The Data Fusion Labeler (dFL): Challenges and Solutions to Data Harmonization, Labeling, and Provenance in Fusion Energy

Craig Michoski[a,d,*], Matthew Waller[a], Brian Sammuli[b], Zeyu Li[b], Tapan Ganatma Nakkina[a], Raffi Nazikian[b], Sterling Smith[b], David Orozco[b], Dongyang Kuang[a], Martin Foltin[c], Erik Olofsson[b], Mike Fredrickson[a], Jerry Louis-Jeune[a], David R. Hatch[a,d], Todd A. Oliver[a,d], Mitchell Clark[b], Steph-Yves Louis[a]

[a]Sophelio, Austin, TX USA
[b]General Atomics, San Diego, CA, USA
[c]Hewlett Packard Enterprise, Palo Alto, CA, USA
[d]University of Texas, Austin, TX, USA

Abstract

Fusion energy research increasingly depends on the ability to integrate heterogeneous, multimodal datasets from high-resolution diagnostics, control systems, and multiscale simulations. The sheer volume and complexity of these datasets demand the development of new tools capable of systematically harmonizing and extracting knowledge across diverse modalities. The **Data Fusion Labeler** (dFL) is introduced as a unified workflow instrument that performs uncertainty-aware data harmonization, schema-compliant data fusion, and provenance-rich manual and automated labeling at scale. By embedding alignment, normalization, and labeling within a reproducible, operator-order-aware framework, dFL reduces time-to-analysis by greater than 50X (e.g., enabling >200 shots/hour to be consistently labeled rather than a handful per day), enhances label (and subsequently training) quality, and enables cross-device comparability. Case studies from DIII-D demonstrate its application to automated ELM detection and confinement regime classification, illustrating its potential as a core component of data-driven discovery, model validation, and real-time control in future burning plasma devices.

Keywords: Fusion Energy, Data Fusion, Data Harmonization, Data Provenance, Machine learning, Data labeling, Multimodal data

1. Introduction

Fusion energy devices and their high-fidelity simulations now produce petabyte-scale data streams spanning up to nine orders of magnitude in time (from microseconds to tens of seconds) and covering spatial scales from the ion gyroradius ($\rho_i \sim 1\,\mathrm{mm} - 1\,\mathrm{cm}$) to the full device ($R \sim 1 - 10\,\mathrm{m}$). These streams arise from heterogeneous sources, such as kilohertz profile diagnostics (Thomson scattering, charge-exchange recombination spectroscopy), megahertz magnetic coils and fast cameras in the edge/SOL [1], neutron and bolometry systems, real-time control telemetry, and synthetic diagnostics embedded in extended MHD and gyrokinetic codes [2, 3]. Managing and interpreting such multimodal data motivates dedicated tools like the **Data Fusion Labeler** (dFL) (viz. Figure 1), which provides harmonization, fusion, and provenance tracking across this diverse landscape.

While extreme in scale, these challenges are not unique to fusion. Comparable issues arise in domains such as brain–computer interfaces (BCI), manufacturing, robotics, biomedical data, finance, weather forecasting, marketing analytics, etc., where multimodal, asynchronous, and noisy time-series data must be aligned, fused, and interpreted under uncertainty. In all such contexts, the central barriers include the following:

- **Multimodality and heterogeneity:** diverse data sources often operate at different sampling rates, in different units, and with distinct metadata conventions, making cross-signal comparisons non-trivial. This is common in domains ranging from multimodal medical imaging to financial

---

time series, and in fusion research arises from disparate diagnostics such as magnetics, Thomson scattering, and neutron detection [4–9].

- **Imbalanced and noisy datasets:** rare but high-impact events—whether seizures in EEG, extreme weather events, fraud in transaction streams, or plasma disruptions in fusion devices [10–12]—occur against a backdrop of far more common stable conditions. Noise and uncertainty from calibration errors, environmental interference, or diagnostic degradation further complicate inference [13, 14].

- **Sparse and incomplete labeling:** expert-generated annotations are costly and often incomplete, forcing workflows into semi- or weakly-supervised regimes. This holds across many fields—clinical medicine, finance, or plasma physics—where the volume of unlabeled data vastly exceeds curated subsets [15–18].

- **Dynamic and nonstationary behavior:** the underlying processes evolve over time, often with changing statistics and regime shifts. Examples include market volatility, neural adaptation, climate variability, or plasma instabilities and control interventions in fusion [19–22].

- **Integration of multiscale simulations and models:** many domains must reconcile simulations or models that span different spatial or temporal scales and mathematical representations, whether in weather forecasting, structural engineering, or multiscale plasma simulations from MHD to gyrokinetics [23, 24].

In fusion energy these barriers are often compounded by temporal imbalances (megahertz magnetics versus kilohertz profiles), missing data [25], and the need for principled handling of imbalanced classes without distorting statistical calibration in hazard-function and survival modeling [26, 27]. Collectively, they mirror the "long tail" of problems in other data-rich sciences: sparse critical events, heterogeneous modalities, dynamic systems, and costly human labels.

By contrast, the inherent alignment of photographic, video, and textual data has allowed those modalities to flourish within diffusion and transformer architectures, accelerating progress across computer vision and natural language processing. The time-series modalities that dominate fusion and other physical sciences lack this natural alignment, and thus have not yet fully benefited from comparable advances. The **Data Fusion Labeler** (dFL) (outlined in Figure 1) is designed to close this gap, bringing the harmonization, fusion, and structured labeling required to elevate time-series data to the same precipice of advancement now transforming other fields.

Converting these diverse, asynchronous, and noise-burdened signals into plasma physics insight, reliable control signals, and design constraints hinges on three interlocking pillars:

I. **Data harmonization** [28]: systematic curation, standardization, alignment, and uncertainty-aware representation that render disparate diagnostics and model outputs *commensurable* in units, coordinates, sampling, and metadata;

II. **Data fusion** [29]: the mathematically principled combination of harmonized sources to produce estimates and decisions with lower bias, tighter credible intervals, and improved robustness relative to any constituent alone; and

III. **Data provenance** [30]: complete, machine-actionable records of origin, processing history, and versioning that enable reproducibility, auditability, and FAIR [31] reuse across devices and campaigns [31, 32].

In magnetic confinement experiments, harmonization can be exemplified by the sub-millisecond synchronization of edge fast-camera movies with Mirnov coil signals, a prerequisite for uncovering causal relationships between visible turbulence structures and magnetic perturbations. Data fusion can be illustrated by combining Thomson scattering and charge-exchange recombination spectroscopy, whose complementary sensitivities to electrons and ions enable construction of self-consistent pressure and rotation profiles unattainable from either diagnostic alone. Data provenance is captured by preserving the exact EFIT version, coil set, and preprocessing filters used in equilibrium reconstruction, since even minor differences in these choices can lead to divergent q-profiles and stability assessments.
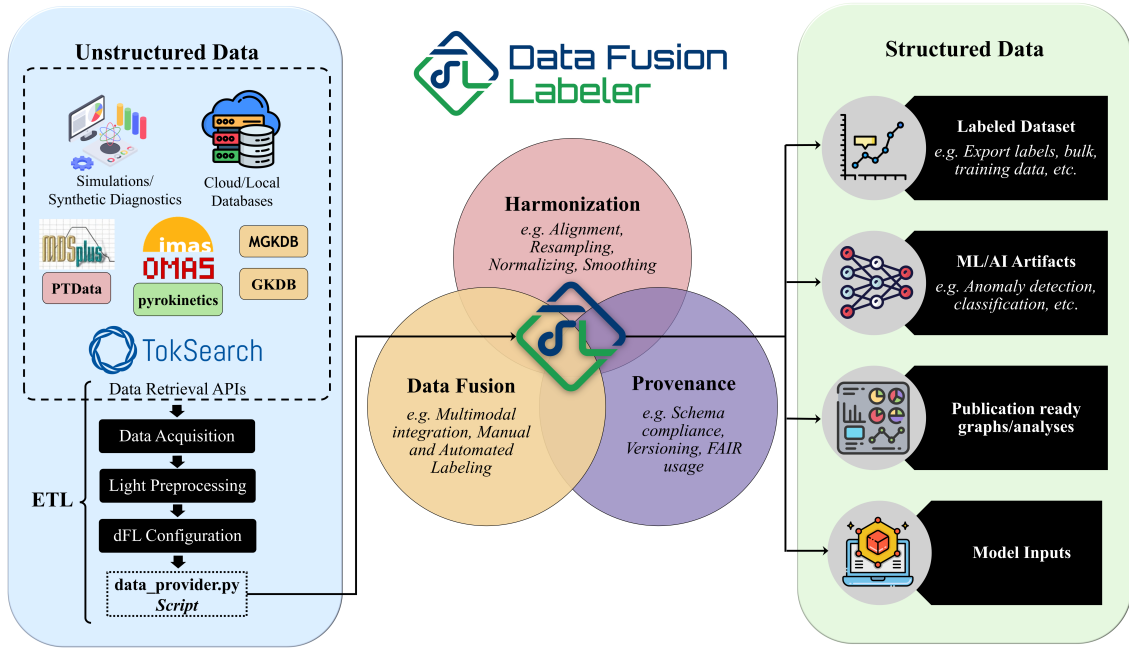
Figure 1: Overview of the Data Fusion Labeler (dFL) workflow.

In data engineering terms, harmonization and data fusion extend the classical ETL paradigm [33]: harmonization can be viewed as a physics-aware specialization of "Transform," ensuring cross-diagnostic commensurability, while data fusion operates downstream of ETL, synthesizing harmonized inputs into higher-level constructs with reduced uncertainty and enhanced robustness. This perspective is consistent with the Joint Directors of Laboratories (JDL) hierarchy, which treats source preprocessing and alignment as "Level 0" [34]—a *prerequisite* for higher-level estimation, identification, and decision-making [35].

In magnetic confinement fusion, this principle is operationalized by the ITER-driven *Integrated Modelling & Analysis Suite (IMAS)*, which provides a machine-independent schema for experimental and synthetic data, enabling toolchains to interoperate without bespoke converters [36]. Workflow environments such as OMFIT then leverage IMAS/IDS structures to compose equilibrium reconstruction, transport solvers, stability analysis, and turbulence models in a single, provenance-aware graph [37].

Harmonization stands as the foundational step before any attempt at learning or control. Without it, downstream analyses risk being dominated by idiosyncrasies—unit drift, timestamp skew, schema mismatches, signal polarity inversions, undocumented calibration changes, diagnostic preprocessing artifacts, or missing metadata—rather than by the underlying physics. The study of edge turbulence and pedestal dynamics, for example, demands sub-millisecond temporal registration between fast imaging and magnetic probes to uncover true causal structure [1]. Similarly, the integration of gyrokinetic surrogates such as TGLF or GENE outputs with profile diagnostics requires consistent geometry, coordinate conventions, and uncertainty representations [2, 38]. Harmonization is therefore not a superficial preprocessing task; it is an *information-preserving filter* that renders multi-diagnostic inference and control both tractable and defensible.

With commensurability established, *data fusion* methods ranging from classical Bayesian estimators, to modern representation-learning techniques, to sophisticated automated labeling and feature extraction can be brought to bear. By exploiting complementary sensitivities across diagnostics, these methods reduce variance, resolve ambiguities, enhance robustness to noise and diagnostic failure, and improve reliability under non-ideal conditions. In disruption prediction, for instance, the fusion of magnetic, radiative, and density signals has outperformed single-channel predictors, enabling generalization across a range of operational scenarios [39]. Similar benefits are observed in event detection tasks such as identifying ELMs, sawteeth, or ITB formation, where combining profile evolution, fluctuation spectra, and actuator traces leads to sharper recall and precision.

Underlying both data harmonization and data fusion is the necessity of rigorous data provenance. The

reproducibility of scientific claims and the portability of trained models depend on transparent lineage: knowing what data were used, how they were transformed, by whom, with which code and parameters, and against which schema version. The FAIR principles codify these requirements [31], and the iterative nature of campaign analysis demands that labels, features, and models be versioned alongside the raw signals. Without such practices, reported performance cannot be independently verified, nor can models be deployed with confidence as devices and diagnostics evolve [32].

In this work, we introduce the **Data Fusion Labeler** (dFL) as a unifying workflow instrument that sits at the interface between data harmonization, data fusion, and data provenance. dFL (i) ingests multi-diagnostic and multimodal experimental and synthetic data in (optionally) IMAS/OMAS-compatible layouts [36, 37, 40] processed through uncertainty aware alignment; (ii) performs filling, resampling, normalization, smoothing, and visualization at multiple resolutions; (iii) supports manual, statistical, physics-informed, simulation-driven, and classifier-based *labeling and autolabeling* to convert raw streams into structured event corpora; and (iv) captures complete, machine-readable provenance for every label and export, enabling exact replay and cross-facility reuse. In practice, dFL reduces time-to-analysis from weeks to hours, turns ad-hoc scripts into documented transformations, and converts fragile, person-specific workflows into durable, shareable artifacts. By standardizing label semantics and embedding them in harmonized, provenance-rich contexts, dFL directly addresses the data scarcity and class-imbalance challenges that pervade disruption prediction, confinement-regime identification, and MHD mode classification [39, 41].

This paper formalizes the architectural requirements for label-centric fusion workflows and presents the design of dFL to meet them. We (1) articulate harmonization constraints induced by fusion physics and diagnostic practice; (2) detail a provenance model aligned with FAIR principles; (3) describe manual and automated labeling modalities, including statistics-based change detection, physics-informed rules, simulation-driven autolabeling, and supervised classifiers; and (4) demonstrate that rigorous harmonization and provenance multiply the physics value extracted by downstream analysis, modeling, and control. Throughout, we ground the discussion in fusion-specific literature and tools (IMAS/OMFIT, turbulence and transport models, high-speed edge diagnostics) [1, 2, 36–38], while also highlighting analogies to other high-stakes data-rich fields (BCI, finance, weather), emphasizing that the challenges and solutions embodied in dFL are broadly relevant.

2. Challenges of Data Fusion in Fusion Energy Data

Fusion energy research faces an especially demanding set of challenges when it comes to the fusion of heterogeneous data sources. Tokamaks and stellarators rely on a diverse set of diagnostics, operational systems, and simulation frameworks, producing massive multimodal datasets that must be harmonized, fused, and labeled under uncertainty. While many of these issues—such as multimodality, imbalance, noise, sparse labeling, and nonstationarity—are also encountered in other data-intensive fields (e.g., medicine, finance, or weather forecasting), they appear in fusion at extreme scale and with unique physical constraints. Addressing them is not optional: they determine the reliability of physics inference, the safety of machine operation, and the reproducibility of results across campaigns and devices.

2.1. Multimodal, Heterogeneous, Imbalanced, Noisy Datasets

In fusion energy research, particularly in the operation and diagnostics of tokamaks and other fusion plasma devices, data from various sources are collected to monitor the system's behavior and ensure safe and efficient operations. These datasets are inherently multimodal, arising from the combination of diagnostic instruments such as magnetic coils [4], Thomson scattering [5], and neutron detectors [6], operational data like plasma parameters [7] and external heating systems [8], as well as simulation and model data in areas like model predictive control (MPC) [9]. These data sources often differ in their resolution, format, and sampling frequency, leading to a heterogeneous dataset that must be reconciled for comprehensive analysis, and especially for ML/AI data-enabled pipelines. Additionally, due to the dynamic and high-energy environment in a tokamak, the data can exhibit high dimensionality, with a wide range of variables describing the plasma state, control systems, and performance metrics. The integration of such diverse data types presents significant challenges for modeling and analysis, requiring sophisticated algorithms capable of handling multiple modalities simultaneously.

Data fusion has emerged as a pivotal strategy to address these challenges by combining and integrating information from diverse sources into a unified, enriched dataset. This process facilitates the resolution

of heterogeneity by aligning and harmonizing data formats, resolutions, and sampling frequencies across modalities. For example, data fusion can effectively reconcile high-frequency magnetic sensor data with low-frequency Thomson scattering measurements, enabling synchronized and comprehensive analysis. By leveraging advanced methods, such as multimodal deep learning [42] and Bayesian frameworks [43], researchers can unify disparate data types, enhancing the interpretability of complex plasma behavior and improving the robustness of downstream models.

A particularly pressing, though often under-emphasized issue in fusion energy research is the prevalence of imbalanced data. Many fusion diagnostics and operational datasets exhibit significant disparities in the frequency and distribution of events of interest. For instance, high-disruption scenarios in tokamaks [10]—critical for understanding and preventing damage to plasma-facing components—occur far less frequently than stable operating conditions. Similarly, certain anomalous behaviors, such as neoclassical tearing modes [11] or edge-localized modes [12], are relatively rare (or sparsely contained in the diagnostic data stream) but hold immense importance for both scientific understanding and operational safety. This imbalance skews the datasets, resulting in challenges for predictive modeling, as standard machine learning algorithms often become biased toward the majority class, failing to adequately capture rare but critical phenomena. The imbalance also complicates the selection of appropriate evaluation metrics, requiring the use of metrics like precision, recall, and F1-score [44] to ensure meaningful model assessment.

Data fusion techniques are particularly advantageous in addressing imbalanced datasets. By integrating data from multiple diagnostics, fusion methodologies can amplify the representation of underrepresented events through the synthesis of complementary information. For instance, correlating rare disruptions captured in magnetic diagnostics with soft X-ray observations can enhance the detection and characterization of such events, mitigating the effects of imbalance. Additionally, data fusion supports robust feature extraction across modalities, which can improve the sensitivity of models to rare but critical phenomena.

Fusion energy datasets are further compounded by temporal imbalances, where data collected at different timescales contribute unevenly to the overall dataset. For example, while magnetic diagnostics may sample at megahertz frequencies [1], other diagnostics, such as Thomson scattering systems, operate at much lower frequencies. This discrepancy can lead to an over-representation of certain data modalities in modeling processes, while others are under utilized despite their potential importance.

