Hydra: An AI-Based Framework for Interpretable and Portable Data Quality Monitoring

Thomas Britton[1,*], *Torri* Jeske[1], *David* Lawrence[1], *Nataliia* Matsiuk[1], and *Raiqa* Rasool[1]

[1]Thomas Jefferson National Accelerator Facility

Abstract. Hydra is an advanced framework designed for training and managing AI models for near real time data quality monitoring at Jefferson Lab. Deployed in all four experimental halls, Hydra has analyzed over 2 million images and has extended its capabilities to offline monitoring and validation. Hydra utilizes computer vision to continually analyze sets of images of monitoring plots generated 24/7 during experiments. Generally, these sets of images are produced at a rate and quantity that is exceedingly difficult for shift crews to effectively monitor. Significant effort has been devoted to enhancing Hydra's user interface, to ensure that it provides clear, actionable insights for shift workers and other users. Gradient Weighted Class Activation Maps (GradCAM) provide added interpretability, allowing users to visualize important regions of the image for classification. Hydra has been containerized to enable the creation of portable demos and seamless integration with container-based technologies such as Kubernetes and Docker. With the user interface enhancements and containerization, Hydra can be rapidly deployed for new use cases and experiments. This talk will describe the Hydra framework, its user interface and experience, and the challenges inherent in its design and deployment.

1 Online Data Quality Monitoring

Ensuring high-quality data in nuclear physics experiments requires continuous monitoring of detector performance, typically through image-based data quality monitoring plots. The sheer volume of data, coupled with varying levels of experience among shift crew members, makes effective manual monitoring challenging. Hydra addresses this issue by automating the visual monitoring process, analyzing images more rapidly and consistently than human operators could achieve.

2 Hydra

Hydra ([1], [2]) is a modular framework comprising three primary components: a backend managing image processing and model inference, a database storing data and facilitating communication, and a user interface (UI) enabling user interaction. Its modularity supports diverse deployment scenarios, from real-time online monitoring to offline calibration and analysis. It is currently deployed in all experimental halls at Jefferson Lab.

*e-mail: tbritton@jlab.org

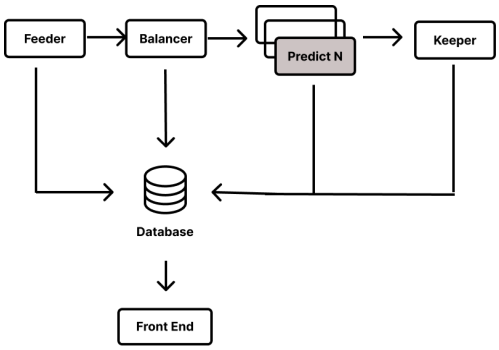

Figure 1. Schematic of Hydra's backend workflow, illustrating image preparation (Feeder), workload balancing (Balancer), inference executions (Preidict), and result-driven actions (Keeper).

2.1 Hydra's Back-end

The processes that make up Hydra's Back-end manage the inference workflow in 4 distinct steps (Fig. 1): Feeder, Balancer, Predict, Keeper. This modularity in workflow allows for more fine grained control of not only how data is handled by the system but the actions it takes.

Feeder

The backend continuously monitors input directories, resizes images to match model specifications, and generates inference orders via ZeroMQ [3], which include metadata, model IDs, and file paths. Images are recognized based on their file name. Unrecognized images are discarded, simplifying the input management.

Balancer

Inference orders are distributed across processes using a round-robin approach. This process lays the groundwork for more advanced algorithms where specialized hardware is involved on different nodes.

Predict

The inference process executes orders sequentially, generating standardized reports that support classification and regression outputs. These reports inform automated responses, such as visual and/or audible alarms.

Keeper

The final step involves recording inference results in the database, updating EPICS [4] process variables, and selecting images for future model training. Stored images include random selections for ongoing performance assessment and any classified Bad image.

2.2 Hydra's Database

The MySQL database comprises 38 tables storing extensive information, including image metadata, labels, inference outcomes, a model registry, and user permissions. This structured dataset facilitates detailed performance analysis and system-wide evaluations. Database schema integrity is maintained using yoyo-migrations [5].

