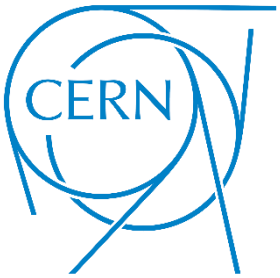

ATLAS Trigger Operations Notification System

ATLAS High Level Trigger

Author

Jane Daou

Summer Student Project 2025

Supervisors

Stefanie Morgenstern (CERN)

Benjamin Kerridge (RAL)

Aleksandra Poreba (CERN)

Table of Contents

1 Introduction

1.1 Background

The Large Hadron Collider (LHC) collides protons at 40 MHz, producing about 80 terabytes of data per second, while only around 8 GB/s can be stored [1]. To manage this, ATLAS uses a two-level trigger system: Level 1 (L1) and High-Level Trigger (HLT), that filters events in real time to retain only the most relevant ones for offline analysis.

Monitoring the trigger system is therefore crucial. Key operational quantities such as trigger rates (number of accepted events per second), deadtime (fraction of time when triggers are not active), luminosity (collision rate per unit area), available HLT processing slots (computational resources for event selection), and stream (logical collection of events grouped together according to their trigger selections or physics content), directly influence data taking efficiency and physics performance. For instance, excessive deadtime lowers the live fraction of triggers used for physics analysis, while unexpected fluctuations in the trigger rate or bandwidth may indicate misconfiguration or detector issues. Real-time visibility of these quantities is therefore essential for the Trigger Operations team to ensure smooth and efficient data taking.

1.2 Motivation

Although the ATLAS Trigger Operations system is highly optimized, its performance must still be continuously monitored and adjusted during data taking, requiring experts to have rapid access to key metrics in order to detect and respond to anomalies. Existing web-based dashboards such as Grafana PBeast provide detailed information, but they are not always practical for fast retrieval.

Since Mattermost has become a widely used communications platform within Trigger Operations, integrating monitoring tools directly into this environment provides a natural way to improve efficiency of operational decision-making. A lightweight, chat-based tool that allows experts to query values of selected metrics and receive automatic alerts within their communication channels will reduce response time to identify abnormal detector behavior, improve situational awareness, and minimize the need to switch between multiple monitoring platforms.

1.3 Problem Statement

At present, there is no integrated system that combines the flexibility of on-demand queries with proactive, real-time alerting inside the Trigger Operations communication environment. While Grafana itself offers webhook functionality that could, in principle,

deliver alerts, it cannot be used directly with ATLAS data due to security and access control restrictions. A dedicated bot layer is therefore required to ensure that sensitive data remains accessible only to authorized users. Furthermore, there is no mechanism to automatically notify the Trigger Operations team on the Mattermost channel when monitored quantities exceed predefined thresholds. This gap hinders operational responsiveness and increases the risk of delayed interventions.

1.4 Initial Idea

The concept of developing a notification bot for ATLAS Trigger Operations was first proposed during the Trigger Operations Workshop in January 2025 [2]. The proposed solution, referred to as `tbot`, is a Mattermost integrated bot designed to enhance both interactive monitoring and automated alerting.

2 Project Objectives

The primary objective of this project is to design and implement a lightweight, reliable, and extensible notification service that can be accessible through the Mattermost communication platform. By combining automated monitoring with interactive user-driven queries, the service aims to improve the responsiveness of the Trigger Operations team while contributing to the overall stability and efficiency of ATLAS data taking.

2.1 Automated Monitoring and Alerts

- **Periodic Checks:** The service periodically queries monitored quantities from existing ATLAS monitoring tools (e.g., Grafana [3][4]).
- **Threshold Evaluation:** Retrieved values are compared against predefined operational thresholds.
- **Alert Generation:** When thresholds are exceeded, `tbot` sends concise, automated alerts directly to a designated Mattermost channel. This ensures that anomalies are flagged, allowing experts to respond promptly.

