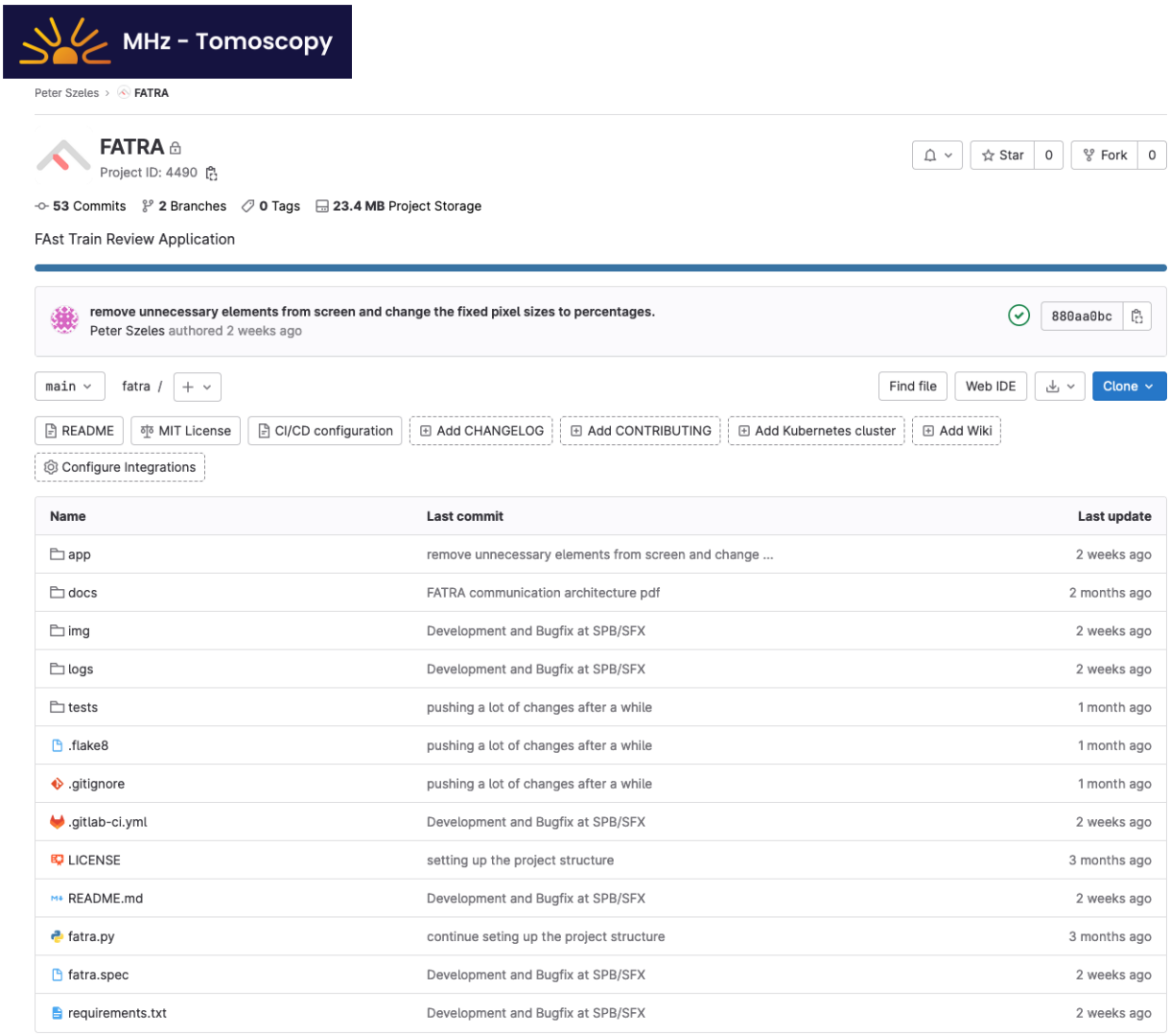

Fig. 5. The Gitlab repository of the FATRA software.