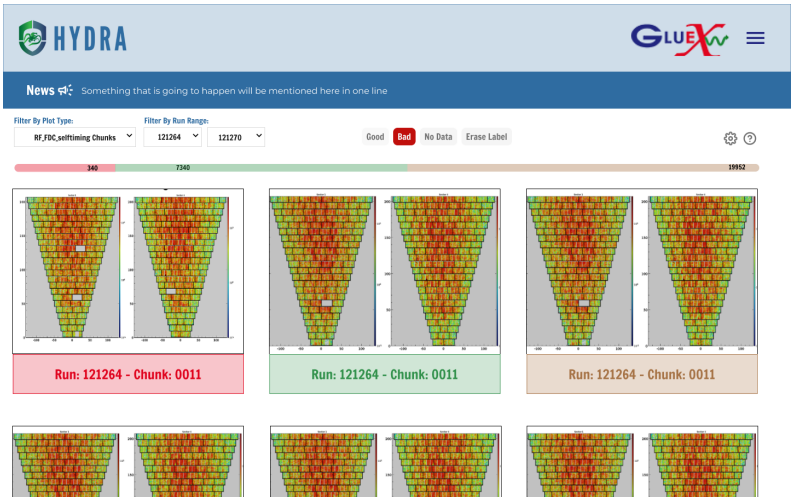

Figure 2. The Labeler enables detector experts to label individual and/or sets of images from different experiment runs efficiently. When labels are added to images and saved, any remaining images to be labeled will automatically appear.

2.3 Hydra's User Interface

The UI caters to diverse user roles such as detector experts, shift crews, and data scientists. Pages are categorized into real-time monitoring, model performance and configuration, and data labeling. UI components are developed using React [6] and the backend integration uses Flask [7], detailed further in Section 3.1.

3 Features

3.1 UI/UX

3.1.1 Image Labeler

The Labeler, shown in in Figure 2 leverages the Von Restorff effect [8], arranging similar images in grids to highlight differences. Users can efficiently label images individually or collectively, with labels stored directly in the database for training dataset assembly. Detector experts can choose specific labels, or default to the standard "Good" and "Bad" options. The stacked horizontal bar chart visualizes the distribution of labeled images.

3.1.2 Hydra Run

Accessible remotely with proper authentication, the Run page, shown in Figure 3, updates continuously, visually distinguishing real-time image classifications. Images that are classified as "Bad" are automatically moved to the "Bad" sidebar. The GradCAM heatmaps are overlayed onto the images, and aid the shift crew in identifying anomalies. Each image card includes options for sharing relevant information (run number, date/time, model outputs) with shift crews and detector experts.

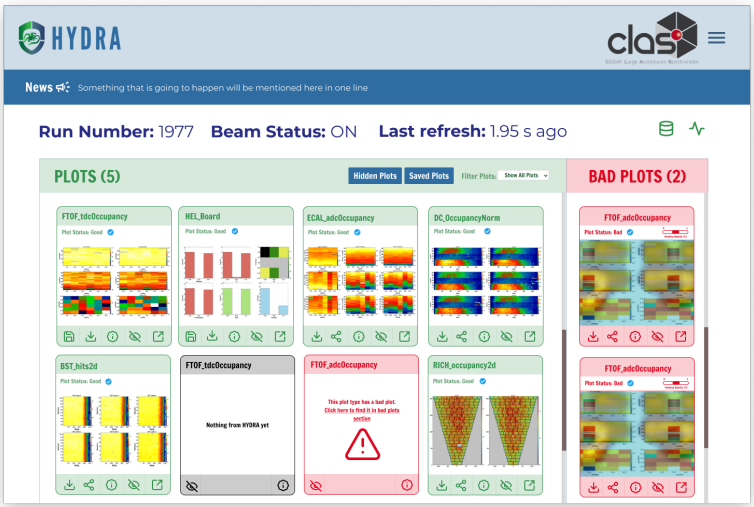

Figure 3. The Run page displays near real-time classifications for shift crews and detector experts. The main components of the page include a status bar, showing relevant information for the shift crews, with a container that separates classified "Bad" plots from the rest of the images.

Figure 4. The Model Library pages visualizes an enhanced confusion matrix, where each cell contains the distribution of output weights from the training and validation data sets. The detector experts can view performance metrics, information about the trained models, and adjust thresholds.

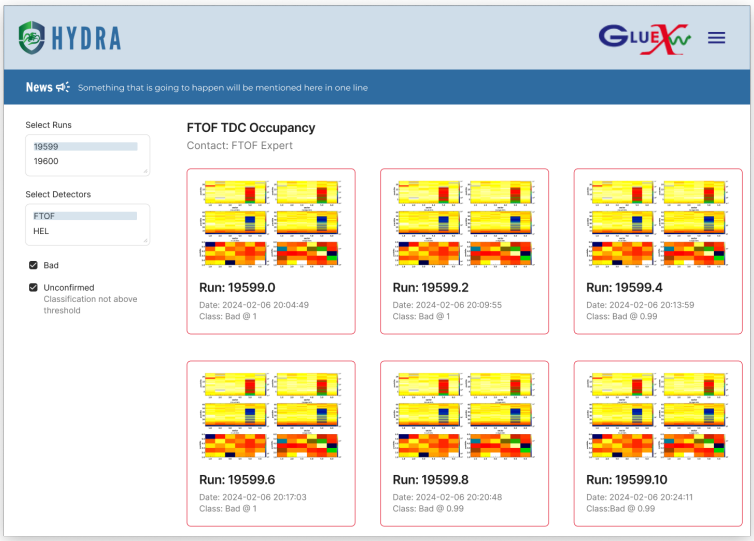

Figure 5. The Log page provides a summary of all "Bad" and unconfirmed images from a given date time. The default setting is a trailing 24 hour window.