2.2 Interactive User Queries

- **On-Demand Access:** Users can interact with `tbot` through the Mattermost channel.
- **Current Values:** Upon request, `tbot` retrieves and displays the most recent or average values of monitored quantities.
- **Operational Context:** In cases where the ATLAS detector is not in the RUNNING state, `tbot` informs the user accordingly.

2.3 Extensibility and Maintainability

- **Scalable Design:** The backend architecture is designed to support the addition of new monitored quantities with minimal modifications.
- **Configurable Thresholds:** Alert thresholds can be updated to adapt to evolving operational requirements.

3 System Design and Implementation

The system is implemented entirely in Python, leveraging the Flask framework [5] for the web service, APScheduler [6] for background task scheduling, and the Requests library [7] for Mattermost webhook integration.

3.1 Main Components

The system consists of four main layers:

- **Web Service & Command Processor:** A Flask-based application that handles user commands, formats responses, and schedules background monitoring tasks.
- **Mattermost Integration:** Enables interaction with the bot in a public channel. This layer also delivers automated alert messages when monitored metrics exceed predefined thresholds.
- **Data Retrieval Layer:** Interfaces with the Grafana monitoring service to fetch live ATLAS metrics.
- **Background Monitoring & Communication:** Continuously monitors selected metrics, compares them to predefined thresholds, and sends alerts to the Mattermost channel when necessary.

3.1.1 Main Application Loop

The Flask web service *(app.py)* runs continuously on OKD (see Section 3.1.6). Background tasks are scheduled using APScheduler, a Python library for scheduling and executing jobs at fixed time intervals, and are executed automatically according to their defined schedules. The system supports fully automated monitoring, requiring no user action for standard operation. Simultaneously, the main application is ready to process commands issued by users in the public Mattermost channel using the `tbot` trigger word (e.g., `tbot l1rate, tbot hltrate`), providing an interactive interface for on-demand queries. The channel is required to be public because outgoing webhooks in Mattermost can only be configured in public channels.

3.1.2 Mattermost Integration

- **Outgoing Webhook:** Allows users to interact with the bot by issuing commands prefixed with the `tbot` trigger word in a public Mattermost channel [8].
- **Incoming Webhook:** Sends automated alert messages when monitored metrics, such as L1 rate, exceed predefined thresholds [9].

3.1.3 Command Parsing and Execution

The logic described in Section 2.2 enables interactive metric requests and automated verification of the ATLAS RUNNING status.

- The *user_requests.py* module maintains a dictionary of available metrics and corresponding retrieval functions (*METRIC_FUNCTIONS*).

Figure 1 shows an example of a user requesting metrics while ATLAS is in the RUNNING state.

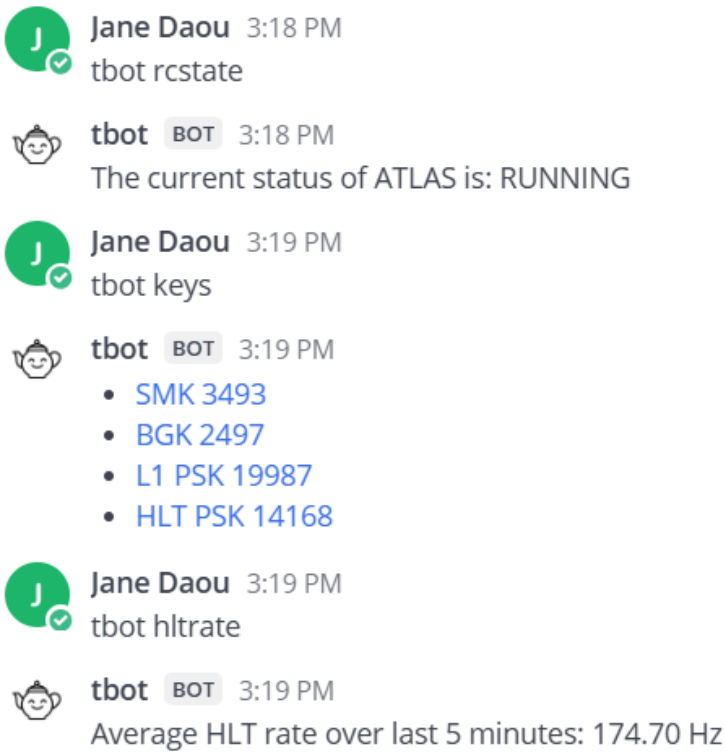

Figure 1: Screenshot of a Mattermost conversation showing a user requesting metrics while ATLAS is in the RUNNING state.