Additionally, fusion datasets often contain significant noise and/or uncertainties [13, 14] arising from a variety of sources, including sensor calibration errors, electromagnetic interference, and signal degradation over time. Noise can obscure critical patterns, particularly in diagnostics requiring high sensitivity, such as those detecting soft X-rays or fast ion populations. Distinguishing meaningful signals from noisy data is particularly challenging in imbalanced datasets, as rare events are more likely to be masked by random fluctuations or measurement errors. Missing values—whether due to diagnostic malfunctions, misalignments in data collection, or physical limitations of the instrumentation—further exacerbate these issues, introducing gaps that must be addressed through imputation [25] or alternative strategies.

To address these challenges, the fusion data community has increasingly focused on developing robust data handling and analysis pipelines. Techniques such as synthetic oversampling (e.g., SMOTE) [45] and statistically robust data resampling methodologies [46] are being explored to mitigate the impact of class imbalance. Advanced data fusion techniques, such as multimodal deep learning [42] and Bayesian frameworks [43], are being employed to reconcile heterogeneous datasets, enabling the extraction of meaningful insights across disparate diagnostic systems. These methods not only resolve heterogeneity but also bolster the ability to manage noisy and imbalanced data effectively. Additionally, methods for denoising data, including wavelet transforms [47, 48] and advanced filtering techniques [15], are becoming essential for preprocessing, particularly in high-noise environments. However, it is important to note that not all imbalance should be corrected through oversampling or reweighting. In some cases—particularly those that are explicitly statistical in their workflow—preserving the original distribution is essential for maintaining statistical calibration of model outputs. For example, in hazard function learning, naive rebalancing can distort the interpretation of survival probabilities unless special precautions are taken [26, 27]. More broadly, there are differing views on whether imbalance should always be "fixed," with some arguing that careful metric selection and calibration can be preferable to data-level interventions.

2.2. Sparse and Incomplete Data Labeling in Fusion Diagnostics

In addition to the challenges posed by heterogeneous and noisy data, another significant issue in fusion energy research is the sparse and incomplete nature of data labeling [15]. Many fusion diagnostics, especially those employed in tokamaks, operate in real-time and are designed to monitor a variety of plasma parameters over extended periods. However, annotating this data with meaningful labels and metatags—such as classifying plasma stability modes, identifying anomalous events, or categorizing different operational regimes—is often sparse due to the complexity of the plasma behavior and the high cost of manual labeling. In many cases, only a small fraction of the collected data is annotated, leading to the problem of semi-supervised or unsupervised learning, where models must make predictions with limited labeled data. This challenge is compounded by the fact that many labels are incomplete, especially when diagnostics fail or data gaps occur due to technical issues or extreme plasma conditions. To overcome this, advanced techniques in active learning [16], transfer learning [17], and weak supervision [18] are required to efficiently handle sparse and incomplete labels while maintaining model accuracy.

2.3. Dynamic and Nonstationary Nature of Fusion Devices

Fusion systems, particularly tokamaks, exhibit highly dynamic and nonstationary behavior, further complicating data analysis. The physical processes within a tokamak [19, 20], such as plasma instabilities, heating, and magnetic confinement, evolve over time and are influenced by a wide range of external and internal factors. These processes often do not follow fixed statistical distributions or patterns, and the underlying system behavior can change rapidly in response to operational adjustments, perturbations, or environmental influences. As a result, fusion data are often nonstationary, meaning that statistical properties such as mean, variance, and correlations can vary over time. This nonstationarity can lead to difficulties in developing predictive models that generalize well across different operational phases or fusion events. Moreover, the dynamic nature of the system means that models must continuously adapt to changing conditions, requiring the use of online learning techniques [21] or adaptive algorithms [22] capable of handling time-varying data. The challenge of capturing both short-term fluctuations and long-term trends in such dynamic environments necessitates sophisticated modeling techniques, such as recurrent neural networks (RNNs), autoregressive approaches, or other time-series forecasting methods, which are designed to accommodate the evolving characteristics of the system.

2.4. Multiscale Simulation Data in Fusion Energy

Another significant challenge in fusion energy research arises from the integration of simulation data generated by multiscale pipelines, which model fusion systems at different spatial and temporal resolutions [23, 24]. These simulation models span a broad hierarchy: large-scale macroscopic descriptions of plasma dynamics, such as magnetohydrodynamic (MHD) models, reduced and extended MHD formulations that capture two-fluid or Hall physics [49–51], and kinetic models that resolve particle distribution functions in velocity space to address turbulence, fast-ion dynamics, and wave-particle interactions [52, 53]. In addition, multiscale efforts extend beyond core plasma physics to encompass kinetic-material interaction models for plasma-material interfaces, plasma facing components (PFCs), and edge/SOL dynamics [54, 55], as well as coupled models for subsystems such as fueling, exhaust, and impurity transport [56, 57].

The data produced at each scale often resides in distinct numerical or mathematical representations: structured or unstructured grids for continuum MHD and fluid models, marker distributions or velocity-space discretization for kinetic solvers, and surface or mesh-based data for material response simulations. Moreover, the geometries employed can vary widely; global core plasma models may impose cylindrical or toroidal symmetry, while localized edge or wall-interaction simulations require realistic three-dimensional geometry with surface roughness or material microstructure resolved.

This variability in representations complicates integration across scales and hinders the derivation of comprehensive insights from the full simulation hierarchy. Bridging these gaps requires the development of robust data fusion techniques that can: (i) map information across scales and between disparate numerical representations, (ii) reconcile differences in resolution, uncertainty, and accuracy, and (iii) incorporate both experimental and simulated data streams to improve predictive capability. Such harmonized workflows are essential not only for theory-experiment comparison, but also for developing reduced models suitable for control and decision-making in real-time tokamak operation.

3. dFL System Overview

The **Data Fusion Labeler** (dFL) provides a modular, extensible framework for ingesting, harmonizing/fusing, and labeling multimodal fusion energy datasets, as shown graphically in Figure 1. Data ingestion is managed through the `data_provider` script, which serves as the primary interface between raw archives and the dFL functionalities. This mechanism enables users to construct *custom ingestion pipelines*, including tailored graphing and analysis tools, custom independent variable formats, user-defined normalization schemes, specialized smoothers, customized labels and autolabelers, or domain-specific filters and feature maps. For low-level access, the `Fetch Data` functionality allows users to override default ingestion and implement arbitrary preprocessing pipelines prior to rendering signals in the dFL interface, ensuring maximal flexibility when integrating novel diagnostics or experimental workflows. The full `data coordinator` API, and extensive examples and demos are provided in the dFL documentation

Beyond extensibility, dFL ships with native support for a full suite of preprocessing operations required for fusion data pipelines, including resampling, normalization, smoothing, trimming, interpolation, etc. These operators are integrated into the graphical interface, ensuring that users can both prototype and deploy harmonization strategies without leaving the dFL environment. All operations can be made fully provenance-tracked, guaranteeing that every transformation—whether native or user-defined—is recorded for reproducibility and downstream audit.

In practice, this architecture allows dFL to serve as both a turnkey environment and a development platform: users can rely on the included preprocessing library for common workflows, or seamlessly inject custom operators at the ingestion level to meet specialized needs. This dual capability makes dFL well-suited for experimental campaigns, simulation archives, and cross-device data fusion tasks, where flexibility, reproducibility, and performance are equally essential.

4. Harmonization & Data Fusion

In fusion energy research, particularly for tokamak experiments and simulations, data harmonization is often undervalued compared to direct physics analysis. Yet it is foundational: without rigorous harmonization, the quality, accuracy, reproducibility, and physical coherence of downstream results are compromised. By enforcing consistency across heterogeneous diagnostics and simulation outputs, harmonization transforms disparate signals into a common, physically meaningful framework suitable for reliable inference and control. Equally, harmonization is the prerequisite for *data fusion*: once signals are co-registered and rendered commensurable, one can combine information across diagnostics in statistically principled ways

Nomenclature Key for DIII-D TokSearch Point Names

DIII-D Signal Name	Physical Measurement Description
`ip`	Plasma current, representing the total toroidal current flowing through the confined plasma (typically in megaamperes, MA).
`magnetics amplitudes`	Amplitudes of toroidal mode components (n) of the poloidal magnetic field (B_p), obtained from `modespec`/`modespyec` analysis of toroidally distributed magnetic probes; typically expressed in Gauss or mT.
`pinj`	Total neutral beam injection (NBI) power delivered to the plasma, summed over all active injectors (in megawatts, MW).
`wnt`	Average of the measured electron density and temperature pedestal width, with the unit in ψ_N.
`tinj`	Neutral beam injection torque, quantifying the total angular momentum input to the plasma from the NBI system (in N m).
`echpwrc`	Electron cyclotron heating (ECH) power coupled to the plasma, as measured by the real-time ECH power controller (in MW).
`betan`	Normalized plasma beta, $\beta_N = \langle\beta\rangle/(I_p/aB_T)$, denoting the ratio of plasma to magnetic pressure normalized to plasma current and magnetic field.
`density`	Line-averaged electron density from interferometry diagnostics (in units of 10^{19} m^{-3}).
`Pedestal ratio`	Ratio of the measured pedestal width to the conventional scaling law based on KBM estimation [58], serving as an indicator of pedestal structure and stability.
`fs0*`	Filterscope channels measuring line-integrated photon flux from D$_\alpha$ and impurity emissions near the plasma edge (in photons s^{-1} cm^{-2} sr^{-1}).

(e.g., inverse-variance weighting, information-form Kalman updates, and multi-output Gaussian-process co-kriging) to produce estimates with reduced posterior variance relative to any single channel.

Key harmonization tasks include synchronizing signals in time and space, reducing datasets to relevant intervals, addressing gaps or missing values, and applying resampling, smoothing, and normalization to ensure comparability without erasing important physics. The sequencing of these operations is critical, as many are non-commutative and can yield different results if applied in different orders. A principled harmonization workflow preserves essential information, minimizes artifacts, and prepares data for advanced analysis, modeling, and multi-device data fusion. In practice, harmonization defines the likelihood terms and error models that downstream fusion methods rely on, e.g., transforming raw measurements into a common state-space with known noise covariance enables Bayesian fusion [59].

In fact—as is already well-established in other data-heavy engineering and scientific disciplines—the single most pragmatic lever to accelerate learning, prediction, and control in fusion-energy science resides not in ever more elaborate model architectures, but rather in improving the *information substrate* upon which those models operate. When heterogeneous diagnostics and simulation outputs are harmonized and fused into a coherent representation, one effectively drives the underlying state-space manifold to *lower informational entropy*—yielding a dataset that is more compact, better aligned with the physics, and less burdened by spurious variation. Symbolically:

$$\text{Better Data} \Rightarrow \text{Lower Entropy Manifold} \Rightarrow \text{Simpler Model} \Rightarrow \text{Higher Generalizability.}$$

By reducing the variance and bias introduced by mis-synchronization, inconsistent units, missing provenance, or unlabeled modes of operation—and by enforcing a physically consistent *operator ordering* in harmonization pipelines, where trimming, filling, resampling, smoothing, and normalization are applied in a reproducible and non-commutative sequence—one ensures that transformations preserve causal and phase relationships among signals. This disciplined treatment of operator composition transforms harmonization from a cosmetic preprocessing step into a mathematically coherent filter, one that preserves invariants of the underlying dynamics and prevents artificial distortion of cross-diagnostic correlations. In doing so, even modest modeling approaches can generalize across campaign, device, and regime boundaries with greater stability and interpretability. Empirical evidence from multimodal fusion studies confirms that improvements in input coherence and alignment frequently yield greater downstream gains than incremental model refinement or neural lift alone [60–62]. In the fusion context, this realization underscores why the architecture of the Data Fusion Labeler (dFL) is focused first on harmonization and fusion, rather than pushing the envelope of operator complexity.

The dFL workflow is designed to provide users with the ability to design and customize their own harmonization strategies. While a large number of native operations are already fully features, dFL is designed so that custom harmonization and visualization strategies can be easily and readily integrated into the existing dFL GUI by simple python scripting. As a consequence, the harmonization functions included in Sections 4.1.3–4.4 should be thought of as options, not restrictions. Some simple examples of implementing custom harmonization operations are shown in the included appendices, and extensive examples are detailed in the dFL documentation. In addition, dFL support the ability to customize the `Fetch Data` option, that allows users to perform any preprocessing steps they like before loading the data into the dFL GUI using the same simple `data coordinator` API (additional details and examples of this ingestion-level customization approach can be found in Appendix D and are extensively detailed in the dFL documentation). Beyond harmonization, dFLâĂŹs labeling and export stages can be used to preserve per-signal uncertainties and masks so that downstream *feature-, score-, or decision-level data fusion* (e.g., stacking [63], Dempster-Shafer combinations [64], or log-odds aggregation [65]) can correctly weight sources and propagate uncertainty [66].

4.1. Basic Data Feature Support

This subsection outlines the essential data handling features in dFL, such as dynamic visualization-driven resampling, data filling, and data trimming, that together ensure data alignment, consistency, and clarity across diverse fusion datasets. It also emphasizes the importance of interactive navigation tools, which allow users to zoom, label, and explore the data at different resolutions without compromising accuracy or performance. These steps further determine the statistical structure (e.g., sampling operators, effective bandwidths, and masks) on which multi-rate data fusion and cross-diagnostic inference depend.

4.1.1. Data Alignment

Data alignment is a foundational prerequisite for any meaningful integration of multimodal fusion datasets. In both experimental and simulation contexts, diagnostics operate on disparate temporal grids, spatial coordinate systems, and reference frames, often with differing latencies and acquisition rates. Without precise alignment—both temporally and spatially—comparisons between diagnostics can introduce phase errors, distort cross-correlations, and obscure causal relationships between measured quantities. In downstream ML/AI or model-based workflows, such misalignments can propagate as systematic biases, degrading predictive accuracy and interpretability. Robust alignment procedures, therefore, are not merely a convenience but a requirement for preserving the physical coherence of the data, enabling valid feature extraction, and supporting reliable fusion of heterogeneous sources into a unified analytic framework. From a formal data fusion perspective, alignment supplies a common state $x(t)$ and observation maps $y_i(t_i) = H_i x(t) + \epsilon_i$ with known $t \mapsto t_i$; without this, likelihoods across i cannot be coherently multiplied nor can cross-covariances be estimated.

As a rule, dFL generally deals with data alignment at the data ingestion level, ensuring that temporal and spatial indices from disparate sources are reconciled as early as possible in the pipeline. By enforcing alignment at ingestion, dFL prevents the accumulation of phase mismatches and indexing errors that would otherwise compound through downstream preprocessing steps such as smoothing, normalization, or feature extraction. However, even with such safeguards, certain *instrumental effects*—most notably digitizer drift, ADC baseline wander, or timing jitter—can subtly distort the apparent temporal coherence between diagnostics. These drifts, arising from hardware instabilities rather than physical dynamics, manifest as slow offsets or clock skews that shift relative signal phases over the duration of a discharge. Because their magnitude and character depend on device-specific electronics and environmental conditions, they typically require *case-by-case calibration or detrending*, often through baseline correction or reference-pulse tracking. dFL treats these as calibration anomalies external to the alignment framework itself but provides
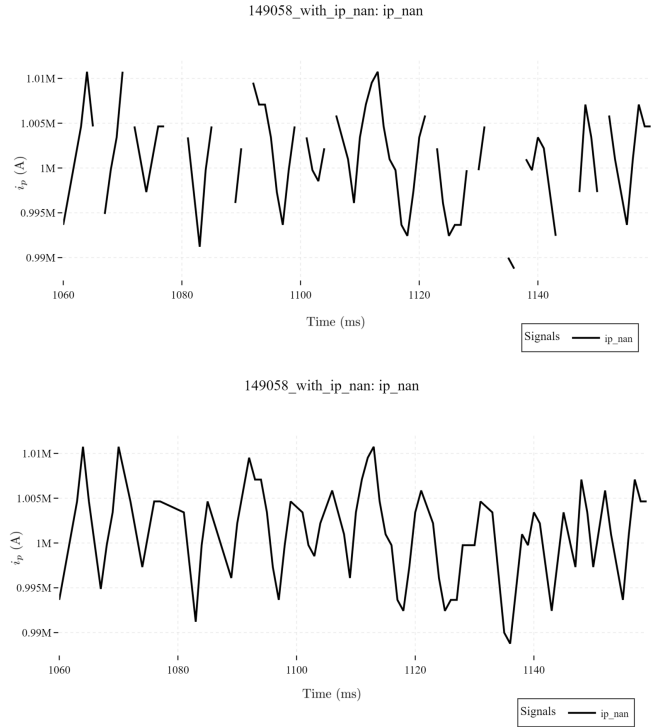

Figure 2: An image showing the fill capability on a signal with NaNs scattered throughout. The top image is the NaN-riddled plasma current i_p signal before fill, and the bottom image is after dFL fill is applied.

mechanisms for their metadata tagging, ingestion realignment, and/or post-hoc correction. This layered approach ensures that subsequent operations (whether for visualization, model training, or real-time control) operate on datasets that are both temporally reconciled and physically trustworthy, preserving the statistical integrity and interpretability of the fused result.

It is also worth noting that dFL can handle raw pulse-counting data via utilizing custom filters/graphs in `fetch data` (within the `data provider`). That is, `fetch data` can be used to ingest list-mode or pre-binned counts with live time, apply dead-time/pile-up corrections, convert to rates with uncertainties, optionally variance-stabilize, align by bin edges, and use Poisson-aware/heteroscedastic losses. Nevertheless care must be taken when utilizing dFL's built-in operations on raw pulse-counting data, as several default operators assume fixed-grid, roughly constant-variance noise, where treating counts as continuous (e.g., generic smoothing or z-scores) can bias both rates as well as fusion weights.

The data presented in this paper integrates with TokSearch [67–69], a high-performance toolkit developed for parallel access to heterogeneous fusion data. TokSearch is a high-performance query and retrieval engine integrated into dFL as a backend data provider, which is widely used to prepare experimental and simulation archives for AI/ML workflows. Within dFL, it provides the data access

backbone, ensuring that labeling operates on consistently aligned inputs (as the data has been pre-processed for alignment in TokSearch). The important and subtle topic of physical units and unit-handling in dFL is discussed in some detail in Appendix I, and the additional alignment considerations from resampling are addressed in Section 4.2.