3.1.3 Model Library

The Model Library, shown in Figure 4, provides detailed insights on model performance on both the training and validation data sets, running performance on newly labeled images, and allows users to adjust classification thresholds to minimize false alarms. The confusion matrix displays both the counts and the distribution of the model's output weights for the training and validation data sets. The classification thresholds are used in order to determine when Hydra will visually alarm. If the output weight of a given classification is below (above) the defined threshold, the image is considered *unconfirmed* (*confirmed*). Hydra visually alarms on confirmed Bad images.

3.1.4 Log

The Log page, shown in Figure 5, collates images that were classified as either "Bad" or unconfirmed within a configurable timeframe, simplifying issue review for users.

3.1.5 Status

The status page (not shown), tracks backend system health, performance metrics, and operational statistics to aid administrators and developers with system maintenance.

3.1.6 Grafana Interface

The Grafana dashboard displays model prediction weights over time, with horizontal dashed lines indicating active classification thresholds. Analyzing trends in these predictions can signal when retraining a model may be necessary or highlight emerging issues, such as component failures.

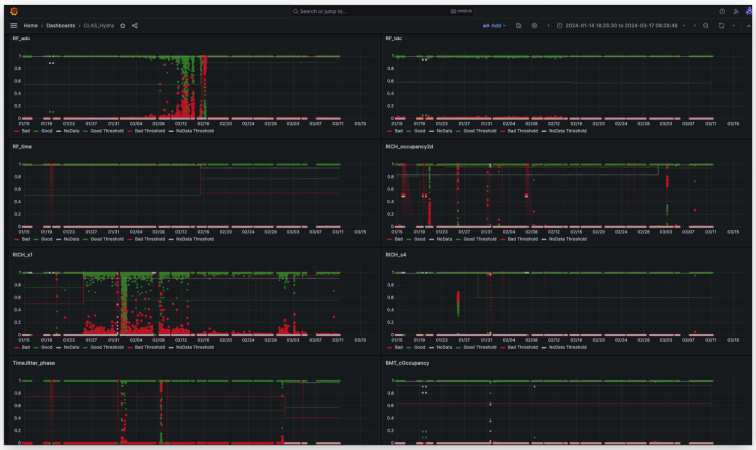

Figure 6. The Grafana dashboard visualizes model predictions for each image over time by plotting output weights for each inference. Different colors represent classification types (e.g., green for "Good," red for "Bad"), while dashed lines indicate the threshold values for each classification. The top classification for each inference is determined by the highest output weight. The top-left panel illustrates how a shift from predominantly "Good" (green) to "Bad" (red) classifications clearly signals the onset and subsequent resolution of a component failure.

3.2 Interpretability

Hydra leverages Convolutional Block Attention Modules (CBAM) [9] to enhance both performance and interpretability in convolutional neural network architectures. Applying attention mechanisms across the channel and spatial dimensions refines the internal feature maps and places greater emphasis on the most relevant regions of an image. For visualization, Gradient-weighted Class Activation Maps (GradCAM) [10] provide a straightforward way to visualize the regions that most strongly influence the model's output. In Hydra, these heat maps are generated and displayed for images classified as Bad. In an offline setting, the heat maps are used to validate and interpret model behavior. Together, CBAM and GradCAM aid shift crews in identifying and resolving data quality issues. An example of this is shown in Fig. 7, where the heat maps are overlayed on the images in the second column. For the "Bad" image, the heat maps highlight the two extra dead tiles, while correctly ignoring the known dead tile present in the "Normal" image. Although identifying anomalies may seem straightforward, shift crews routinely monitor thousands of histograms per run. The heat maps make it significantly easier for the shift crew to identify dead tiles, enabling faster and more consistent responses during data-taking.

4 Containerized Hydra

Hydra was containerized to improve its portability, enhance development and testing efficiency, and increase scalability. Packing the application and its dependencies inside a container ensures consistent behavior across different environments, streamlines the deployment process, and ensures effective management of resources. Two containers were developed for demonstration and development.