In addition, Figure 2 demonstrates a user requesting metrics while ATLAS is not in RUNNING state, with the current state displayed by `tbot`.

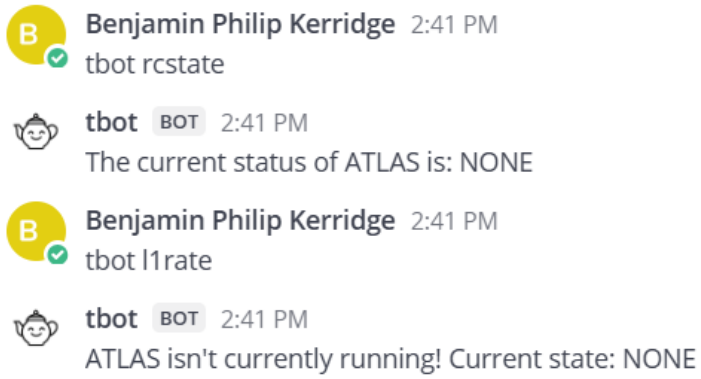

Figure 2: Screenshot of a Mattermost conversation showing a user requesting metrics while ATLAS is not RUNNING, with the current state displayed.

- Moreover, user input is parsed and validated, handling special cases such as `tbot stream <stream-name>` and `tbot stream-events <stream-name>`.

An example is shown in Figure 3 of how `tbot stream` and `tbot stream-events` are used with an additional stream name.

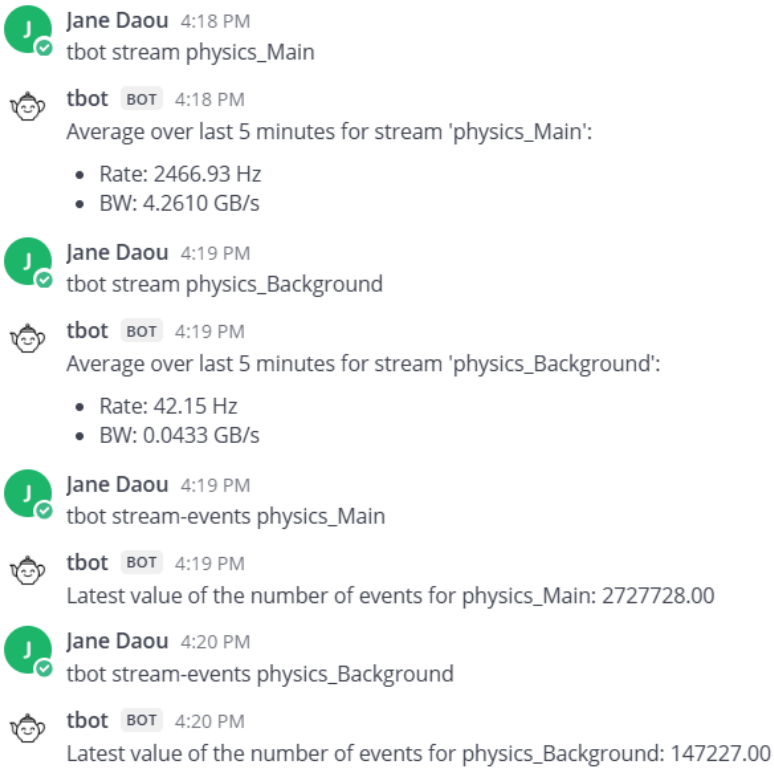

Figure 3: Screenshot of a Mattermost conversation demonstrating the use of tbot stream and tbot stream-events commands with an additional stream name.