4.1.2. Data Trimming

When large heterogeneous diagnostics (e.g. such as magnetic probes at 2 MHz, Thomson scattering at a kilohertz, synthetic turbulence fields on adaptive meshes, etc.) are loaded *en masse*, the cardinality of samples (N) quickly exceeds what a browser, a GPU, or even a high-bandwidth PCIe bus can ingest without stalling. A judicious *trim*, applied *before* gap filling or resampling, collapses the problem from $\mathcal{O}(N)$ I/O and memory traffic to $\mathcal{O}(n)$, where $n \ll N$ is the subset aligned with the scientific question at hand. Trimming can also be utilized to localize data fusion to regimes where the underlying process is, e.g., approximately stationary, improving the validity of stationary noise assumptions in weighted least squares and spectral fusion, etc.

Conceptually, trimming is nothing more than restricting the domain of a signal: $x(t) \mapsto x(t)\, \mathbb{1}_{[t_s, t_e]}$. Practically, the operation re-indexes metadata, rewrites chunk boundaries, and releases memory pages—whether GPU-resident (VRAM) or system-resident (virtual memory)—thereby accelerating every down-stream transform (e.g., FFT, filtering, feature extraction, etc.) while also guarding against aliasing that would arise if one down-sampled first. By reducing the number of resident buffers, trimming alleviates pressure across the memory hierarchy: GPU workloads gain additional VRAM headroom, while CPU-based workflows avoid excessive paging and cache thrashing in system RAM. Because no samples are overwritten, the cut is reversible: clearing the window restores the full record, preserving provenance and reproducibility. Analysts are encouraged to trim iteratively, i.e., begin with a coarse window to maintain GUI responsiveness, refine as patterns emerge, and, if adjacent segments must later be concatenated, preserve a sliver of overlap ($\approx 5\%$) to mitigate edge effects in convolutional or recurrent models. This strategy echoes long-standing advice from digital signal processing and tidy-data theory: reduce data volume before heavy computation, and delineate observational units prior to statistical modeling [70, 71].

4.1.3. Data Fill, Leakage, & Causality

In many practical circumstances, data sequences contain scattered null entries. Missing samples (NaNs, sensor dropouts, empty rows) disrupt statistical analyses and machine-learning pipelines, etc. The dFL pipeline therefore supplies three canonical remedies: (1) **linear interpolation** estimates interior gaps from adjacent points, producing smooth first-order continuity but leaving edge NaNs untouched; (2) **constant edge extension** copies the first or last valid value outward, guaranteeing a complete vector for real-time or frequency-domain algorithms, albeit with flat segments at the boundaries; and (3) **hybrid fill** applies edge extension at the ends and linear interpolation inside, yielding a fully populated record with minimal distortion. An example of how the fill looks is provided in Fig. 2. Further algorithmic details can be found in [72–74] and the dFL documentation. More advanced data imputation strategies are currently under development, and are planned for dFL automatic updates. It should also be noted that users may also easily implement their own fill strategies into the GUI (see the dFL documentation), or perform them using `Fetch Data` in the `data coordinator` API (see Appendix D).

On Causality and Data Leakage: A further concern in harmonization pipelines (particular in real-time control or MPC) is the risk of *data leakage*, where labels or derived features inadvertently incorporate information that would not be causally available at the time of prediction. For example, if gap-filling or normalization procedures draw on post-disruption signals (e.g. feature normalization), or if labels are constructed using smoothed quantities that extend into the future of the prediction window, then downstream ML models may achieve artificially inflated performance. Causal constraints are equally essential for sequential data fusion techniques (e.g., Kalman filtering), where future-dependent preprocessing violates the Markov property and yields non-realizable filters. The dFL framework is designed to mitigate this by allowing causal constraints to be imposed during labeling and/or preprocessing, such as restricting fill operations to past-only data, or by explicitly tracking provenance so that analysts can audit whether any step introduces information from outside the causal horizon. Ensuring causal consistency is therefore essential for both scientific validity and the credibility of ML-driven inference in fusion workflows, and should be kept in mind for all subsequent harmonization and labeling steps.
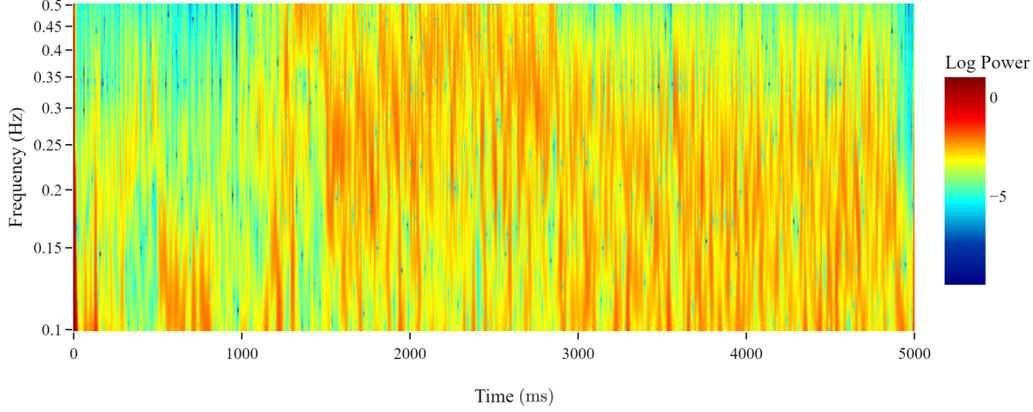
Wavelet Analysis: ip

Figure 3: Here we show one of the natively supported frequency space graphs, a wavelet transform, with `Window Size`=256, `Overlap`=50%, and `Morlet Width Param`=5, for the plasma current i_p (Shot #149058).

4.1.4. Visualization, Navigation, Analysis, and Dynamic Resampling

Interactive visualization is central to effective data exploration, physical intuitive building, and labeling in dFL, particularly when working with large, high-dimensional time-series or spatial datasets. The interface leverages Plotly's built-in navigation tools for *panning*, *zooming*, *resetting axes*, *interacting* with and *selecting* different signals, and *autoscaling*, enabling users to adjust their view of the data in real time. These interactions are tightly coupled with *dynamic resampling* to ensure that both the visualized data and its associated labels remain clear, accurate, and responsive at all scales.

Labeling tasks such as marking plasma states, identifying anomalies, or tagging regions of interest, often require changing the granularity of the displayed data. When a user zooms in on a temporal or spatial region, the system resamples the data to a finer resolution, revealing more detail and enabling precise placement of labels and annotations. Without such resampling, labels can become misaligned, overlap, or lose contextual meaning. Conversely, when zooming out, the system reduces resolution to maintain visual clarity, avoid overplotting, and ensure smooth interface performance. In dFL, the default visual resampling method is based on the MinMax LTTB algorithm [75], which efficiently preserves salient features of the signal while adapting the level of detail to the current view.

Beyond just navigation, dFL offers a diverse suite of interactive graphing and analysis tools for deep insight generation. Standard visualizations include time series plots with interactive signal toggling, multiple correlation plot types (scatter, heatmap, pairwise grids) supporting Pearson, Spearman, and Kendall methods, distribution plots (histograms and kernel density estimates), statistical analysis plots (rolling z-scores, CUSUM charts, moving-average confidence bands), and time-frequency visualizations such as spectrograms, short-time Fourier transforms (STFT), and continuous wavelet transforms (CWT) (see Fig. 3, for example). All plots support parameter tuning, overlays, and result export, with behaviors optimized for responsiveness even on large datasets. Full descriptions of all natively supported graphs types, along with definitions of all parameter settings, can be found in the full dFL documentation.

The system is also designed for extensibility. Custom graph types can be readily integrated by writing lightweight Python backends using the dFL `custom grapher dictionary` in the `data coordinator` API, which can easily leverage libraries such as `matplotlib` or `seaborn`, if so desired. This architecture allows advanced users to tailor visualizations to domain-specific needs, incorporate specialized statistical metrics, and adapt styles for publication-ready output. Detailed descriptions, parameterization guidelines, and implementation examples for all supported visualization types are provided in the full dFL documentation, and an example script is provided in Appendix A. Custom visual layers can be used to compute data fusion-relevant summaries (e.g., per-mode SNR or posterior weights), then incorporated
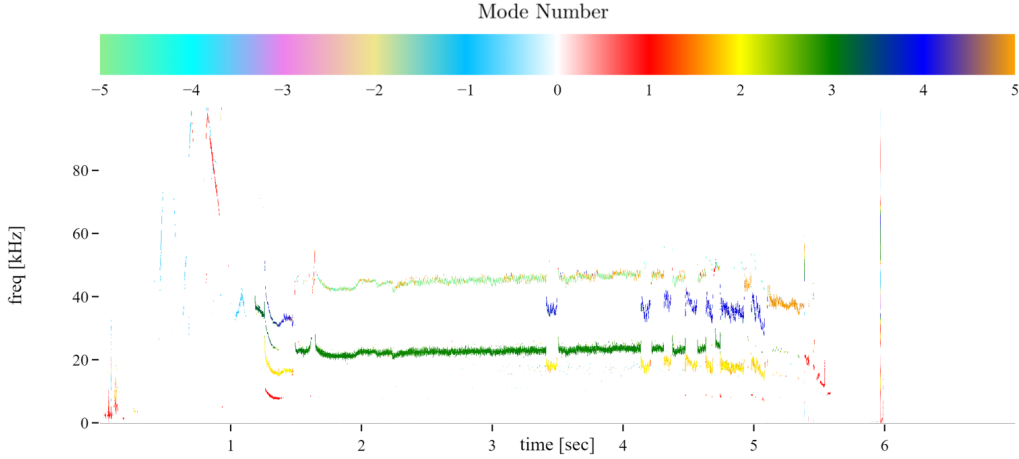
Figure 4: Here we show a custom graph type incorporated using the python-based Modespyec program for magnetic mode analysis. The toroidal modes are set by contour colors (Shot #149091).

into the dFL GUI, and their lineage exported. Figure 4 shows an example of a highly customized Modespyec graphing tool integrated and available in dFL, with details provided in the dFL documentation.

4.2. Data Resampling

Resampling plays a critical role in aligning and adjusting datasets from different sources, particularly when integrating data from fusion diagnostics and simulations. Fusion diagnostics, which monitor real-time plasma behavior, often collect data at varying sampling rates to capture transient phenomena such as plasma instabilities or rapid changes in magnetic fields. However, simulations, which model long-term behavior or large-scale dynamics, tend to produce results at often either lower (or at times, much higher) temporal resolutions than the corresponding experimental information. To make meaningful comparisons between these data types, resampling techniques must be employed to adjust the sampling rate of one, and often both datasets, ensuring that they are aligned temporally on regular numerically stable grids. This may involve downsampling high-frequency experimental data to match the lower resolution of simulation outputs or upsampling the simulation results to match the more granular experimental measurements, and any mixture of the two between different diagnostics and or simulations frameworks, etc.

The resampling process must be executed carefully to avoid pitfalls such as *aliasing*, where high-frequency components of the data are incorrectly represented when downsampled, or the loss of important granular features that can occur when upsampling. To mitigate these issues resampling techniques, such as, e.g., linear, spline, or cubic interpolation, can be employed to ensure that key characteristics of the data are preserved. For example, cubic interpolation is
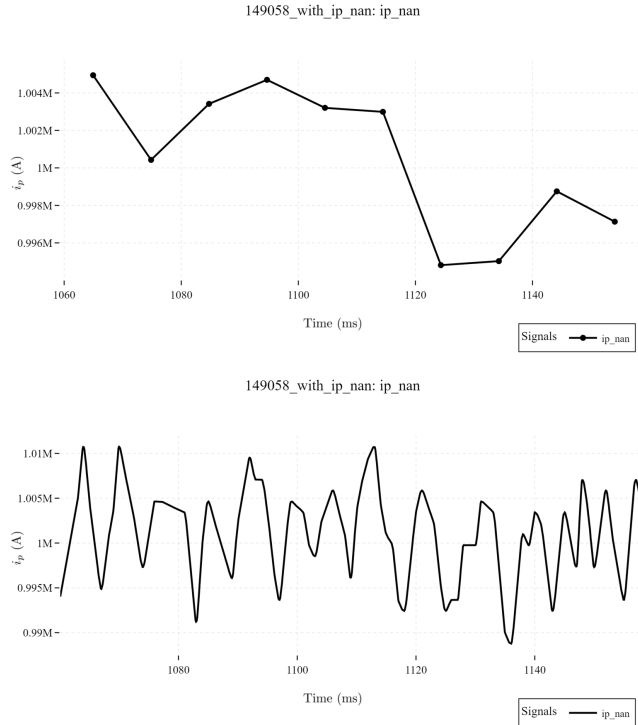

Figure 5: The same 100 point data segment shown in figure 2, first filled, then on top downsampled to 10 points using importance downsampling preserving the first two central moments, and on bottom upsampled to 1000 points using Mono-PCHIP.

often used when upsampling to maintain smooth transitions between data points and to prevent sharp discontinuities that could arise from simpler methods like linear interpolation. While resampling ensures consistency in time-scale, it also introduces challenges in balancing computational efficiency, data fidelity, and data consistency, particularly when dealing with the large, multimodal datasets found in fusion energy. Critically, downsampling before data fusion should include matched anti-alias filtering so that information from a high-rate channel is not spuriously injected at frequencies unsupported by lower-rate channels, which would otherwise bias frequency-domain data fusion (e.g. coherence-weighted estimators).

dFL natively supports five upsamplers and five downsamplers. Most of these sampling techniques are conventional and implemented using well-established techniques. An example of the resampling is provided in Figure 5. Note, additional details and references to all the resampling techniques are provided in the online dFL documentation and in Appendix E and Appendix F. It is worth noting that in certain instances, specific properties of the signal need to be preserved (e.g. spectral properties), in which case custom resampling might be required. Options for integrating user-defined resampling methods into the dFL GUI are included in the dFL documentation (e.g., any DSP tool from `scipy.signal` can be integrated with a few lines of code), as are preprocessing examples using `Data Fetch` to load the data into dFL, as shown in Appendix D.

4.3. Data Smoothing

Smoothing is a foundational operation in the broader process of data harmonization, serving as a bridge between raw, irregular, or noisy measurements and the coherent, analysis-ready representations required for high-fidelity interpretation. Far from being a cosmetic transformation, smoothing plays a decisive role in suppressing spurious high-frequency components often arising from sensor limitations, electromagnetic interference, numerical discretization artifacts, or signal degradation that do not correspond to the underlying physical phenomena. In fusion energy research, particularly in the interpretation of tokamak diagnostic signals, these noise sources can mask or distort the signatures of genuine plasma behavior, leading to mis-characterization of events or erroneous parameter estimation. In addition, it should be duly noted that generally smoothing, like down-sampling, generates information loss and is non-invertible. As a consequence, in formal data fusion, for example, smoothing alters both the autocovariances and cross-covariances that many fusion techniques exploit. This is another reasons why dFL records and tracks smoothing kernels so that downstream estimators can adjust R_i (and the covariance between signals R_{ij}) accordingly.

Because of this nuance the importance and delicacy of smoothing is sometimes underestimated in discrete data workflows, where the emphasis instead often shifts prematurely toward feature extraction, modeling, or machine learning. Without proper smoothing, however, the downstream stages of data harmonization risk propagating noise as if it were signal, reducing predictive accuracy and obscuring physically meaningful correlations. For example, in spectral data fusion (e.g., coherence-weighted mode tracking), over-smoothing can attenuate narrow-band features and degrade fusion weights tied to cross-power estimates. In this sense, smoothing is not merely a preparatory step, but a necessary safeguard against the contamination of the scientific inference pipeline.

The choice of smoothing method is highly context-dependent: it must respect both the statistical structure of the data and the physical context of the measurement process. An effective smoothing strategy should remove noise without erasing sharp but genuine features such as mode transitions, instability onsets, or abrupt control interventions. Striking this balance requires a principled understanding of the datasetâĂŹs spectral content, temporal or spatial resolution, and the downstream tasks it must support.

It is also important to distinguish smoothing from resampling, though certain resampling operators (see Section 4.1) incorporate implicit smoothing to suppress aliasing. Within
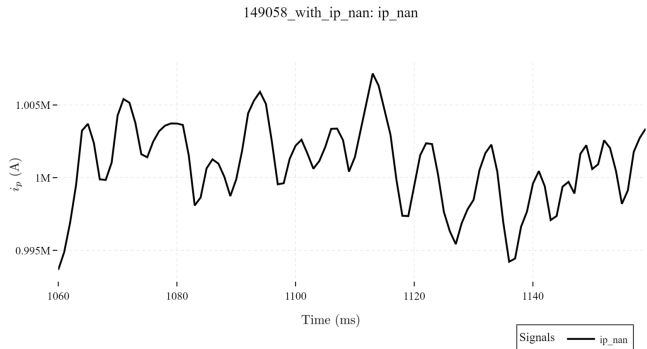

Figure 6: The same 100 point data segment shown in figures 2-5, but here smoothed after fill using EMA with a span of 5.

dFL, smoothing is treated as a first-class, explicit operation whose parameters can be tuned independently of resampling, ensuring the flexibility to adapt to heterogeneous sources and irregular sampling grids typical of multimodal fusion datasets. The nuances of where smoothing belongs in the data processing pipeline—especially in the context of the order of operations—is addressed in Section 4.6. Moreover, building in custom smoothing and/or resampling into the dFL GUI can sometimes be ideal in order to preserve domain-specific features, e.g., spectral information. Details of how to incorporate new smoothing/resampling algorithms can be found in the dFL documentation, and instructions on how to preprocess datasets using the `fetch data` functionality in the dFL `data coordinator` API is show in Appendix D. Examples and definitions of natively supported smoothing algorithms in dFL are listed in Appendix G. All smoothing parameters are exported so that data fusion methods can reconstitute the effective transfer function applied to each channel.