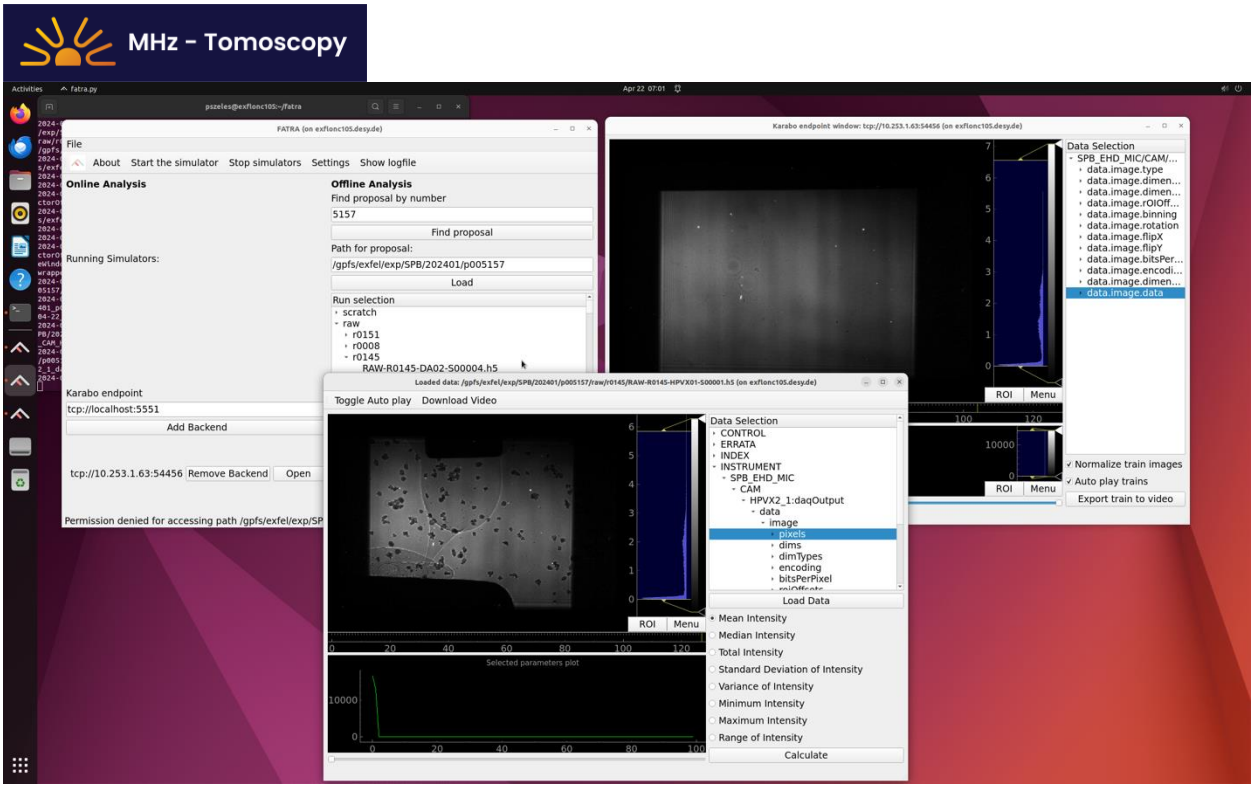

Fig. 6. FATRA GUI running on Online Cluster.
The main window (left), the Online detector window (right) and the oflline analysis window (middle).