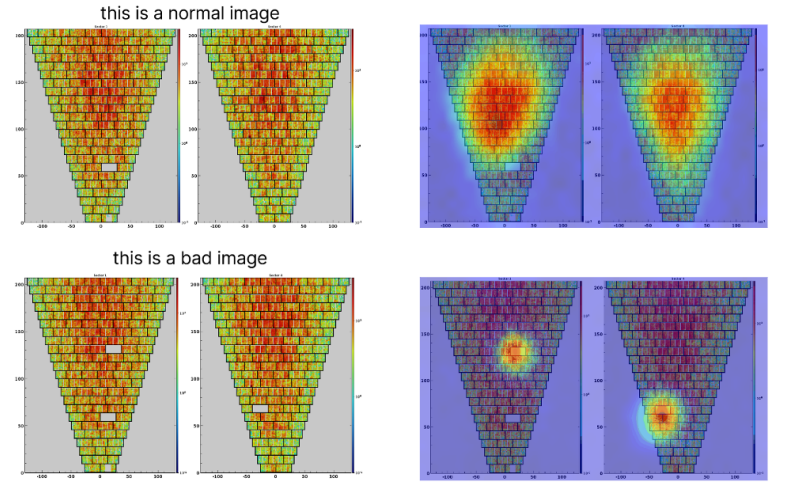

Figure 7. The first row depicts a "Normal" monitoring image and the associated GradCAM heat map overlay, while the second row depicts a "Bad" monitoring image and the associated GradCAM heat map overlay. For the "Normal" image, the heat map generally highlights the regions where there is data. For the "Bad" image, the heat map clearly indicates the extra two dead tiles, and correctly ignores the known dead tile in the lower left region of the image.

4.1 Demonstrations

Hydra can be deployed for demonstrations in a containerized environment and accessed locally once the container has been pulled from Dockerhub. Within the container, Hydra's core functionalities, such as labeling, training, analysis, and back-end operations, can be utilized to train models and process a pre-loaded set of images. Users can access the same UI for monitoring via local host.

4.2 Development

To facilitate a standardized development process, a second container called *HydraShell* was created. Its goal is to provide a flexible, safe, and uniform environment for developing Hydra's components. The container can be configured to run only a specific part of the system—such as the front end, back end, or database—allowing developers to focus on individual components and carry out routine tasks consistently. Running *HydraShell* without any specifications automatically clones the main branches of Hydra's three GitHub repositories into the container. It also populates the database with a small development dataset, providing a convenient start point for new features.

5 Conclusions

Hydra is a modular and extensible framework design to train, deploy, and manage computer vision models for near real-time data quality monitoring in nuclear physics experiments. It automates the full inference workflow: detecting new images, preparing them for analysis, running inference with trained models, and taking appropriate actions based on the results. The system is supported by a robust MySQL database and a user-friendly web interface that

enables users to monitor results, label data, review detector health, and manage models, all without requiring deep knowledge of the underlying machine learning infrastructure. To further support interpretability, Hydra incorporates visual explanations that help users understand how the models are making decisions. Hydra is distributed in two containerized environments: a demonstration container that allows users to explore its capabilities locally, and a flexible development container offering a consistent, self-contained environment for contributing new features. These containers support future scaling efforts, including deployment in Kubernetes clusters. Hydra's future enhancements focus on new feature development, optimization for scalability, and expanding use cases, such as advanced image segmentation, multi-modal (image, annotations, numeric data) analysis, and unsupervised techniques.

6 Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177.

References

[1] T. Jeske, T. Britton, D. Lawrence, and K. Rajput, "Hydra: Computer Vision for Online Data Quality Monitoring," *EPJ Web of Conf.*, vol. 295, pp. 02008, 2024. DOI: 10.1051/epjconf/202429502008.

[2] T. Britton, D. Lawrence, and K. Rajput, "AI Enabled Data Quality Monitoring," *EPJ Web of Conf.*, vol. 251, pp. 04010, 2021. DOI: 10.1051/epjconf/202125104010.

[3] ZeroMQ, *ZeroMQ*. Available: https://zeromq.org/.

[4] M. Clausen and L. Dalesio, "EPICS: Experimental physics and industrial control system," *ICFA Beam Dyn. Newslett.*, vol. 47, pp. 56–66, 2008.

[5] yoyo-migrations, https://pypi.org/project/yoyo-migrations/.

[6] React, https://react.dev/.

[7] Flask, https://flask.palletsprojects.com/.

[8] Von Restorff, H., "Über die Wirkung von Bereichsbildungen im Spurenfeld (The effects of field formation in the trace field)", *Psychol. Forchung*, vol. 18, pp. 299–342, 1933, doi: 10.1007/BF02409636.

[9] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional Block Attention Module," *CoRR*, vol. abs/1807.06521, 2018. Available: http://arxiv.org/abs/1807.06521.

[10] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization," *CoRR*, vol. abs/1610.02391, 2016. Available: http://arxiv.org/abs/1610.02391.