4.4. Data Normalization

Normalization is another foundational operation in the data fusion and harmonization pipeline, ensuring that heterogeneous measurements can be meaningfully compared, integrated, and analyzed without scale-induced biases. In multimodal fusion datasets, diagnostic variables span orders of magnitude in both units and absolute values, ranging from electron temperatures in the keV regime, to magnetic field strengths in milliteslas or teslas, to particle densities exceeding 10^{20} m^{-3}. Left unnormalized, such disparities distort statistical measures, inflate condition numbers in numerical algorithms, and can cause optimization procedures in numerical models to converge poorly or toward biased minima. Normalization also improves numerical stability in data fusion solvers that invert normal equations (or accumulate information matrices), reducing sensitivity to poor conditioning that can otherwise misallocate weights across modalities.
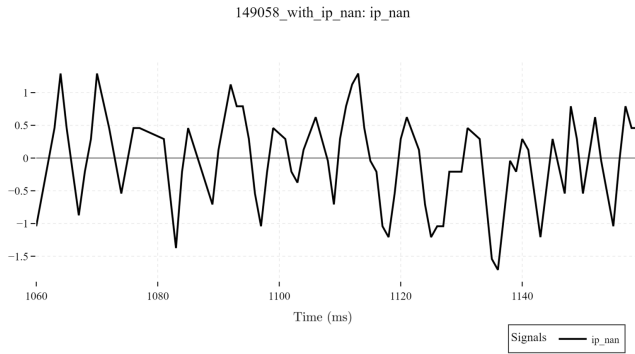

149058_with_ip_nan: ip_nan

Figure 7: The same 100 point data segment shown in figures 2-6, but here normalized after fill using Median-IQR (units are dimensionless).

The role of normalization extends beyond mere unit adjustment: it is a deliberate transformation that places diverse features on a comparable scale while preserving their intrinsic relationships and physical interpretability. For example, a model ingesting raw temperature and density measurements without normalization may incorrectly prioritize the variable with the largest numerical range, masking subtler but physically significant variations in other channels. This is particularly detrimental in fusion research, where meaningful cross-diagnostic correlations, such as between density gradients and magnetic topology, may occur at vastly different scales.

Different normalization strategies (e.g., Min-Max scaling, z-score standardization, robust scaling) impose different statistical and physical constraints on the transformed data. The appropriate choice depends on the intended downstream use, the noise characteristics of each diagnostic, and whether the preservation of relative amplitudes or the enforcement of statistical standardization is more critical. In the context of fusion workflows, normalization is not just a statistical convenience, it is an essential precondition for integrating diverse measurements into unified, physically coherent datasets suitable for reliable feature extraction, model training, and multi-source data fusion.

4.5. Integrating Custom Filters & Feature Maps into dFL

Any custom filter (be it a DSP filter, a custom normalization, resampling, smoothing, imputation, etc.), or, more generally, feature map may be integrated into dFL using either the `Fetch Data` functionality in the dFL `data coordinator` API discussed in the dFL documentation, or by integrating custom filters into the smoothing or normalization GUIs (NOTE: these can be custom filters for any purpose, that will only show up in the smoothing or normalization dFL dropdowns, along with any provided parameters). It is worth noting that feature maps often serve as the inputs to feature-level data fusion (e.g., combining per-mode amplitudes from Modespyec with edge profile gradients); where dFL preserves their lineage and parameters so that fused conclusions remain auditable.

149091: magnetics_amp_5, magnetics_amp_4, magnetics_amp_3, magnetics_amp_2, and magnetics_amp_1
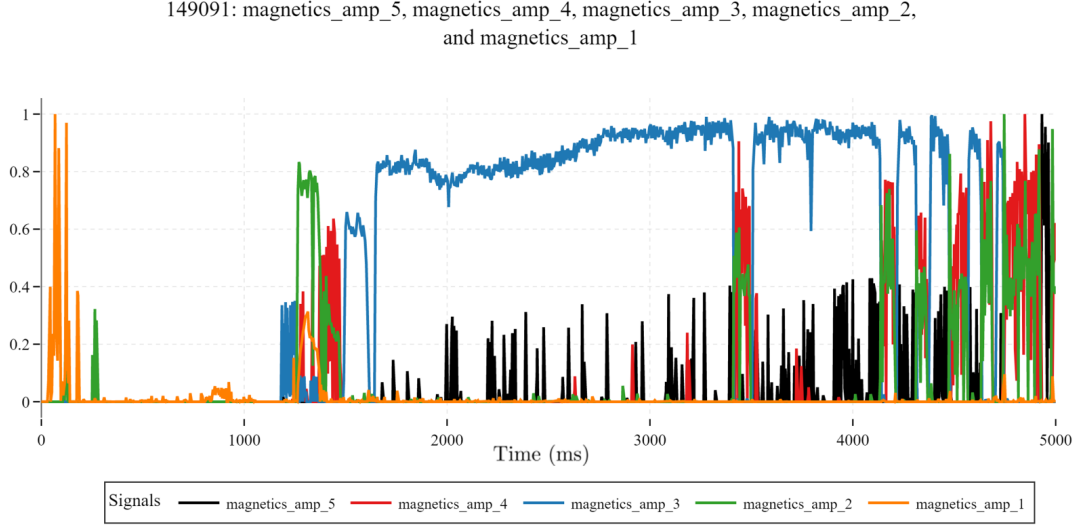
Figure 8: RMS amplitude vs. time of first 5 (positive) coherent toroidal modes n for shot 149091 (Dark theme), projected from the 2D mode spectrum data in Fig. 4. Estimates use two toroidally separated probes ($\Delta\theta$), keeping only frequency bins with high cross-coherence and cross-phase $\phi_{12} \approx -n\Delta\theta$; the mean autospectral power over selected bins yields the output. Traces are Min-Max normalized for visualization (units are dimensionless).

A simple example of using a custom feature map to extract relevant 1D time-series signals from 2D source signals is shown in Figure 8, where amplitudes associated with toroidal mode numbers are extracted from the Modespyec spectrum. More details and full code is available in the dFL documentation.

4.6. Order of Operations & Exporting to Databases

The preprocessing of fusion data—prior to export for simulation, analysis, machine learning (ML) and artificial intelligence (AI) tasks—involves a sequence of operations that act as functional transformations on noisy, sparse, and multimodal inputs. Each operation constitutes a potentially non-commutative map on the input space, and therefore, the order in which these transformations are applied is mathematically and empirically critical. This section introduces a mathematically principled view of the recommended pipeline implemented in the Data Fusion Labeler (dFL), emphasizing the critical importance of operator ordering, and illustrating the consequence through practical examples.

Operator Ordering in Data Harmonization

Let $\mathcal{X}$ denote the space of input signals, such that $x \in \mathcal{X}$ is a univariate or multivariate time series. Each preprocessing step, Trim (T), Fill (F), Resample (R), Smooth (S), Normalize (N) acts as an operator on $\mathcal{X}$. Generally any two of these operators A and B commute only if $A \circ B(x) = B \circ A(x)$ for all $x \in \mathcal{X}$. In practice, most preprocessing operators do *not* commute, i.e.,

$$N \circ S \neq S \circ N, \quad S \circ R \neq R \circ S, \quad R \circ F \neq F \circ R.$$

This non-commutativity implies that arbitrary reordering of operations yield different outputs, even when the same raw signal is used as input.

Recommended Preprocessing Pipeline

To ensure statistical consistency, scientific reproducibility, and to minimize the risk of introducing artifacts that propagate into numerical models (e.g. classical simulations, ML/AI pipelines, data analysis, etc.) we recommend (by default) the following composite operation O_3 when dealing with large data sets on irregular or multiscale grids, given by:

$$O_3 := \text{Trim} \to \text{Fill} \to \text{Resample} \to \text{Smooth} \to \text{Normalize} \to \text{Export},$$

15

where each step in O_3 is chosen to logically prepare the data for the next corresponding step. This workflow often makes sense when each step is needed, though different orderings can be used to greater or lesser effectiveness given a specific context (examples below). However, it should be noted that classical DSP theory strongly recommends smoothing first (before resampling) to avoid generating aliasing error when dealing with uniformly sampled data at a fixed sample rate that has real high-frequency content (i.e. edges, spikes, oscillations close to the Nyquist frequency), and is being downsampled by a large factor, and/or is intended to be analyzed in frequency space (e.g. Fourier transformed, etc.). Another example of where O_3 may be non-optimal is smoothing before normalizing. Since smoothing generally changes the signals variance, normalizing should generally be done second, but often numerically ill-conditioned physical units (e.g. 10^{23}) can make tuning smoothing parameters challenging and fraught, making the seemingly redundant workflow, Normalize $\rightarrow$ Smooth $\rightarrow$ Normalize, much more practical in order to preserve signal variances between channels. As a consequence, depending on how many different types of data are being incorporated into your fusion pipeline, determining the correct order of operations can be subtle, and should be done in a principled way.

Simple Example: Non-Commutativity of Operations

Consider a raw signal sampled every 1 second, and given by the data vector, $x = [2, \text{NaN}, 6, 5]$. Let us define the preprocessing pipeline with the following configurations: (i) Fill (F) via linear interpolation; (ii) Resample (R) to 0.5 s (upsampling by a factor of 2); (iii) Smooth (S) via a 3-point moving average; and (iv) Normalize (N) using z-score standardization. Performing these operations on a pipeline A, such that $A: F \rightarrow R \rightarrow S \rightarrow N$, directly yields the following:

1. Fill: $x_{\text{filled}} = [2, 4, 6, 5]$

2. Resample: $x_{\text{resampled}} = [2, 3, 4, 5, 6, 5.5]$

3. Smooth: $x_{\text{smoothed}} = [\text{NaN}, 3, 4, 5, 5.5, \text{NaN}]$

4. Normalize: $\mu = 4.38$, $\sigma \approx 0.96 \Rightarrow x_{\text{norm}} \approx [\text{NaN}, -1.43, -0.39, 0.65, 1.17, \text{NaN}]$

Now, performing these same operations in a different order, namely pipeline B, which simply changes the order of one operation, $B: F \rightarrow R \rightarrow N \rightarrow S$, yields:

1. Fill: $x_{\text{filled}} = [2, 4, 6, 5]$

2. Resample: $x_{\text{resampled}} = [2, 3, 4, 5, 6, 5.5]$

3. Normalize: $\mu = 4.25$, $\sigma \approx 1.41 \Rightarrow x_{\text{norm}} \approx [-1.6, -0.89, -0.18, 0.53, 1.24, 0.89]$

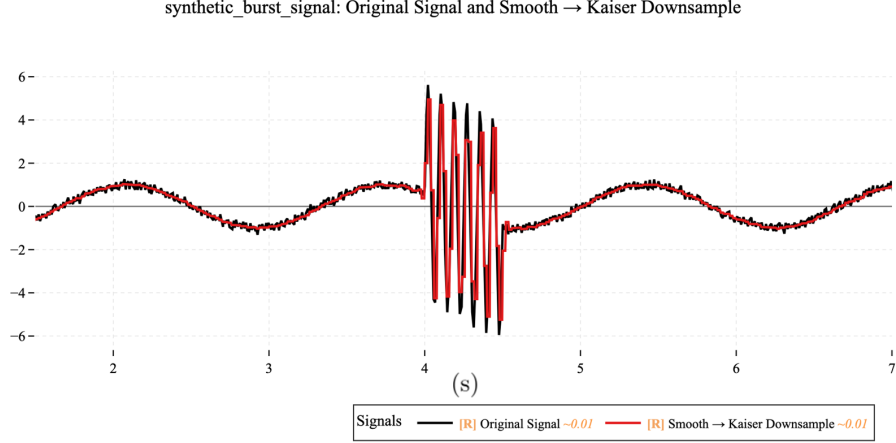4. Smooth: $x_{\text{smoothed}} \approx [\text{NaN}, -0.89, -0.18, 0.53, 0.89, \text{NaN}]$

Note that the final output differ in both magnitude and structure between the pipelines. Notably, applying smoothing after normalization distorts the statistical properties that the normalization was intended to standardize, showing that $N \circ S \neq S \circ N$ and highlighting the mathematical necessity of order-awareness in data processing workflows.

Advanced Example: Non-Commutative Smoothing & Polyphase Kaiser Down-sampling

To highlight how operator order alone can turn a dramatic transient into a barely perceptible wiggle (even when using advanced signal processing tools), we synthesize a 10 s record that contains a slow 0.6 Hz sine, mild Gaussian noise, and a brief, high-amplitude 12 Hz burst confined to $t \in (4, 4.5)\,\text{s}$ (see Appendix B for details). The two preprocessing pipelines applied are:

$$\text{Pipeline A}: \text{SG}(31, 3) \longrightarrow \text{Kaiser}(d{=}10, \beta{=}2),$$

$$\text{Pipeline B}: \text{Kaiser}(d{=}10, \beta{=}2) \longrightarrow \text{SG}(31, 3),$$

where 'SG" is a Savitzky-Golay local cubic smoothing [76] with a window of 31, and Kaiser" is a Kaiser polyphase FIR (Finite Impulse Response) [77] with a downsampling factor $d = 10$ and the Kaiser-Bessel parameter set to $\beta = 2$. The results are stretched back to the original grid for direct comparison (working code included in Appendix B). Figure 9 shows that Pipeline A retains the burstâĂŹs ± 5 swing, whereas

synthetic_burst_signal: Original Signal and Smooth → Kaiser Downsample

(a) Pipeline A: Smooth → Kaiser Downsample. In this case the burst survives, attenuated but unmistakably still present.

synthetic_burst_signal: Original Signal and Kaiser Downsample → Smooth

(b) Pipeline B: Kaiser Downsample → Smooth. AntiâĂŚalias filtering eliminates most high-frequency energy before the smoother sees it; the burst is nearly erased.

Figure 9: Operator order matters even with an 'ideal' polyphase FIR. A single transient critical for fault detection can vanish when downâĂŚsampling precedes smoothing, underscoring the nonâĂŚcommutativity of these preprocessing steps even when utilizing advanced/composite signal processing tools (Publication theme).

Pipeline B suppresses it, leading to a dramatic numerical difference in the two simple processing pipelines, $\max |A - B| = 4.6$. It is worth noting that polyphase Kaiser down-sampling is popular in part because it applies a low-pass filter whose cutoff is matched to the new Nyquist rate, and that internal filter is the 'smooth-before-downsample" safeguard from classical DSP theory [78]; especially when applied to regularly gridded data. Doing it here ensures alias-free rate conversion before you touch any cosmetic noise shaping, but at the cost of a loss in potentially meaningful physically relevant information (i.e. the burst in Fig. 9).

Exporting Harmonized Datasets

Once the preprocessing pipeline O_3 (or a custom mapping) has been applied, the processed dataset $\hat{x} = O_3(x)$ is ready for export. The dFL supports schema-aware export (e.g., CMF, IMAS, TokSearch), preserving both the harmonized data and associated metadata. While the default O_3 flow may be safest assuming a high dimensional irregular multimodal dataset, users may easily override it with a custom sequence tailored to their specific scientific goals, provided they take note of the consequences of non-commutativity and information loss, and the critical need to store operation order as metadata in the

17

data provenance pipeline (discussed in more detail in Section 5). Extensive options are available in dFL for custom exporting of datasets and details can be found in the dFL documentation.

5. Data and Metadata Standardization and Provenance

Provenance in dFL is a critical and central design principle. Every action taken within the system—from data ingestion, to preprocessing, to manual or automated labeling, to export—is recorded and can be extended by the user to capture custom metadata. This is achieved through three extensibility points: the `manual_labeling_hook`, the `label_export_hook`, and the `data_export_hook`. Each hook exposes the internal state of the workflow at a critical transition, allowing users to inject additional context, perform side logging, or direct data and labels into external provenance frameworks such as CMF.

The `manual_labeling_hook` is invoked each time a user creates a label. By default, dFL records dataset identifiers, shot numbers, and temporal bounds, but this hook allows the label row to be augmented with richer metadata—for example, the annotator identity, timestamp, or derived features computed on-the-fly. This ensures that manual operations are traceable and reproducible across campaigns. The `label_export_hook` intercepts the labeling dataset at the moment of export, enabling versioning, stamping, and redirection into institutional repositories or metadata services. Finally, the `data_export_hook` governs the export of harmonized signal data. It is invoked both for individual graph-level exports and for bulk operations, providing not only the processed signals but also a structured metadata document describing the applied pipeline (e.g., fill strategy, resampling parameters, smoothing operations, and order of execution).

Together, these three hooks establish a flexible but rigorous provenance layer: all transformations are explicit, all exports are accompanied by machine-readable metadata, and users can extend the default tracking to meet local compliance or archival requirements. This architecture ensures that downstream analyses in fusion science are not only reproducible in principle but auditable in practice, with full transparency about what data were transformed, how, and by whom.

5.1. Benefits of Standardized Data Schemas (e.g., IMAS, OMAS)

The International Magnetics Fusion Research (IMAS) data standard is a cornerstone framework for organizing, storing, and exchanging data within the nuclear fusion community. It defines a unified schema capable of representing heterogeneous data sources (e.g., from diagnostic measurements and simulation outputs to control system telemetry) in a consistent and interoperable manner. By enforcing a common data model across institutions and experiments, IMAS enables seamless integration, comparison, and cross-validation of results, thereby accelerating collaborative research and reducing redundancy. This interoperability is particularly critical for multi-facility projects and for building machine learning (ML) models that require large, diverse, and standardized datasets.

OMAS (Ordered Multidimensional Array Structures) complements IMAS by providing a flexible yet schema-driven data structure optimized for the complex, high-dimensional datasets produced in fusion experiments and simulations. OMAS facilitates hierarchical organization of time-dependent and multidimensional data, ensuring compatibility with multiple software tools and analysis pipelines. Its efficient serialization formats and scalable architecture make it well-suited for both archival and real-time processing needs. Together, IMAS and OMAS form a robust ecosystem for ensuring that data are not only interoperable but also provenance-aware, enabling transparent and reproducible analysis in both research and operational contexts.

5.2. dFL Integrated Tools for Data & Metadata Management, Validation, and Tracking

The Data Fusion Labeler (dFL) also integrates with a variety of backend systems and frameworks to ensure that data ingestion, labeling, curation, and export occur in a standardized, reproducible, and provenance-rich manner. These integrations allow users to seamlessly access heterogeneous datasets, maintain metadata integrity, and comply with international data schema standards.