- Responses are formatted appropriately before posting back to the Mattermost channel for different metrics available.
- Finally, the supported commands can be displayed using `tbot help`, as demonstrated in the Figure 4 below.

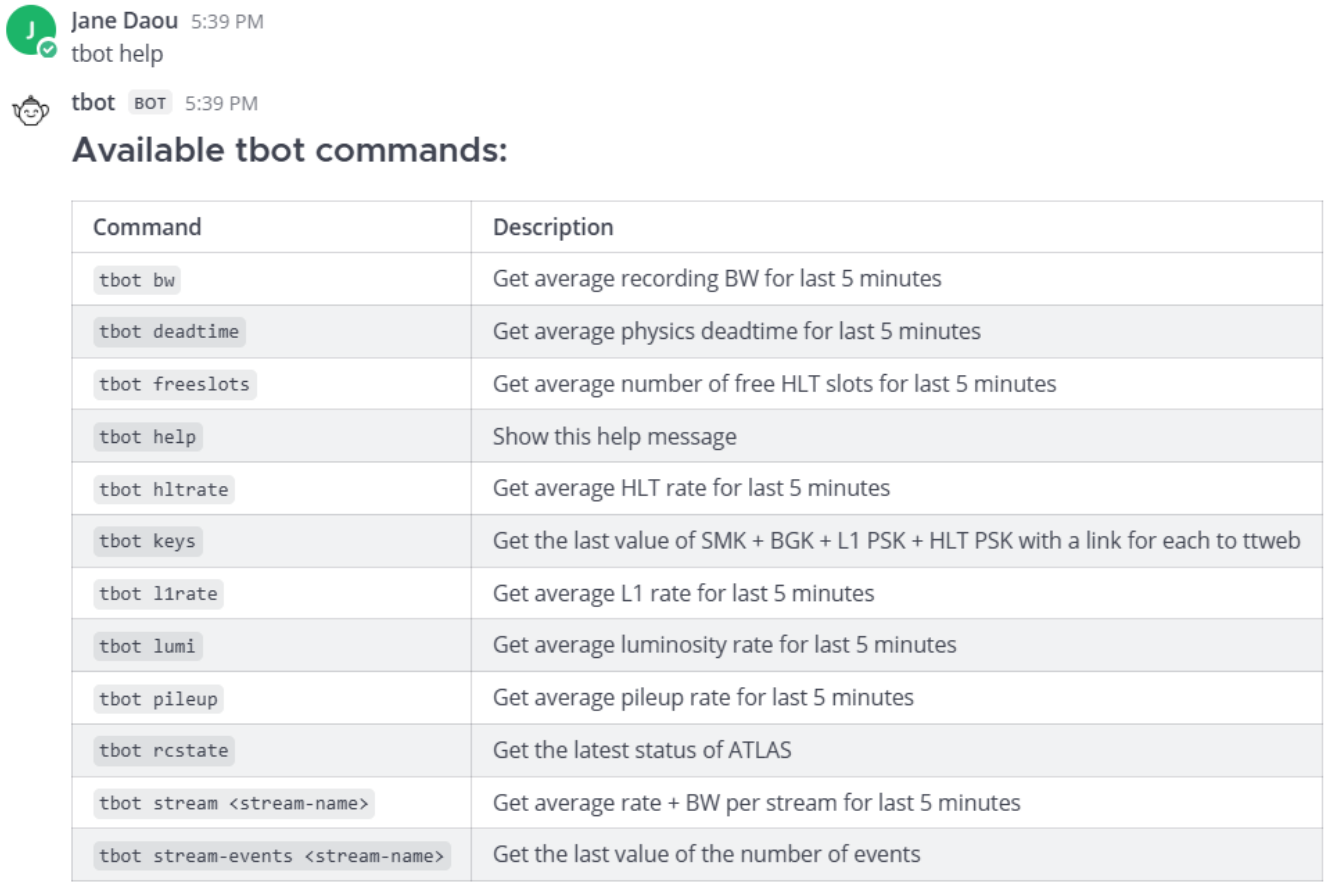

Figure 4: Screenshot of a Mattermost conversation demonstrating the tbot help command and the list of supported commands.