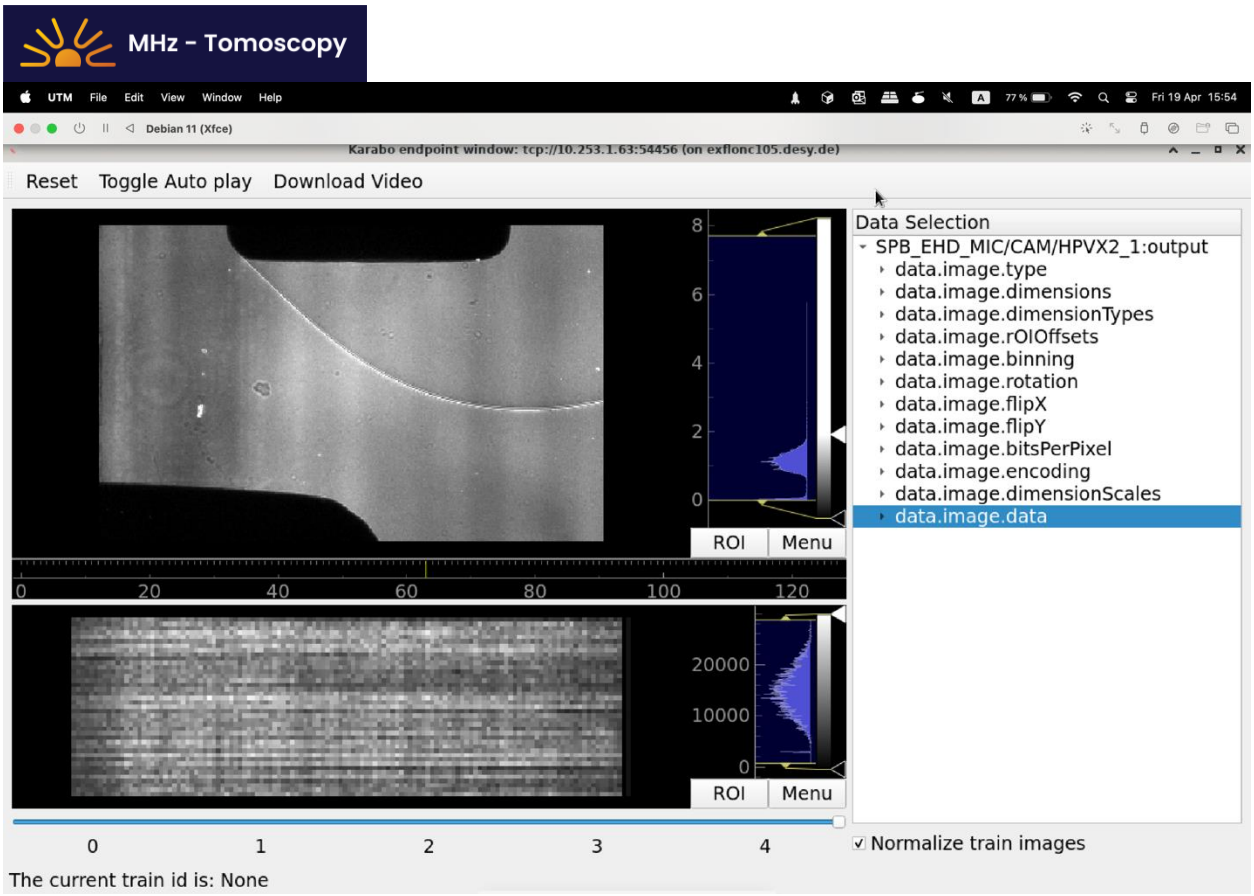

Fig. 7. FATRA GUI Online Detector window at experiment 5157

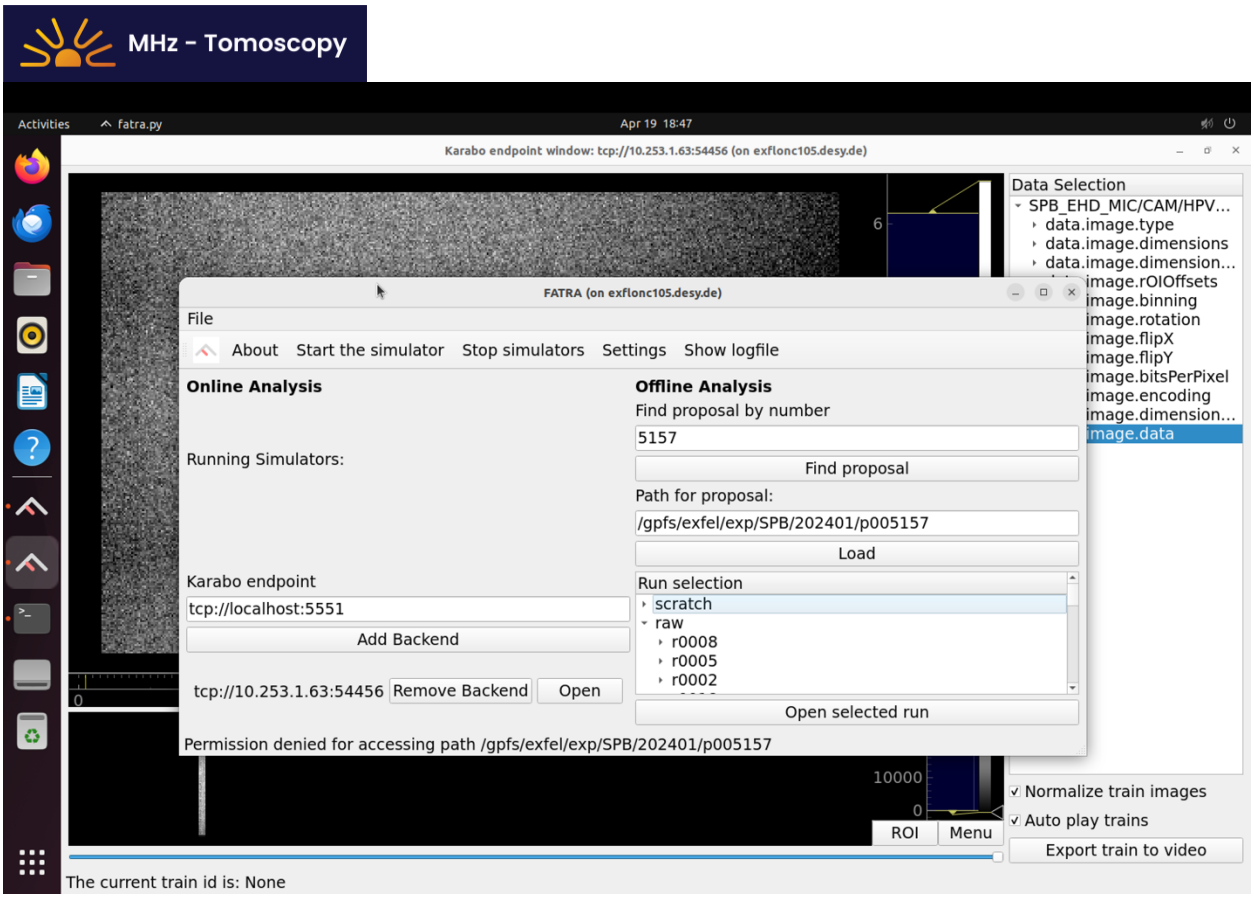

Fig. 8. The FATRA GUI main window.

On the left side at the main window is the online analysis section showing thet there is one running backennd on port tcp://20.253.1.53:54456 and on right side is the actual proposal's folder tre view.