5.2.1. TokSearch

TokSearch is a high-performance query and retrieval engine developed for fusion energy research. Integrated into dFL as a backend data provider, TokSearch enables rapid, targeted queries across massive, multimodal experimental datasets (i.e. often spanning decades of operation) without requiring manual file handling. Its query system supports flexible searches by shot number, diagnostic type, temporal range, operational regime, or derived physical parameters. In practice, TokSearch can, for example, return all bolometry and magnetic fluctuation signals from shots exhibiting specific confinement transitions, or retrieve synchronized diagnostics and actuator traces for disruption analysis.

When used within dFL, TokSearch ensures that labeled datasets are drawn from reliable sources and are consistently aligned across diagnostics. Queries can be embedded directly in dFL ingestion scripts, so labeling workflows always operate on well-prepared subsets of data. Importantly, TokSearch has been demonstrated to scale efficiently on HPC systems, with parallel backends (Ray, Spark, and SLURM integration) enabling throughput improvements of several orders of magnitude compared to traditional access methods. This scalability makes it practical to apply dFL to petabyte-scale archives typical of modern fusion facilities, and its ongoing extension to multiple devices further broadens the scope of labeling and analysis that dFL can support.

5.2.2. CMF

The HPE Common Metadata Framework (CMF) [79, 80] provides a modular, service-oriented architecture for managing the complex, heterogeneous datasets inherent to fusion energy research. CMF addresses challenges such as handling multimodal, imbalanced, noisy, and harmonized datasets, as well as the sparse labeling characteristic of fusion diagnostics. Its composable design allows seamless integration of preprocessing tasks, normalization, resampling, smoothing, etc., while providing robust infrastructure for manual and automated labeling. CMF manages code, data and metadata (parameters, metrics) in Git-like fashion in a single framework, enabling reproducibility and facilitating reuse of workflows that preprocess multi-diagnostic data streams into a harmonized format, ensuring consistency before they enter the dFL labeling interface.

Crucially, CMF offers advanced metadata management capabilities aligned with IMAS and OMAS standards, supporting dynamic updates as new labels or harmonized datasets are generated. This allows provenance metadata to be recorded at fine granularity whether during batch exports, individual label creation, or automated labeling runs, capturing timestamps, personnel identifiers, data versions, and processing history. With comprehensive data, metadata and lineage query capabilities, CMF helps development of data discovery tools based on analyzing correlations between quality of results, workflow parameters, and input data. CMF supports management of labels as metadata, enabling rapid analysis of model sensitivities to preprocessing tasks and label sets without the need to access the data stores. Integration with dFL is achieved through Python ingestion scripts that invoke CMF commands at key stages of the workflow, enabling transparent, version-controlled curation.

It should be noted that CMF can be readily integrated at the dFL GUI level, where dFL will handle metadata handling for all harmonization and labeling tasks natively. However, if the user wants to incorporate custom preprocessing using the `Fetch Data` approach shown in Appendix D, then the metadata handling must be dealt with at the CMF level.

5.2.3. MGKDB

The Multiscale Gyrokinetic Database (MGKDB) is a specialized data infrastructure for storing, managing, and analyzing the large, high-dimensional outputs of gyrokinetic simulations—essential for advancing the understanding of plasma turbulence and transport phenomena. Uniquely, MGKDB natively stores its contents in an *IMAS-compliant schema*, ensuring that simulation results share a common structure with experimental data repositories. This schema conformity enables seamless interoperability with other IMAS-aware tools and workflows, and allows simulation data to be cross-referenced, merged, and jointly analyzed alongside diagnostic measurements from operational devices.

MGKDB's hierarchical organization accommodates complex, multi-parameter simulation outputs, capturing both scalar and field quantities across space, time, and configuration parameters. When integrated with the Data Fusion Labeler (dFL), synthetic diagnostics generated from simulations can be labeled using the same conventions and interfaces as experimental diagnostics. This alignment not only streamlines data curation but also improves the fidelity of machine learning models that bridge theory and experiment, by ensuring that training datasets reflect a harmonized representation of both

sources. Furthermore, MGKDBâĂŹs standardized access patterns and version-controlled metadata allow simulation-derived labels and processed datasets to be reproduced, audited, and compared across research institutions, fostering collaborative validation and model benchmarking at scale.

5.2.4. OMFIT

The One Modeling Framework for Integrated Tasks (OMFIT) is a modular environment for managing, analyzing, and visualizing fusion energy data. Its built-in modules for profile analysis, MHD stability, transport modeling, and equilibrium reconstruction make it highly relevant to labeling workflows where physical interpretation and consistency checks are essential. OMFITâĂŹs scripting capabilities allow for the automation of preprocessing, labeling, and export tasks within dFL pipelines. Furthermore, OMFITâĂŹs native support for IMAS/OMAS ensures schema compliance, while its ability to handle dynamic, nonstationary, and multiscale datasets makes it well-suited for labeling tasks that require time-dependent context. In addition, TokSearch is integrated with OMFIT to support large-scale data ingestion, providing a seamless path from high-throughput retrieval to analysis workflows. The integration of OMFIT into dFL enables a unified workflow in which advanced modeling and manual or automated labeling reinforce one another, improving both label quality and downstream scientific utility.

6. Data Labeling

The value of any machine learning (ML) or artificial intelligence (AI) framework in fusion energy science hinges critically on the availability of accurate, context-rich, and physically meaningful labels. In the absence of such labels, supervised learning pipelines cannot reliably distinguish between stochastic fluctuations, diagnostic artefacts, and the true signatures of underlying plasma phenomena. Data labeling is thus not an ancillary step, but a central act of *knowledge encoding*: it embeds domain expertise into the dataset, transforming raw multimodal streams into structured corpora that can train models, validate theories, and guide operational decisions.
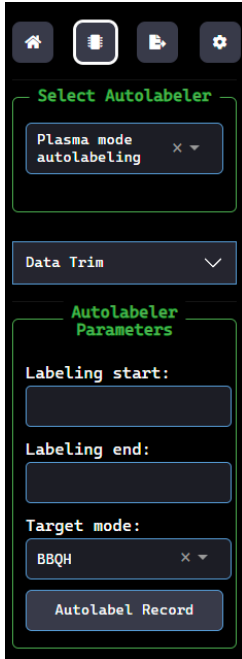

Figure 10: The classifier-based plasma mode auto-labeler in the dFL GUI (Sophelio dark theme).

Fusion energy datasets pose unique labeling challenges compared to other scientific domains. First, their complexity is inherently *multimodal*, combining signals from diverse systems, such as magnetic diagnostics, bolometry, reflectometry, Thomson scattering, neutron detection, imaging systems, synthetic diagnostics from multiscale simulations, etc. Each modality captures different projections of the same evolving plasma state, with diverse units, sampling rates, noise characteristics, and temporal alignments. This heterogeneity demands a harmonization step before labeling can even begin, or else labels refer to inconsistent or misaligned events. Second, fusion data exhibit extreme *class imbalance*: events of high scientific and operational significance—such as major disruptions, edge-localized modes (ELMs), or neoclassical tearing modes—occur far less frequently than stable operating intervals, yet their correct identification is crucial for both understanding plasma physics and ensuring machine safety.

Labeling in this context is further complicated by the *nonstationary* nature of the plasma and its surrounding environment. Operational conditions, control schemes, and even diagnostic calibrations evolve across shots and campaigns, so that the same physical event may present differently depending on machine configuration, plasma scenario, or sensor drift. In this sense, temporal context becomes essential: a label attached to a transient must be interpretable relative to preceding and subsequent plasma conditions. Moreover, many phenomena of interest often unfold across disparate timescales, from microsecond turbulence bursts to confinement regime changes over seconds, necessitating multi-resolution labeling strategies that preserve hierarchical temporal relationships.

From a methodological perspective, the sparsity of human-labeled data in fusion research cannot be overstated. Expert labeling is expensive in both time and cognitive load, and is often constrained to a handful of well-characterized discharges. This scarcity has motivated the adoption of semi-supervised, weakly supervised, and active learning paradigms [16, 18], in which algorithms leverage limited labeled data to guide the selection of new samples for expert annotation or to infer labels across unlabeled regions. In these frameworks, label uncertainty must be explicitly represented and propagated to downstream models, particularly

when labels are derived from heuristic thresholds, statistical classifiers, or surrogate models trained on simulation outputs [23, 24].

Finally, the integration of labeling into the broader *data fusion* workflow (cf. Section 4) has deep implications for reproducibility and scientific validity. Labels that are not tied to harmonized, provenance-aware data streams risk becoming ambiguous or non-transferable between facilities, campaigns, or code bases. Conversely, when labeling is embedded in a harmonized pipeline, complete with temporal alignment, unit standardization, and metadata preservation, it becomes a reusable asset, enabling cross-experiment benchmarking, multi-device model training, and transparent physics discovery. In this way, robust labeling not only serves immediate ML tasks but also contributes to the long-term goal of building a federated, interoperable corpus of fusion knowledge.

Below we explore the manual, automated, and hybrid approaches to labeling in fusion energy science, detailing the algorithmic strategies, human-machine interaction models, and integration points with dFL that make high-quality labeling tractable at scale.

6.1. Manually Labeling Multimodal Data Sets

Manual labeling remains one of the most reliable and context-aware methods for annotating complex fusion energy datasets. Its primary strength lies in the ability of expert human judgment to identify subtle, context-dependent phenomena, such as precursor oscillations to disruptions, nuanced changes in edge fluctuations, or diagnostic artefacts, that might evade purely automated algorithms. When performed carefully, manual labeling can capture the richness and ambiguity inherent in real-world experimental data, ensuring that event definitions are physically meaningful, aligned with research objectives, and robust against spurious detections. This process is particularly valuable in early-stage campaigns, during exploratory analyses of new diagnostics, or when curating benchmark datasets for algorithm validation.

However, manual labeling also comes with inherent limitations. It is time-consuming, often requiring subject-matter experts to inspect large volumes of data point-by-point or shot-by-shot, which can become prohibitively labor-intensive in the petabyte-scale data regimes typical of modern tokamaks and stellarators. Furthermore, the process can be susceptible to human variability, e.g., different annotators may interpret borderline cases differently, leading to inconsistencies across large datasets. In such contexts, manual labeling is best employed in a targeted manner, either as a gold-standard reference set for validating automated pipelines, or as an expert-in-the-loop refinement step where algorithmic pre-labeling provides candidate regions that are then confirmed or adjusted by humans.

The Labeler (dFL) provides a robust, intuitive, and scientifically oriented interface purpose-built to simplify the manual labeling of complex fusion datasets, be they multidimensional, multimodal, heterogeneous, imbalanced, or contaminated by noise. Its design enables researchers to work with both precision and efficiency, while preserving the full informational richness and physical context of the data.

Within dFL, data are ingested as discrete *records*, selectable from the `Select Record` drop-down menu. A "record" can correspond to a specific *shot* or *discharge* number, a simulation *model* type, a diagnostic *component*, or any other uniquely identifiable data grouping defined by the user. Once a record is selected, all available *signals* associated with that record are loaded into dFL and can be visualized by selecting `Add Graph`. The Labeler natively supports multiple graph types, as outlined in Section 4.1.4. The default view—a high-resolution time-series plot—often serves as the optimal choice for labeling, but alternative visualizations such as spectral plots or user-defined custom plots can be seamlessly integrated and used for manual labeling.

Label creation begins by defining a label name in the `Settings` tab, after which it becomes available in the `Select Labels` drop-down of the main control panel. Additional customization of labels, such as pre-defining categories, structures, or metadata, can be implemented directly within the `data provider` ingestion script. Additional full details are provided in the dFL documentation.

Once a label name is chosen, the labeling process itself can proceed in one of two ways. The user may (*i*) manually enter the start and end times for the label directly into the main control panel, or (*ii*) enable `Label Selection Mode` from the display options. In the latter case, the plotted graphs become fully interactive, allowing the user to highlight the region of interest directly with the mouse. Upon confirming the selection via `Confirm Label`, the label is committed to the session and becomes visible in the `Current Labels` tab of the control panel. This tab not only lists all active labels for the session but also supports direct export of the labeling results to file, ensuring they are immediately ready for downstream analysis, archiving, or integration into machine-learning workflows. A simple manual
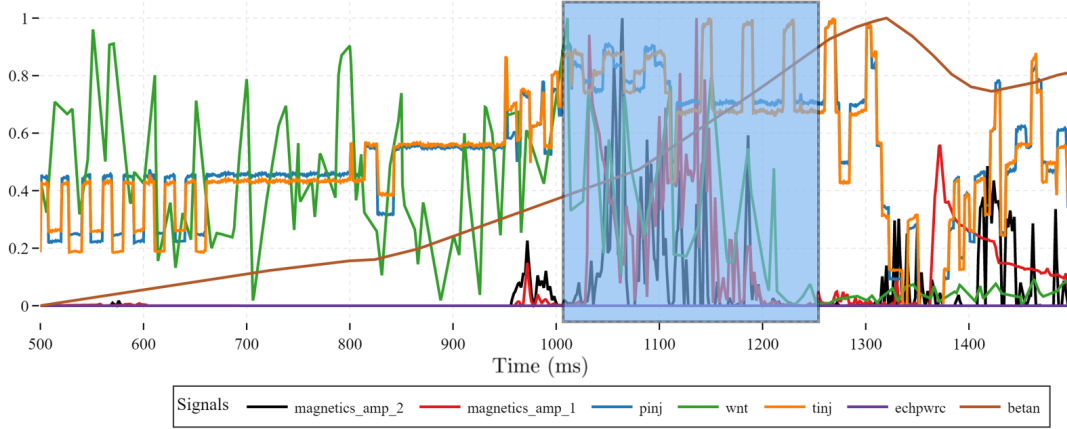
Figure 11: Here we manually label a generic 'magnetic burst corresponding to an average pedestal width "wnt" event' where seven different signals (by TokSearch point names) of a single discharge/shot from DIII-D are loaded on a single plot, and the signals are Min-Max normalized to reveal synchronized onset behavior.

labeling example is shown in Figure 11, and video demos and extensive user support may be found in the dFL documentation.

6.2. Automated Labeling

The complexity, scale, and heterogeneity of data in fusion research—whether from experimental diagnostics, synthetic diagnostics embedded in simulations, or large-scale multiscale modeling pipelines—renders purely manual labeling infeasible at scale. Modern fusion experiments routinely produce petabyte-scale, multimodal time series and imaging datasets per campaign [15, 81], encompassing diagnostics with widely varying sampling rates, signal-to-noise ratios, and spatial/temporal resolutions. Automated labeling frameworks are therefore indispensable for efficiently and consistently annotating these datasets with scientifically and operationally meaningful features. Once deployed, such frameworks not only enable rapid downstream analysis and machine learning (ML) model training but also support real-time or near-real-time operational decision-making, particularly in disruption prediction, regime identification, and advanced control scenarios.

Automated labeling in fusion can be systematically organized into five complementary categories:

i) *Deterministic, physics-informed methods:* closed-form, algorithmic approaches provide interpretable and computationally efficient labeling strategies for signals with reproducible, physically well-defined patterns. Examples include:

- Peak-finding algorithms to detect transient events such as Edge-Localized Modes (ELMs), sawtooth crashes, or pellet ablation signatures in high-cadence magnetic or bolometric diagnostics [82].

- Derivative thresholding to flag abrupt changes indicative of MHD instabilities, confinement transitions, or actuator faults.

- Turning point and zero-crossing detection to mark extrema in simulation outputs, such as minima in magnetic energy during relaxation events or maxima in temperature gradients near transport barriers.

Such methods are particularly effective when coupled to uncertainty-aware preprocessing pipelines (cf. Section 4), but their reliance on fixed thresholds or prescribed signal morphology can make them sensitive to noise and less adaptable to multivariate dependencies.

ii) *Statistics-based methods:* Statistical inference and change detection techniques identify anomalous or regime-shift behavior by quantifying deviations from nominal baselines in a probabilistic framework. Approaches include classical hypothesis testing, control charts, parametric and non-parametric density estimation, and time-series change-point detection [83–85]. For example, statistical process control can flag gradual drifts in diagnostic calibration, while kernel density estimation can detect distributional shifts in reflectometry or bolometry signals. These methods balance interpretability, computational efficiency, and formal uncertainty quantification, qualities that are highly valued in high-consequence scientific workflows. However, they often assume stationarity or specific distributional forms, and performance may degrade in highly non-linear, multi-regime operational settings.

iii) *Data-driven and adaptive methods:* Machine learning-based approaches, ranging from supervised classifiers to fully unsupervised representation learning, offer greater flexibility in handling noisy, high-dimensional, and partially labeled datasets. These methods often rely on learning low-dimensional *embeddings*—compact, structured representations that preserve salient physical or statistical relationships while suppressing noise—to enable robust downstream analysis. Examples include:

- Pre-trained classifiers fine-tuned on fusion-specific data, such as generic convolutional neural networks for identifying plasma boundary features or classifying MHD modes from diagnostic imagery [86].
- Transfer learning from related tasks, enabling rapid adaptation of general-purpose models (e.g., video anomaly detectors, time-series transformers) to specialized fusion contexts without requiring prohibitively large labeled corpora.
- Unsupervised clustering (e.g., k-means, Gaussian Mixture Models, spectral clustering) to reveal latent operational regimes, confinement states, or instability precursors without pre-existing labels [87].
- Representation learning via autoencoders or contrastive methods to generate feature embeddings that are amenable to downstream classification or event detection.

iv) *Simulation and model driven autolabelers:* These methods leverage outputs from high-fidelity physics simulations, synthetic diagnostics, or reduced-order models to generate labels in both simulated and experimental datasets. In fusion research, first-principles simulations such as extended-MHD (e.g., NIMROD, M3D-C1) [51, 88] and gyrokinetic turbulence codes (e.g., GENE, GS2) [2, 3] can be post-processed to extract physically meaningful event markers—such as instability onset times, fluctuation amplitudes, or mode structures—that are then mapped onto experimental data streams via synthetic diagnostic forward models [89]. Reduced-order transport solvers (e.g., TGLF, QuaLiKiz) [90, 91] can similarly provide regime classification boundaries (e.g., L- to H-mode transitions, internal transport barrier formation) that serve as consistent labeling criteria across large archives.