3.1.4 Data Retrieval Layer

The *pbeast_query.py* module interacts with the Beauty [10] service to retrieve selected ATLAS Trigger metrics.

Two generic functions are used:

- *get_latest_value_generic()* → Returns the latest value for a metric over a specified time window in days (default: 1 day).
- *get_avg_rate_generic()* → Returns the average value of a metric over the last N minutes (default: 5 minutes).

Wrapper functions are provided for all available `tbot` commands (e.g., *get_l1_rate()*, *get_hlt_rate()*, *get_recording_bw()*).

Error handling ensures missing or incomplete data does not break the system; messages are logged and optionally sent to the Mattermost channel via *send_error_notification()*.

3.1.5 Background Monitoring

Implemented in *background_monitoring.py*, the system periodically checks critical metrics (currently L1 rate) using APScheduler.

When a metric exceeds its threshold, an alert message is sent to the Mattermost channel via the incoming webhook (as shown in Figure 5).

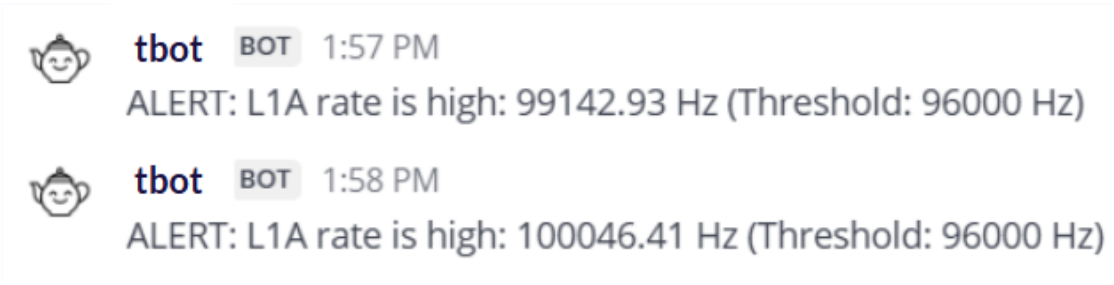

Figure 5: Screenshot of a Mattermost conversation showcasing alert messages from tbot.

This architecture is designed to be easily extensible for monitoring additional metrics in the future.

3.1.6 Deployment on OKD

The service is containerized using Docker and deployed on CERN's OKD platform [11][12],an enterprise-ready Kubernetes distribution that provides scalability, monitoring, and seamless integration with CERN's infrastructure. Flask is used as the web framework, exposing endpoints for the Mattermost outgoing webhook (*/mattermost_webhook*) and for health checks (*/*). The application is served using Gunicorn [13], a Python WSGI HTTP server that supports multiple worker processes, enabling efficient request handling and fault tolerance. OKD and Gunicorn were chosen to ensure robustness, reliability, and maintainability while aligning with CERN's standardized deployment practices.

3.2 Architecture Diagram of the `tbot` Notification System

Figure 6 illustrates the system architecture, divided into two mini systems: the automatic monitoring of a selected quantity and the handling of user requests. The main application loop and PBeast [14] readout modules serve as common elements supporting both processes.