This simulation-informed labeling approach is particularly valuable in data-sparse or rare-event regimes, where experimental coverage is limited but predictive models are available. It enables pre-labeling of operational scenarios that have yet to be experimentally realized, accelerates the training of machine learning models in unexplored parameter spaces, and supports physics-based consistency checks in hybrid labeling pipelines. The primary limitation lies in the fidelity and validation status of the underlying models; any bias, missing physics, or diagnostic misalignment in the simulation may propagate directly into the labels. Consequently, best practice involves cross-validating model-generated labels with statistically independent experimental datasets and quantifying label uncertainty via ensemble or Bayesian simulation frameworks [13].

v) *Hybrid and expert-in-the-loop systems:* The most powerful automated labeling pipelines often blend deterministic physics-informed algorithms, statistical detectors, and adaptive ML-based components, while incorporating expert oversight. For instance, an ELM detector may first use derivative-based thresholds to produce candidate events, which are then filtered and refined by a neural network trained on expert-labeled exemplars. This hybrid approach enhances robustness to noise, addresses rare-event scarcity, and retains interpretability—factors that are essential for operational

trust and scientific reproducibility [16, 18]. When integrated into platforms such as the Data Fusion Labeler (dFL), hybrid pipelines can generate, validate, and deploy labels at scale across archival and real-time data streams.

In the broader context of fusion informatics, automated labeling is not merely a convenience, it is an enabling technology for the next generation of data-driven discovery, model validation, and real-time plasma control. When combined with standardized data models (e.g., IMAS/OMAS) and provenance-aware harmonization workflows, these techniques enable researchers to rapidly extract structured knowledge from complex, multimodal datasets, bridge experimental and simulation domains, and feed high-fidelity, physics-consistent labels into predictive models. As ML and AI methods continue to mature, automated labeling will become a cornerstone of integrated modeling, cross-facility data sharing, and autonomy in the fusion energy ecosystem. Sections 6.2.1–6.2.3 below discuss autolaber types already integrated into dFL.

6.2.1. Statistics-based Autolabeling

Statistics-based autolabeling constitutes a critical component of automated annotation pipelines in fusion research, providing interpretable, uncertainty-aware mechanisms for detecting, characterizing, and segmenting features of interest in large-scale, heterogeneous datasets. These approaches derive their strength from formal statistical inference, allowing event detection to be framed in terms of hypothesis testing, threshold exceedance, or confidence interval breaches. They have been applied to a wide range of fusion data analysis tasks, from identifying abrupt confinement transitions and fluctuation-level changes preceding disruptions, to detecting anomalous sensor behavior in magnetic, bolometric, and reflectometric diagnostics.

In the *Data Fusion Labeler* (dFL), three widely used statistical anomaly detection methods are implemented within the `Stats` graph type (e.g., see Figure 12). Once the user selects `Stats` under the *Graph Type* menu, they may choose from: (*i*) a moving-window z-score deviation detector, (*ii*) a moving-window cumulative sum (CUSUM) chart, and (*iii*) a moving-average with confidence intervals (CI). Each method operates over a user-defined window size W and includes an additional sensitivity parameter (e.g., threshold or confidence level) that governs detection strictness.

Figure 12: A moving-window z-score deviation anomile detector, setting the window size to 200 ms and the threshold to 3σ on a Min-Max normalized Mirnov coil magnetics signal (units are dimensionless).

Viewing options include whether or not to highlight the location of anomalies in addition to viewing the selected base signal, and selected threshold cutoff lines. In all three cases the `Anomalies` may be autolabeled by selecting `Capture Anomalies into Proposed Labels`; at which point labels will appear in "Current Proposed Labels" under the `Auto Labeler` tab, where clicking `Confirm Autolabels for ALL Records` will save them. To export the resulting labels click the `Current Labels` button in the main control panel tab.

1. Moving-Window z-Score Deviation: this method computes, for each time index t, the standardized deviation of the current sample x_t from the mean μ_t of the preceding W samples:

$$z_t = \frac{x_t - \mu_t}{\sigma_t}, \quad \mu_t = \frac{1}{W} \sum_{i=t-W+1}^{t} x_i, \quad \sigma_t = \sqrt{\frac{1}{W-1} \sum_{i=t-W+1}^{t} (x_i - \mu_t)^2}. \quad (1)$$
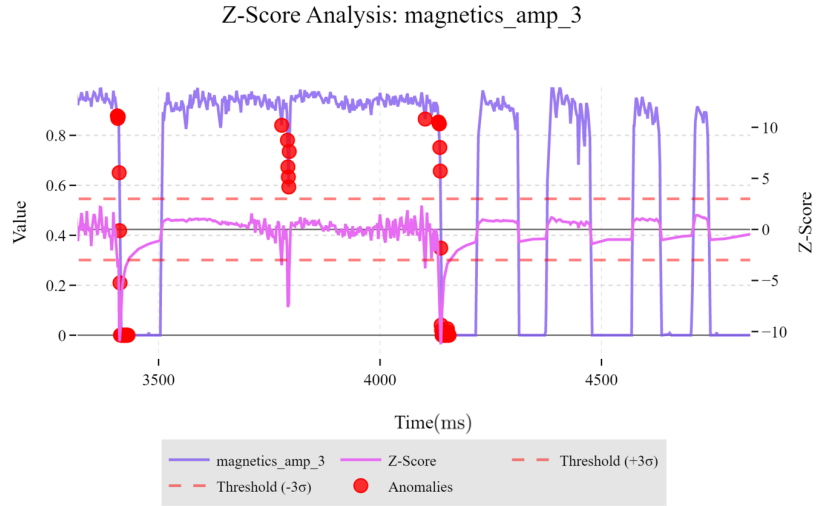
24

An anomaly is flagged whenever $|z_t| > \tau_\sigma$, where τ_σ is the user-specified σ-threshold. *Pros:* Interpretable, computationally efficient, and effective for stationary signals with approximately Gaussian noise. *Cons:* Sensitive to nonstationarity and outliers; performance degrades when the baseline distribution changes rapidly. For a review of Z-score-based detection in time series, see [92].

2. Moving-Window CUSUM Chart: the cumulative sum (CUSUM) method detects small, persistent shifts in the mean of a process by recursively computing:

$$C_t^+ = \max\left[0, C_{t-1}^+ + (x_t - \mu_0 - k)\right], \tag{2}$$
$$C_t^- = \max\left[0, C_{t-1}^- + (\mu_0 - x_t - k)\right], \tag{3}$$

where μ_0 is the nominal process mean (estimated from the window), k is the reference value controlling sensitivity, and C_t^+/C_t^- track positive and negative mean shifts, respectively. An alarm is raised when $C_t^+ > h$ or $C_t^- > h$, where h is the user-specified threshold. *Pros:* Highly sensitive to small, sustained changes; robust for quality-control-type monitoring. *Cons:* Requires a good estimate of the nominal mean; less effective for highly transient or oscillatory events. For applications and theory, see [93, 94].

3. Moving Average with Confidence Intervals: this method computes a rolling mean μ_t over a window W and constructs a $(1 - \alpha)$ confidence interval assuming approximate normality:

$$\mathrm{CI}_t = \mu_t \pm z_{1-\alpha/2} \cdot \frac{\sigma_t}{\sqrt{W}}, \tag{4}$$

where $z_{1-\alpha/2}$ is the quantile of the standard normal distribution corresponding to the desired CI level (user-specified), and σ_t is the rolling standard deviation. Points outside CI_t are flagged as anomalies. *Pros:* Statistically rigorous, interpretable bounds; adjustable false positive rate through α. *Cons:* Assumes approximate normality and independent samples within the window; performance may degrade for autocorrelated or heavy-tailed noise. A statistical foundation for CI-based monitoring can be found in [95].

Role in Fusion Data Pipelines: while each of these methods has distinct strengths, their statistical assumptions (e.g., stationarity, Gaussian noise) may be violated in fusion datasets characterized by multi-regime behavior, non-linearities, and mixed-mode fluctuations. In practice, they can be made more robust by combining them with physics-informed preprocessing (Section 6.2.3) or by embedding them in hybrid expert-in-the-loop pipelines where human operators verify algorithmic detections. As components of dFL, these detectors allow rapid, first-pass identification of candidate events for subsequent expert refinement or ML-driven classification.

6.2.2. Classifier-based Autolabeling

Classifier-based autolabeling leverages supervised machine learning (ML) models to automatically assign labels to incoming data streams, making it particularly powerful when high-quality, expert-labeled datasets already exist or when pre-trained classifiers have been developed for the signals of interest. In dFL such models can be seamlessly integrated through the data ingestion script, enabling automated label generation on both archival datasets and real-time diagnostic feeds.

In dFL the `Plasma Mode Autolabeler` is included under the `Auto Labeler` tab, in the `Select Autolabeler` dropdown menu, as shown in Figure 10. The record may then be autolabeled by simply clicking `Autolable Record`. The resulting labels are stored in the `Current Proposed Labels` section, where confirming the autolabels will save them to the `Main Control Panel` tab for analysis and/or export. This autolabeler is specifically designed to identify plasma modes in DIII-D data, and will be discussed separately in Section 7.2. More broadly, any classification-based autolabeler may be integrated into dFL, which will be discussed in more detail in Section 6.3. Below is a brief overview of types of classification models that can be readily incorporated into dFL's autolabeler options.

Mathematical Formulation: given an input signal segment $\mathbf{x} \in \mathbb{R}^d$ (where d represents the dimensionality of the feature space, which may include time-domain, frequency-domain, or multimodal fusion features), a classifier implements a mapping:

$$f_\theta : \mathbb{R}^d \to \{1, 2, \ldots, K\}, \tag{5}$$

where K is the number of possible classes (labels), and θ denotes the model parameters learned from training data $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$. The predicted class is:

$$\hat{y} = \arg\max_{k} \ p_\theta(y = k \mid \mathbf{x}), \tag{6}$$

where $p_\theta(y = k \mid \mathbf{x})$ is the model's estimated posterior probability for class k. These probabilities can also be used to define confidence scores for downstream filtering or expert review.

Supported Model Types: a wide range of classifiers may be deployed in dFL, including:

- *Linear Models:* Logistic regression, linear discriminant analysis (LDA) – interpretable and efficient, suitable for low-dimensional or linearly separable feature spaces [96].

- *Tree-based Models:* Random forests, gradient boosted trees – robust to heterogeneous features and noise, often requiring minimal feature scaling [97].

- *Kernel Methods:* Support vector machines (SVM) with non-linear kernels – effective for complex decision boundaries in moderate-dimensional spaces [98].

- *Neural Networks:* Fully connected feedforward networks, convolutional neural networks (CNNs) for spatial/temporal patterns, and recurrent neural networks (RNNs) or transformers for sequential dependencies [99, 100].

Pros and Cons: classifier-based autolabeling offers several important advantages. Once trained, such models enable rapid, large-scale labeling, making it feasible to process vast quantities of fusion data with minimal human intervention. They can capture complex, multi-dimensional dependencies among features—relationships that are often inaccessible to simple threshold-based or rule-driven methods. Furthermore, the probabilistic outputs of many classifiers provide confidence scores that can be used for probabilistic label validation, enabling human-in-the-loop workflows where experts focus their attention on lower-confidence or ambiguous cases. These strengths, however, are balanced by notable limitations. Achieving high accuracy typically requires substantial quantities of high-quality labeled training data, unless techniques such as transfer learning are employed to adapt models from related domains. Classifiers can also be vulnerable to dataset shift, where performance degrades if diagnostic conditions evolve or the system operates in previously unseen regimes. Finally, compared to physics-informed or purely statistical detectors, many classifiers (particularly deep neural networks) may lack interpretability, making it more challenging to directly link classification decisions to underlying plasma physics.

Role in Fusion Data Pipelines: classifier-based autolabeling is particularly valuable for tasks such as identifying instability types from magnetic coil arrays, classifying plasma shapes from real-time EFIT reconstructions, or tagging operational modes from multiple diagnostics. In hybrid workflows, classifier predictions can serve as initial candidate labels, which are then confirmed or corrected by domain experts or further refined by statistical post-processing (cf. Section 6.2.1). This synergy allows fusion researchers to combine the scalability of ML with the rigor and interpretability of statistical and physics-informed methods.

6.2.3. Physics-informed Autolabeling

Physics-informed autolabeling leverages explicit knowledge of the governing equations, conservation laws, and phenomenological models underlying plasma behavior to identify events, regimes, or structural features of interest directly from raw or preprocessed diagnostics. By embedding first-principles physics into the detection algorithms, these methods provide both interpretability and strong generalization within the operational space of interest [16, 101]. Such approaches can operate on scalar time series, spatially resolved measurements, or volumetric simulation data, using algorithms tuned to detect physically meaningful features.

Common examples include:

- **ELM detection via magnetic fluctuation envelopes:** using physics-motivated thresholds on the amplitude of edge magnetic fluctuations to detect the onset of Edge Localized Modes (ELMs) [102].

- **Sawtooth crash identification:** tracking core temperature profiles or q-profile evolution to flag abrupt changes associated with sawtooth collapses [103].

- **Transport barrier detection:** locating sharp gradients in temperature or density profiles, often associated with H-mode or internal transport barriers, via derivative-based criteria [104].

Table 1: Zero-crossing orders $\mathcal{Z}_n$ available in the Archaieus Autolabeler, their mathematical definitions, and representative use-cases.

Order	Definition	Representative Use-Case
$\mathcal{Z}_0$	$f(t) = 0$	Crossing of reference level (e.g., density perturbations crossing baseline, mode amplitude sign changes).
$\mathcal{Z}_1$	$f'(t) = 0$	Detection of local extrema in diagnostic signals (e.g., peak stored energy before disruption).
$\mathcal{Z}_2$	$f''(t) = 0$	Inflection point detection (e.g., onset of rapid confinement degradation).
$\mathcal{Z}_3$	$f^{(3)}(t) = 0$	Curvature extrema, can signal transition between growth/decay phases of instabilities.
$\mathcal{Z}_4$	$f^{(4)}(t) = 0$	Higher-order changes, useful for detecting subtle pre-cursors in mode chirping or nonlinear saturation.

The main advantage of physics-informed labeling is that it encodes invariances and scaling laws intrinsic to the plasma, making the output robust to diagnostic noise, moderate parameter drifts, and changes in operating regime. However, such methods can be brittle when underlying physics assumptions break down or when phenomena are driven by unexpected mechanisms not covered by the detection model.

The Archaieus Autolabeler: among the physics-informed plugins available in dFL, the *Archaieus Autolabeler* occupies a unique niche by enabling the user to smooth any scalar time-series signal to C^5 continuity before performing feature extraction. This degree of differentiability ensures stable and accurate estimation of up to fourth-order temporal derivatives, even in the presence of moderate diagnostic noise [105, 106]. Once smoothed, the algorithm computes the zero-crossing sets $\mathcal{Z}_n$ for $n = 0, 1, 2, 3, 4$, where

$$\mathcal{Z}_n = \{\, t_i \mid f^{(n)}(t_i) = 0 \,\}$$

denotes the set of times at which the n-th derivative vanishes.

These zero crossings provide a structured hierarchy of dynamical markers that can reveal subtle precursor phenomena, oscillatory regime changes, or shifts in waveform morphology that might otherwise remain undetected. Table 1 summarizes their definitions, physical interpretations, and representative fusion-related use-cases.

Because higher-order derivatives amplify noise, the C^5 smoothing stage is essential: it suppresses spurious oscillations without attenuating physically relevant features at the chosen derivative order. Lower-order zero crossings are generally robust and suited to coarse event detection, while higher-order crossings are best reserved for well-conditioned signals or post-processed simulation outputs where derivative stability is assured.

Physics-informed methods, including tools like the *Archaieus Autolabeler*, thus combine domain knowledge with algorithmic precision, offering interpretable, reproducible, and operationally useful event detection capabilities that remain aligned with the underlying plasma physics.

6.3. Integrating Custom Autolabelers into dFL

The autolabeling system provides a flexible framework for automatically detecting and labeling temporal events in time-series, or sequential, data. The architecture follows a modular design where domain-specific autolabeling functions are registered with a central coordinator that manages data access, parameter handling, and result aggregation. The system supports both single-shot and bulk processing modes, with the ability to apply custom algorithms to multiple data records simultaneously.
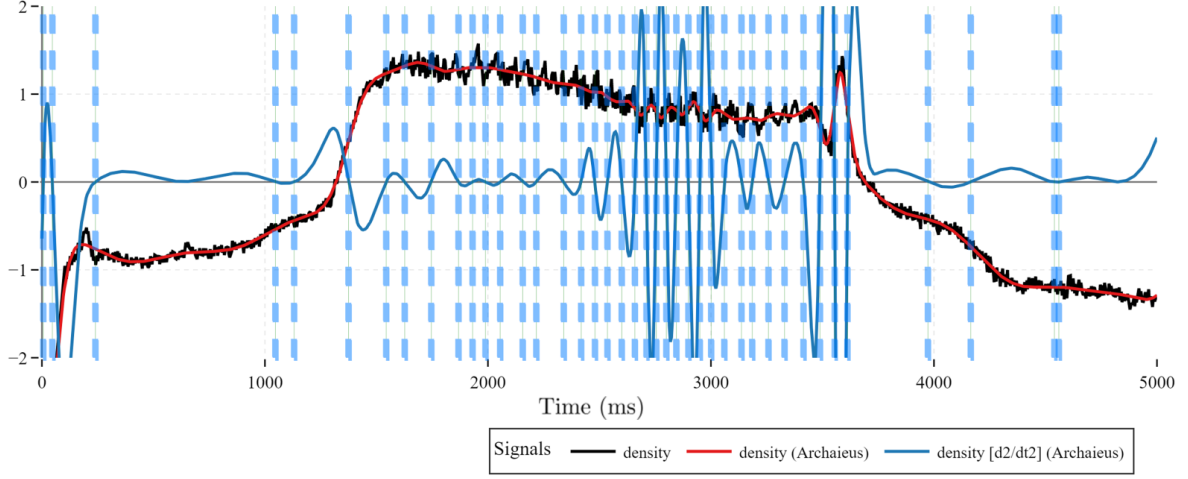
Figure 13: Inflection points automatically detected and selected (e.g., for labeling) with blue-dashed lines using the zero point crossing autolabeler derived from the second derivatives of the plasma density (i.e. $\partial^2 n/\partial t^2 = 0$), after z-score normalizing all then denoising the density using the Archaieus smoother (dimensionless units).