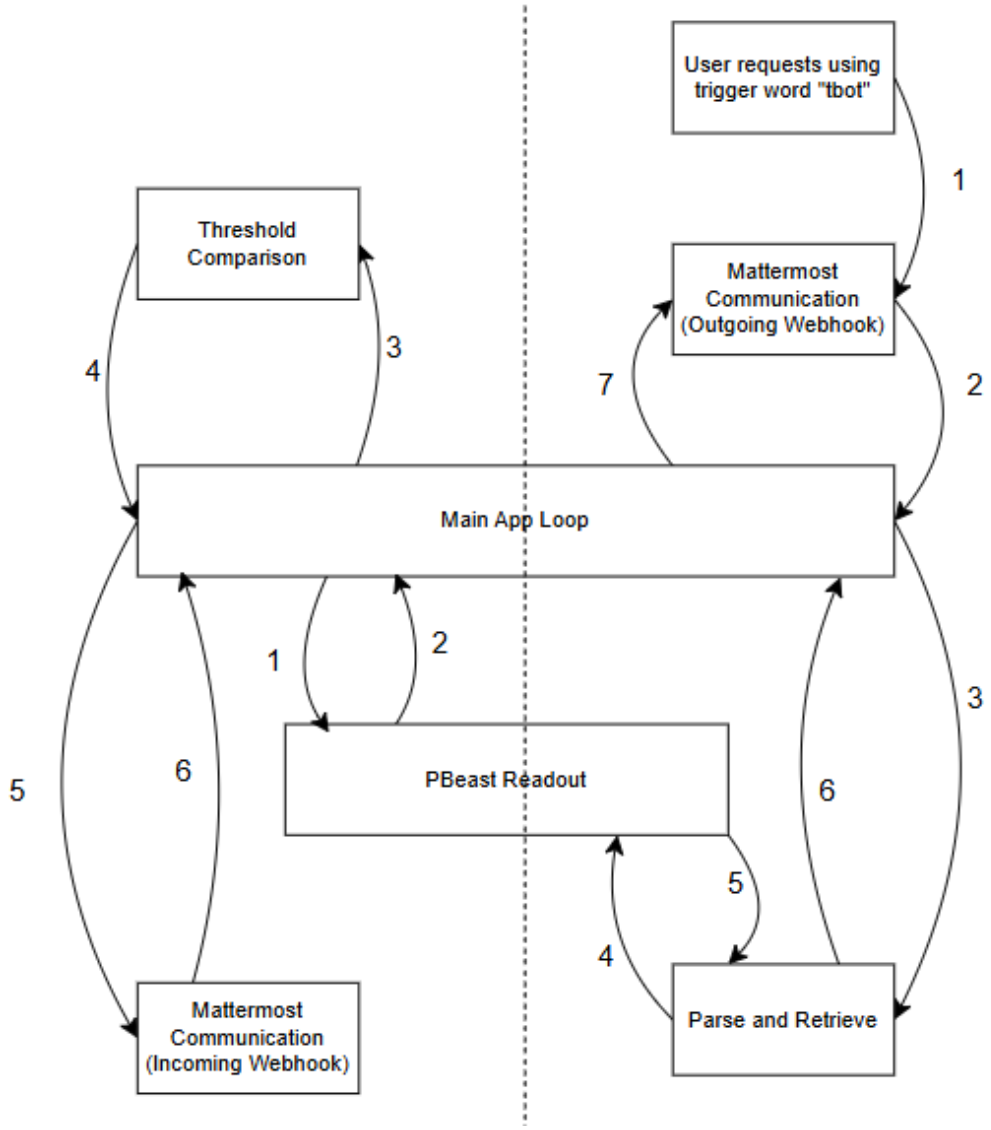

Figure 6: Diagram of the tbot Notification System where the dashed line separates the two mini systems: the automatic monitoring of a selected quantity and the handling of user requests, and the numbering represents the flow of the two different processes.

4 Current Stage of the Project

At this stage, the main components of the notification system have been successfully developed and deployed. The web service is hosted on CERN's OpenShift platform (OKD), ensuring reliable integration with existing infrastructure. A Mattermost bot, `tbot`, has been implemented to interact with users in a public channel. This interaction occurs when users explicitly request monitoring information. In addition, notifications can be received when the bot automatically sends updates resulting from background monitoring tasks.

In parallel, an automated process continuously monitors the L1 rate as a test case, retrieving and processing values at regular intervals. These results are seamlessly delivered to the Mattermost channel, reducing the need for manual checks. Finally, the main application loop and PBeast readout modules operate as central components of the system, supporting both automated monitoring and user interactions, thereby ensuring that the service remains responsive and reliable.