The core autolabeling workflow consists of three main components: (1) a `data coordinator` API that abstracts data access patterns, (2) a registry of autolabeling functions that implement domain-specific detection algorithms, and (3) a callback system that integrates with the web interface for user interaction. Each autolabeling function follows a standardized signature that accepts dataset identifiers, shot IDs, a data coordinator reference, and additional parameters. The system automatically handles parameter validation, data fetching, and result formatting, allowing researchers to focus on implementing their detection logic rather than infrastructure concerns.

The autolabeling functions are defined by the user and can implement various detection strategies, from simple signal processing methods like threshold crossing and peak finding, to more sophisticated algorithms that can use machine learning to classify the data. As an example we show a threshold autolabeling function that detects events by monitoring when signal values cross user-defined thresholds. A code example of how to write such a custom Python autolabeler is presented in Appendix C. Extensive examples and demos are available in the dFL documentation.

7. Fusion Energy Case Studies

Below are a set of example use-cases from DIII-D data provided through the fully integrated TokSearch backend integration. The following Table shows the case studies examined in this section, with a high level overview of how dFL impacts the relevant workflows. Beyond single-shot demonstrations, each workflow can be executed in dFL's bulk mode over $\mathcal{O}(10^6)$ discharges in the DIII-D archive via TokSearch-backed iterators, producing machine-actionable labels and summary statistics that enable systematic indexing, search, and meta-analysis across campaigns. These labels can also be exported in IMAS/OMAS schema for cross-device training and benchmarking, thereby supporting multi-facility generalization.

7.1. Case Study: Automating the Labeling of Edge-Localized Modes (ELMs)

The dataset for this case study is sourced from the DIII-D National Fusion Facility and retrieved via the TokSearch interface [68] and Section 5.2.1, which provides structured, programmatic access to large volumes of experimental signals. For post-discharge ELM identification, the primary diagnostic is the filterscope system, which measures spectrally filtered D_α emission from the plasma edge [107]. These measurements offer high temporal resolution and strong sensitivity to the rapid light bursts associated

Summary of Fusion Energy Case Studies

Case Study	Methodology	Key Challenges	Efficiency Gain
Automated ELM Identification	Moving-window z-score and slope-based outlier detection applied to filterscope D_α signals.	Variability in amplitude/shape, diagnostic saturation, baseline drift, high-frequency noise.	Reduced inspection from hours per shot to minutes in DIII-D ELM detection pilot project; >50X faster than manual labeling with dFL.
Manual Plasma Mode Labeling	Expert-in-the-loop classification using magnetics, filterscopes, BES, and pedestal widths.	Requires expert judgment; labor-intensive inspection of multimodal diagnostics; subjectivity in borderline cases.	Baseline for comparison; typically days of expert time to label $\mathcal{O}(100)$ discharges. dFL improves by 5–8X.
Autolabeling of Plasma Modes	XGBoost multiclass classifier trained on ~792k samples from 360 shots.	Differentiating QH, BBQH, WPQH regimes; class imbalance; subtle mode signatures; reproducibility.	Classifies hundreds of discharges in minutes; ~ 50X faster than expert manual classification with dFL.

with ELM events, making them well-suited for automated detection. Key challenges include shot-to-shot variability in signal amplitude and waveform shape due to changes in plasma configuration, fueling, and control schemes, as well as occasional diagnostic saturation, baseline drift, and data dropouts. Manual labeling can be subjective and inconsistent across operators, motivating the development of automated threshold-based and machine learning-assisted labeling workflows to provide consistent, reproducible identification across large datasets.

Several recent studies provide robust, peer-reviewed benchmarks for automated ELM identification. OâĂŹShea et al. demonstrated a parameter-free statistical algorithm for detecting ELMs in DIII-D filterscope data, achieving over 97% precision and recall across hundreds of discharges [108]. In parallel, Song et al. developed a convolutional neural networkâĂŞbased detector for KSTAR that achieved comparable accuracy without hyperparameter tuning [109].

The present dFL implementation does not aim to replace these established algorithms but rather to integrate them within a unified, provenance-aware framework that also supports statistical and slope-based detectors. This enables consistent execution and comparative benchmarking across devices and diagnostics. A formal benchmark of dFLâĂŹs built-in ELM detectors against these state-of-the-art approaches is planned as future work.

At scale, the same detectors can be swept over the full DIII-D archive to produce per-shot ELM corpora (onset/offset times, waiting-time distributions, burst statistics) stored with provenance. These labels support archive-wide queries (e.g., "find shots with high ELM frequency at fixed q_{95}âĂŹâĂŹ), enable data-balanced training sets for disruption/ELM predictors, and systematically organize understudied discharges rather than focusing on a few canonical shots. Beyond local usage, the same corpora can be exported in IMAS/OMAS schema to seed cross-device generalization (e.g., training predictors simultaneously on DIIIâĂŞD and NSTXâĂŞU), and to align experimental ELM intervals with synthetic diagnostics from nonlinear MHD simulations for model validation.

To autolabel ELMs using filterscope data we have a number of options natively supported in dFL. The simplest (and surprisingly effective approach) is to utilize a statistics-based autolabeler. More specifically, using the moving-window z-score deviation detector from Section 6.2.1, we can easily identify ELMs as simple statistical anomalies of the filterscope signal. Figure 14 shows that all ELMs are identified as statistical outliers of the signal. The remaining challenge, in order to automate the labeling over all ELMy shots then becomes how to set the statistical autolabelerâĂŹs free parameters `Window Size` and standard deviation `Threshold` over all shots of interest *a priori*. If this cannot be easily accomplished, then an *expert-in-the-loop* system can be performed, allowing the user to set these parameters independently on each shot (i.e. still much faster than manually labeling each ELM). In bulk execution, dFL supports parameter grids (or adaptive parameterization via robust statistics of each shot) and writes out per-shot quality metrics so that archive-wide sensitivity analyses are possible. For real-time application, the same z-score detector can be implemented causally (one-sided window) with bounded latency and deployed as a lightweight module running alongside the Plasma Control System (PCS). This module would stream real-time event trigger signals (e.g., imminent ELM onset) that the PCS can subscribe to, enabling automated responses such as pellet pacing or RMP coil actuation, or alternatively provide
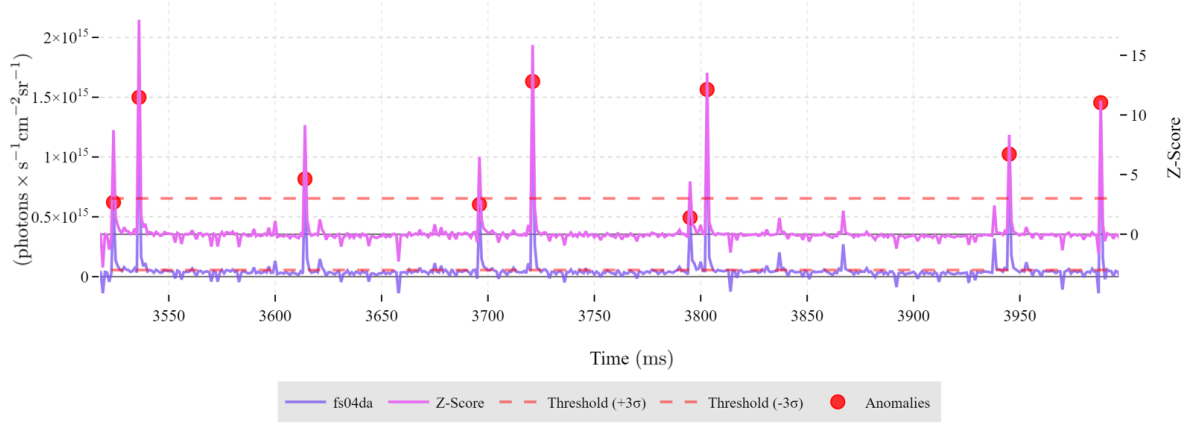
Figure 14: Automatically Identifying ELMs from filterscope data using the simple built-in z-score deviation detector from Section 6.2.1, setting `Window Size`=1000 and `Threshold`=3 standard deviations, shot #149092.

a feature stream for model predictive control (MPC) that incorporates predicted ELM likelihood into constraint handling. Additional downstream applications include mining rare-event statistics (e.g., compound bursts or compound ELM-RMP interactions), generating benchmark datasets for ML training, and feeding uncertainty-aware ELM probability traces into Bayesian fusion frameworks that propagate label uncertainty through to disruption forecasts.

7.1.1. Case Study: Automating ELM Labeling with Slope-Based Outlier Detection

An alternative approach to automated ELM identification leverages slope-based outlier analysis of the filterscope $D_\alpha(t)$ signal, and is available in dFL natively or as a script provided as input into the dFL data provider found in the dFL documentation. Previous work by Eldon and Xing [110] demonstrated that rapid variations in filterscope D_α signals could be used to detect and filter ELMs for equilibrium reconstruction. Building on this idea, the method presented here detects events by examining the time derivative of the filtered signal. Large positive spikes in $D_\alpha(t)$ correspond to the onset of an ELM, while sharp negative slopes denote the termination of a burst. These features are statistically evaluated using z-scores of the positive and negative derivative distributions computed over the entire shot, allowing the algorithm to robustly identify start and end points. In this way, slope-based detection complements amplitude-based methods by focusing on the transient growth and crash dynamics that characterize ELM activity, providing an alternative approach that emphasizes slope dynamics rather than amplitude.

Slope-based methods are inherently more sensitive to high-frequency noise, so preprocessing tools are provided to help users improve detection reliability. Low-pass filtering and signal smoothing (mean, median, exponential moving average, or Gaussian) allow users to reduce high-frequency fluctuations before computing slopes, with the low-pass cutoff frequency typically selected by an expert in the loop to balance noise suppression and temporal resolution. Once ELMs are identified, an ELM merging window can be applied to combine closely spaced bursts into a single event. This reflects the physical behavior observed in strongly ELMy plasmas, where regions often produce frequent, small-amplitude events that are best interpreted as part of a single extended ELMing episode rather than independent bursts. Together, these options give users the flexibility to tailor detection to the characteristics of each shot, producing reproducible, interpretable ELM labels while preserving the underlying dynamics captured by slope-based analysis. Full details of the algorithm and code are available in the dFL documentation.

For archive-scale processing, the derivative-based detector can be run in batch with per-shot adaptive smoothing (e.g., bandwidth set by noise floors estimated from non-ELMy intervals), yielding consistent onset/offset annotations across hundreds of thousands of discharges and enabling population-level statistics (e.g., D_α slope distributions conditioned on fueling or q_{95}). Such systematic slope-based catalogs can also be used to construct archive-level scaling laws for ELM crash dynamics and to cross-validate synthetic slope signals from nonlinear MHD simulations, etc.

30

7.2. Case Study: Automated Plasma Mode Identification

The dFL provides the fusion data community with an automated labeling (autolabeler) capability built on an Extreme Gradient Boosting (XGBoost) classifier [111], trained on a curated dataset comprising approximately 792,000 samples drawn from 360 plasma shots. The classifier is designed to identify three distinct advanced tokamak confinement regimes of quiescent H-mode (QH-mode) operation. As widely acknowledged in the magnetically confined fusion community, bursty intermittent disruptive events, known as edge-localized modes (ELMs), eject uncontrollable particle and heat flux, leading to severe damage to plasma-facing components. One of the most promising candidates for ELM-free operation is the quiescent H-mode (QH-mode), which preserves good overall confinement performance while maintaining ELM suppression.

In DIII-D, three types of QH-mode have been identified: (*i*) **QH-mode**—the canonical quiescent H-mode, characterized by the absence of type-I ELMs and sustained edge harmonic oscillations (EHOs) [112, 113]; (*ii*) **BBQH-mode**—broadband QH-mode, exhibiting a broad spectral signature of edge fluctuations rather than a narrow EHO peak, yet retaining the ELM-free condition [114]; and (*iii*) **WPQH-mode**—wide-pedestal QH-mode, obtained by reducing external injected torque and marked by unusually broad pedestal widths in edge temperature and density profiles while maintaining quiescence [115, 116]. These regimes are of particular interest due to their potential for enabling high-performance, ELM-free operation in future reactors. Understanding the entry conditions and key parameters for accessing these regimes is therefore of great importance.

It is also worth noting that while QH-mode remains a leading candidate for sustained ELM-free operation, the expert-in-the-loop methodology developed here is not restricted to this specific regime. A similar workflow in dFL can be extended to other confinement and exhaust-handling states that are of growing interest for reactor design and control optimization. For example, I-mode operation (first identified on Alcator C-Mod) achieves improved energy confinement without triggering ELMs, characterized by a strong edge temperature pedestal but weak particle confinement [117]. Automated and reproducible identification of I-mode intervals using dFL would enable large-scale, cross-device studies of pedestal formation and access conditions, advancing predictive models for next-generation devices. Likewise, systematic labeling of divertor detachment states, which play a central role in managing heat and particle fluxes to plasma-facing components [118], could facilitate data-driven training of control algorithms for advanced exhaust scenarios. Together, these extensions emphasize that dFL provides a generalizable infrastructure for supervised regime identification and expert-in-the-loop training across confinement and detachment statesâĂŤnot solely for QH-mode classification.

Manual labeling of QH-mode discharges in DIII-D has been performed using the extensive suite of diagnostics, and forms the basis of the experimental QH database covering 2012–2018. The labeling procedure is as follows:

1. Verify plasma density, current, and magnetic field to confirm a valid plasma shot in H-mode;

2. Inspect D_α signals from filterscopes to identify periods with ELM activity;

3. Check the pedestal width and compute the ratio between the measured pedestal width and the conventional pedestal width scaling [58]. A ratio above 1.25 is labeled as WPQH;

4. If the pedestal width ratio is lower than 1.25, analyze magnetics data to determine the presence of coherent harmonic modes and confirm edge localization with other diagnostics, such as Beam Emission Spectroscopy (BES). If an edge harmonic oscillation (EHO) is confirmed, the discharge is labeled as QH-mode; otherwise it will be labelled as BBQH;

5. Perform cross-checks with other key signals (e.g., total injected torque, edge rotation) to confirm the consistency of the classification.

The labeled dataset was constructed using an expert-in-the-loop approach, in which experienced plasma physicists manually classified operating regimes from multiple DIII-D diagnostics (including magnetic fluctuations, reflectometry, and pedestal profile data) following the established mode-identification criteria above. This process ensured that the training set reflected accurate, domain-validated mode boundaries, including cases with subtle or borderline signatures.

The machine learning pipeline employs a multiclass classifier using the 'multi:softprob' objective in XGBoost [111], which outputs a probability distribution over the three classes for each sample, with

Figure 15: Automated Plasma Mode Identification on three separate shots/discharges (zoomed in) from DIII-D, all signals feature scaled (Min-Max normalized), where the top (yellow labeled section) is QH mode, the middle (green labeled section) BBQH mode, and the bottom (magenta labeled section) WPQH mode (units dimensionless).

model evaluation based on the multiclass logarithmic loss (mlogloss). Training was performed using a 5-fold GroupKFold cross-validation strategy, grouping by shot identifier to prevent data leakage between training and validation folds. To address moderate class imbalance, balanced class weights were computed with the `compute_class_weight` utility from scikit-learn [119]. Hyperparameters were tuned to promote generalization, including L2 regularization (`reg_lambda`=0.5), L1 regularization (`reg_alpha`=0.1), a maximum tree depth of `max_depth`=5, a minimum split loss (`gamma`) of 0.1, and a `min_child_weight`

of 3. Early stopping was applied to prevent overfitting.

Feature preprocessing included renaming columns into a $\{f_0, f_1, \ldots, f_n\}$ format to ensure compatibility with the Open Neural Network Exchange (ONNX) serialization standard [120], while retaining a mapping to the original physical feature names for interpretability. Final evaluation across the complete dataset, using the best-performing cross-validation model, achieved an overall classification accuracy exceeding 90%, indicating robust performance across diverse plasma conditions. Deployed across the full DIII-D archive, the autolabeler yields a time-indexed catalog of QH/BBQH/WPQH intervals with calibrated class posteriors, enabling large-scale mining of entry/exit conditions, actuator settings, and robustness margins; this moves the community beyond a handful of well-studied shots to statistically grounded conclusions over $\gg 10^6$ discharges. For operationalization, the ONNX-exported classifier can run as a low-latency PCS sidecar that streams the posterior probability of the mode given the most recent diagnostic features $\mathbf{x}_t$, i.e., $p(\text{mode}|\mathbf{x}_t)$, over each control cycle; MPC can then incorporate these posteriors to enforce mode-aware constraints (e.g., avoiding ELM-prone regimes) or to track toward QH targets by adjusting torque, fueling, or RMP settings in closed loop. In addition, probabilistic mode posteriors enable uncertainty-aware control (risk-weighted constraints), support cross-device transfer learning when exported in IMAS schema, and populate archive-level regime maps (e.g., pedestal-width vs. torque scans) that reveal physics trends across campaigns. These datasets can further serve as benchmark corpora for the community, accelerating the development of standardized ML pipelines in fusion.

Taken together, these case studies illustrate the dual role of dFL autolabeling pipelines: (i) enabling archive-scale mining that systematically organizes hundreds of thousands of discharges across DIII-D (and beyond, when exported in schema-compliant formats), and (ii) providing causal, low-latency outputs that integrate directly into PCS and MPC workflows. Beyond immediate classification, they open downstream opportunities for simulation-experiment fusion, rare-event discovery, cross-device benchmarking, uncertainty-aware control, and the creation of standardized benchmark datasets that catalyze community-wide ML development.