5 Future Work

Although the current system establishes a solid foundation, several enhancements are envisioned to expand its capabilities and improve its usability. Future developments include the expansion of monitoring beyond the L1 rate to cover additional ATLAS performance metrics, thereby offering experts a broader and more informative overview.

Another key step involves the introduction of an external JSON-based configuration file. This will allow system parameters and monitored values to be defined in a more flexible way, simplifying customization and enabling new quantities to be added without modifying the codebase.

Further improvements are also planned in alerting and visualization. The addition of configurable thresholds and richer notification options would provide more meaningful and actionable insights for users. Finally, ongoing work will focus on system optimization and scaling. Enhancing the efficiency of the background scheduler and ensuring smooth performance as new metrics and users are integrated will be central to sustaining the service over time.

6 Conclusion

The development of the ATLAS Trigger Operations notification system, `tbot`, demonstrates the feasibility of integrating automated monitoring with user-driven interactions to support ATLAS Trigger experts. The system is already deployed on OKD and capable of responding

to user requests while continuously tracking the L1 rate, providing a strong foundation for more advanced monitoring solutions. With planned extensions to additional metrics, improved configurability, and enhanced alerting, `tbot` has the potential to become a reliable and scalable tool for real-time performance insights. This work highlights both the immediate benefits already achieved and the promising opportunities for future development.

7 References

[1] S. Morgenstern, *The ATLAS Trigger System*, ATLAS Lecture Series, 19 Mar. 2024. https://indico.cern.ch/event/1366443/ (restricted)

[2] K. Maj, reporting on behalf of the Trigger Operations team, *Trigger Operations Workshop*, CERN, 17 Jan. 2025. https://indico.cern.ch/event/1487330/#4-online-dqdebug (restricted)

[3] ATLAS Collaboration, *PBeast Dashboard*, ATLAS Trigger and Data Acquisition (TDAQ) Resources, CERN. https://atlasop.cern.ch/tdaq/pbeastDashboard/dashboards (restricted)

[4] ATLAS Collaboration. *The ATLAS TDAQ system: From design to commissioning and operation (ATL-DAQ-PROC-2017-001),* 2017. https://cds.cern.ch/record/2242032/files/ATL-DAQ-PROC-2017-001.pdf (restricted)

[5] Pallets. *Flask quickstart*, 2010. https://flask.palletsprojects.com/en/stable/quickstart/

[6] A. Grönroos and Contributors. *APScheduler user guide*. Advanced Python Scheduler, 2009. https://apscheduler.readthedocs.io/en/3.x/userguide.html

[7] K. Reitz and Contributors. *Requests: HTTP for humans*. https://requests.readthedocs.io/en/latest/

[8] Mattermost. *Outgoing webhooks*, 2025. https://developers.mattermost.com/integrate/webhooks/outgoing/

[9] Mattermost. *Incoming webhooks*, 2025. https://developers.mattermost.com/integrate/webhooks/incoming/

[10] ATLAS Collaboration, *Beauty Monitoring Library (Beauty)*, ATLAS Trigger-Menu GitLab repository. https://gitlab.cern.ch/atlas-trigger-menu/beauty

[11] R. Rasool. *TOAST: TDAQ online analysis and service tool*. CERN Document Server, 2023. https://repository.cern/records/chqey-spy79/preview/TOAST_Report.pdf

[12] CERN. *CERN Platform-as-a-Service (PaaS) documentation*. https://paas.docs.cern.ch/

[13] Pallets Project, *Gunicorn deployment guide — Flask Documentation*, Deploying to Production section, Flask (Pallets Projects). https://flask.palletsprojects.com/en/stable/deploying/gunicorn/

[14] I. Soloviev *et al.*, ATLAS Operational Monitoring Data Archival and Visualization, 18 Feb. 2020.

https://cds.cern.ch/record/2710109/files/10.1051_epjconf_202024501020.pdf cds.cern.ch+1