8. Discussion

The results presented here demonstrate that data harmonization, fusion, and provenance in fusion energy science arise not only from software design choices but from the underlying physics and engineering constraints of confined plasmas. The diversity of temporal and spatial scales, the heterogeneity of diagnostic and simulation modalities, and the nonstationary nature of operating conditions impose constraints that shape every stage of the information pipeline. In this environment, harmonization is not an optional preprocessing step; it is the means by which raw measurements are transformed into a physically coherent basis suitable for inference, control, and reproducibility.

From a systems perspective, the Data Fusion Labeler (dFL) occupies a strategic position in this pipeline. Its capacity to ingest multimodal, asynchronous data streams, align them in both temporal and spatial coordinates, and normalize them into schema-compliant structures ensures that downstream fusion and labeling operations are physically and statistically consistent. This capability is particularly impactful in scenarios where data imbalance and sparsity of high-value events—such as disruptions, ELMs, or internal transport barrier formation—would otherwise hinder supervised learning or bias automated detection. By embedding uncertainty-aware harmonization early, dFL preserves the subtle signal features that are often diagnostic of rare events, enabling their recovery in subsequent fusion and classification stages.

The integration of automated and physics-informed labeling modules within a provenance-aware framework further addresses two persistent challenges in fusion data science: scalability and reproducibility. Automated labeling pipelines, whether statistical, classifier-based, or simulation-driven, accelerate the curation of large event corpora, while data provenance capture ensures that every label can be traced to its originating signals, processing steps, and algorithmic parameters. This linkage is crucial for scientific auditability; without it, labeled datasets risk becoming "orphaned" from their context, undermining cross-device benchmarking and model transferability. Conversely, when labels are embedded in a harmonized, versioned data context, they can be federated across facilities and campaigns, contributing to a shared, interoperable knowledge base.

The implications extend beyond the immediate case studies. The architectural principles embodied in dFL—operator-order awareness, multimodal synchronization, schema conformity, and integrated provenance—are directly portable to other high-consequence, sensor-rich scientific domains, from climate

modeling to high-energy physics. Within fusion, these capabilities form the connective tissue between advanced diagnostics, real-time control, and predictive modeling, enabling a closed-loop *measure* → *harmonize* → *fuse* → *label* → *learn/control* → *reproduce* cycle. As devices approach reactor-relevant performance, and as campaigns become increasingly multi-institutional, such integrated, provenance-rich data systems will be indispensable for both operational safety and accelerated physics discovery.

In summary, the present work demonstrates that the rigorous, principled treatment of harmonization, fusion, and labeling—supported by a robust provenance model—is not simply good data hygiene, but a scientific necessity in fusion energy research. The dFL provides a concrete, operational realization of these principles—reducing time-to-analysis, improving label quality, and enabling reproducible— producing physics-aligned workflows that will be critical for the data-driven control and optimization of future burning plasma devices.

9. Conclusion

The Data Fusion Labeler (dFL) addresses a fundamental gap in fusion energy research workflows by unifying data harmonization, data fusion, and data provenance into a coherent, operationally deployable system. Modern fusion devices and their multiscale simulations generate petabyte-scale, heterogeneous, and temporally asynchronous datasets whose scientific value cannot be realized without rigorous preprocessing and alignment. By implementing uncertainty-aware harmonization at ingestion, embedding both manual and automated labeling within a schema-compliant architecture, and capturing complete provenance metadata, dFL ensures that downstream analysis, machine learning, and control tasks operate on physically consistent, reproducible data streams.

The case studies presented demonstrate that such integration is not merely a convenience, but a scientific enabler. Automated detection of ELMs, classification of advanced confinement regimes, and simulation-informed event identification all benefit from a pipeline in which alignment, normalization, and label generation are intrinsically linked and version-controlled. This tight coupling reduces time-to-insight from weeks to hours, improves the statistical robustness of predictive models, and facilitates cross-device and cross-campaign comparability.

Looking ahead, the principles embodied in dFL—data harmonization, operator-order awareness, multimodal synchronization, and embedded provenance—are extensible to future fusion facilities and multi-facility collaborations. As burning plasma experiments and reactor prototypes emerge, the demands on data systems will intensify: higher diagnostic densities, greater operational variability, and tighter control loops will require harmonization and labeling at unprecedented scale and speed. The architecture presented here positions dFL to meet these challenges, offering a path toward fully integrated, reproducible, and scientifically transparent fusion informatics pipelines. Beyond its immediate utility, dFL also has the potential to serve as a benchmarking platform for comparing alternative data harmonization and labeling methods, while hosting effectiveness-proven algorithms contributed by the broader fusion community.

In essence, dFL operationalizes the aspirational cycle of *measure* → *harmonize* → *fuse* → *label* → *learn/control* → *reproduce* for the fusion energy community. By doing so, it not only accelerates data-to-insight conversion, but also establishes a durable framework for knowledge retention, model transferability, and collaborative physics discovery in the era of large-scale, data-driven fusion research.

Acknowledgments

Disclaimer

Disclaimer: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for

the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

During the preparation of this work the author(s) used Large Language Model (LLM) tools as collaborative aids (similar to google search) to refine language, enhance clarity, verify consistency, and explore alternative technical framings across multiple disciplinary contexts in fusion energy science and data harmonization in order to edit, improve, discover, and deepen understanding of the presented topics through an iterative process of humanâĂŞmachine co-working. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the published article.

Appendix A. Incorporating Custom Graphs into dFL

The custom graphing system within dFL (introduced in Section 4.1.4) provides a flexible framework for creating domain-specific visualizations that integrate seamlessly with the main labeling application. The architecture follows a plugin-based design where custom graphing functions are registered with a central coordinator that handles parameter management, data access, and user interface generation. The system supports both simple text-based outputs and complex interactive visualizations using Plotly, with automatic parameter validation and dynamic UI generation.

The custom graphing workflow consists of three main components: a registry of graphing functions that implement domain-specific visualization logic, an automatic parameter generation system that creates user interface components based on function specifications, and a callback system that manages the interaction between user inputs and graph updates. Each custom graphing function follows a standardized signature that accepts application control parameters (containing global state like current shot ID, trim values, and theme settings) and function-specific parameters (user-configurable options like multipliers, signal selections, and visualization options).

The system automatically generates user interface components for parameter configuration based on declarative specifications in the function registry. Parameters can be defined with various types (numeric inputs, dropdowns, text fields) and constraints (minimum/maximum values, default values, option lists). The framework supports nested parameter structures, allowing complex configurations with conditional parameter visibility based on user selections. This declarative approach enables rapid development of new visualization types without requiring custom UI code. Here is a Plotly-based visualization function that imitates Seaborn styling. For more information on the `data provider` see Appendix B and the dFL documentation.

```python
def seaborn_plotly_example(app_control_parameters, parameters):
    """
    Generate a Plotly line chart with Seaborn-like styling.
    Args:
        app_control_parameters (dict): Global parameters from the application controller
        parameters (dict): Parameters specific to this graphing function
    Returns:
        plotly.graph_objects.Figure: A Plotly figure object
    """
    # Extract user-configurable parameters with defaults
    multiplier = parameters.get("seaborn_plotly_example_multiplier", 1.0)
    signal_option = parameters.get("seaborn_plotly_example_signal_option", "Temperature"
                                   )
    # Fetch time-series data using the data coordinator
    shot_data = app_control_parameters["data_coordinator"].fetch_data_async(
        app_control_parameters["data_coordinator"].data_folder,
        dataset_id=None,
        shot_id=app_control_parameters["shot_id"],
        signals=[signal_option],
        global_data_params={},
        trim_1=app_control_parameters["trim_t1"],
        trim_2=app_control_parameters["trim_t2"],
        timeout=10,
    )
    # Extract and process signal data
    signal = shot_data["signals"][0]["data"]
    times = shot_data["signals"][0]["times"]
    signal = np.array(signal).flatten() * multiplier
    times = np.array(times).flatten()
    # Create Plotly figure with custom styling
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=times, y=signal, mode='lines',
        line=dict(width=2.5), name=signal_option
    ))
    # Apply theme-aware styling
    fig.update_layout(
        title=dict(text=f"{signal_option} Over Time (Multiplier: {multiplier})"),
        xaxis=dict(title="Time", showgrid=True, gridcolor='rgba(0,0,0,0.1)'),
        yaxis=dict(title=signal_option, showgrid=True, gridcolor='rgba(0,0,0,0.1)'),
```

```
            template="plotly_white" if not app_control_parameters["theme_value"] else "
                                             plotly_dark",
            plot_bgcolor='rgba(0,0,0,0)', paper_bgcolor='rgba(0,0,0,0)'
        )
        return fig
# Function registration in the data provider
custom_grapher_dictionary = {
    "seaborn_plotly_example": {
        "display_name": "Seaborn Plotly Example",
        "parameters": {
            "multiplier": {
                "default": 1.0,
                "min": 0.0,
                "max": None,
                "display_name": "Multiplier"
            },
            "signal_option": {
                "default": "Temperature",
                "options": {
                    "Temperature": "Temperature",
                    "Pressure": "Pressure",
                    "Precipitation": "Precipitation"
                },
                "display_name": "Signal Option"
            }
        },
        "function": seaborn_plotly_example,
    }
}
```

Appendix B. Incorporating Custom Filters into dFL & Advanced Operator Ordering

Here we show the entire pipeline of generating a `data provider` and `utilities` script for a custom python filter showing the non-commutativity of smoothing and downsampling from Section 4.6. Note that custom filters may be performed in sequence in the `data provider`, thus customizing the operator ordering as much as needed, e.g., performing multiple normalizations sequentially.

```
"""
Data Provider Module

This module provides a data provider interface for paper research signals demonstrating
the non-commutative nature of smoothing and downsampling operations.

The provider generates synthetic signals with high-frequency bursts and provides
custom processing functions for:
- Savitzky-Golay smoothing
- Kaiser windowed downsampling
- Pipeline comparison visualization

Key Features:
- Generates synthetic signals with controllable parameters
- Provides custom graphing functions for pipeline comparison
- Supports data processing options (smoothing, downsampling)
- Includes robust error handling and logging
"""

from datetime import timezone
from pathlib import Path
import numpy as np
import pandas as pd
import logging
import traceback
import matplotlib
matplotlib.use('Agg')  # Use non-interactive backend to avoid GUI warnings
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from paper_utilities import (
    matplotlib_to_plotly_base64,
```

```python
    savitzky_golay_smoothing ,
    kaiser_downsample_processing ,
    parse_value ,
    trim_data ,
    generate_synthetic_signal
)
from scipy.signal import resample_poly , savgol_filter


def pipeline_comparison_grapher ( app_control_parameters , parameters ):
    """
    Generate a comparison plot showing both processing pipelines side by side.

    This function demonstrates the non-commutative nature of smoothing and downsampling
    by showing:
    - Pipeline A: Smooth âĘŠ Kaiser Downsample
    - Pipeline B: Kaiser Downsample âĘŠ Smooth

    Args:
        app_control_parameters (dict): Global parameters from the application controller
        parameters (dict): Parameters specific to this graphing function

    Parameters Keys:
        - "downsample_factor": int (default: 10)
        - "beta": float (default: 2.0)
        - "window_length": int (default: 31)
        - "polyorder": int (default: 3)
        - "show_both_plots": bool (default: True)

    Returns:
        plotly.graph_objects.Figure: A Plotly figure object
    """
    print(f"Pipeline comparison parameters: {parameters}")

    # Extract parameters with defaults
    downsample_factor = int(parameters.get("
                                            pipeline_comparison_grapher_downsample_factor
                                            ",
                                            parameters.get("downsample_factor", 10)))
    beta = float(parameters.get("pipeline_comparison_grapher_beta",
                                parameters.get("beta", 2.0)))
    window_length = int(parameters.get("pipeline_comparison_grapher_window_length",
                                       parameters.get("window_length", 31)))
    polyorder = int(parameters.get("pipeline_comparison_grapher_polyorder",
                                   parameters.get("polyorder", 3)))
    show_both_plots = parameters.get("pipeline_comparison_grapher_show_both_plots",
                                     parameters.get("show_both_plots", True))

    # Get the signal data
    print(f"Pipeline Comparison: Fetching Original Signal for shot {
                                            app_control_parameters['shot_id']}")

    try:
        shot_data = app_control_parameters["data_coordinator"].fetch_data_async(
            app_control_parameters["data_coordinator"].data_folder ,
            dataset_id=None ,
            shot_id=app_control_parameters["shot_id"] ,
            signals=["Original Signal"] ,
            global_data_params={} ,
            trim_1=app_control_parameters.get("trim_t1") ,
            trim_2=app_control_parameters.get("trim_t2") ,
            timeout=10 ,
        )
    except Exception as e:
        print(f"ERROR: Pipeline comparison data fetch failed: {e}")
        raise

    if len(shot_data["signals"]) == 0:
        raise ValueError(f"No signals could be fetched for pipeline comparison.
                                            Available signals: {
```

```python
                                                    app_control_parameters['
                                                    data_coordinator'].
                                                    all_possible_signals}")

signal = np.array(shot_data["signals"][0]["data"]).flatten()
times = np.array(shot_data["signals"][0]["times"]).flatten()
N = len(signal)

# Define operators (from the original code)
def smooth(sig):
    """Savitzky-Golay local cubic smoothing (window = window_length samples)."""
    # Ensure window_length is odd and valid
    wl = window_length
    if wl % 2 == 0:
        wl += 1
    if wl > len(sig):
        wl = len(sig) if len(sig) % 2 == 1 else len(sig) - 1
    if wl < 1:
        wl = 1
    po = min(polyorder, wl - 1)
    return savgol_filter(sig, wl, po)

def kaiser_downsample(sig, factor=downsample_factor, b=beta):
    """Downsample by 'factor' using a polyphase FIR whose taps are shaped by a
                                        Kaiser window."""
    return resample_poly(sig, up=1, down=factor, window=('kaiser', b))

# Two non-commuting pipelines
pipe_A = kaiser_downsample(smooth(signal), downsample_factor)  # Smooth âĘŠ
                                        Downsample
pipe_B = smooth(kaiser_downsample(signal, downsample_factor))  # Downsample âĘŠ
                                        Smooth

# Stretch results back to original grid for overlay
def stretch(arr, factor, target_len):
    return np.repeat(arr, factor)[:target_len]

pipe_A_stretched = stretch(pipe_A, downsample_factor, N)
pipe_B_stretched = stretch(pipe_B, downsample_factor, N)

# Create subplots if showing both, otherwise just one
if show_both_plots:
    fig = go.Figure()

    # Plot 1: Pipeline A (Smooth âĘŠ Kaiser Downsample)
    fig.add_trace(go.Scatter(
        x=times,
        y=signal,
        mode='lines',
        line=dict(width=1, color='lightblue'),
        name='Original',
        opacity=0.4,
        yaxis='y1'
    ))

    fig.add_trace(go.Scatter(
        x=times,
        y=pipe_A_stretched,
        mode='lines',
        line=dict(width=2, color='orange'),
        name='Smooth âĘŠ Kaiser Downsample',
        yaxis='y1'
    ))

    # Plot 2: Pipeline B (Kaiser Downsample âĘŠ Smooth)
    fig.add_trace(go.Scatter(
        x=times,
        y=signal,
        mode='lines',
        line=dict(width=1, color='lightblue'),
```

```python
        name='Original (B)',
        opacity=0.4,
        showlegend=False,
        yaxis='y2'
    ))

    fig.add_trace(go.Scatter(
        x=times,
        y=pipe_B_stretched,
        mode='lines',
        line=dict(width=2, color='lightcoral'),
        name='Kaiser Downsample âĘŠ Smooth',
        yaxis='y2'
    ))

    # Update layout for subplots
    fig.update_layout(
        title=dict(
            text="Pipeline Comparison: Order Matters in Signal Processing",
            font=dict(size=16),
            y=0.95
        ),
        xaxis=dict(
            title="Time [s]",
            domain=[0, 1]
        ),
        yaxis=dict(
            title="Pipeline A: Smooth âĘŠ Kaiser Downsample",
            domain=[0.52, 1],
            anchor='x'
        ),
        yaxis2=dict(
            title="Pipeline B: Kaiser Downsample âĘŠ Smooth",
            domain=[0, 0.48],
            anchor='x'
        ),
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)',
        template="plotly_white" if not app_control_parameters["theme_value"] else "
                                            plotly_dark",
        margin=dict(l=60, r=20, t=80, b=60),
        height=600
    )
else:
    # Show only Pipeline A
    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=times,
        y=signal,
        mode='lines',
        line=dict(width=1, color='lightblue'),
        name='Original',
        opacity=0.4
    ))

    fig.add_trace(go.Scatter(
        x=times,
        y=pipe_A_stretched,
        mode='lines',
        line=dict(width=2, color='orange'),
        name='Smooth âĘŠ Kaiser Downsample'
    ))

    fig.update_layout(
        title=dict(
            text="Pipeline A: Smooth then Kaiser Downsample",
            font=dict(size=16),
            y=0.95
        ),
```

```python
            xaxis=dict(title="Time [s]"),
            yaxis=dict(title="Amplitude"),
            plot_bgcolor='rgba(0,0,0,0)',
            paper_bgcolor='rgba(0,0,0,0)',
            template="plotly_white" if not app_control_parameters["theme_value"] else "
                                             plotly_dark",
            margin=dict(l=50, r=50, t=80, b=50)
        )

    return fig


def simple_signal_plot(app_control_parameters, parameters):
    """
    A simple signal plot showing the selected signal.

    Args:
        app_control_parameters (dict): Global parameters from the app
        parameters (dict): Parameters specific to this function

    Returns:
        plotly.graph_objects.Figure: A Plotly figure object
    """
    # Try different possible parameter keys
    signal_name = (
        parameters.get("simple_signal_plot_signal_name") or
        parameters.get("signal_name") or
        "Original Signal"
    )

    print(f"Simple Signal Plot: Requesting '{signal_name}' for shot {
                                     app_control_parameters['shot_id']}")

    try:
        shot_data = app_control_parameters["data_coordinator"].fetch_data_async(
            app_control_parameters["data_coordinator"].data_folder,
            dataset_id=None,
            shot_id=app_control_parameters["shot_id"],
            signals=[signal_name],
            global_data_params={},
            trim_1=app_control_parameters.get("trim_t1"),
            trim_2=app_control_parameters.get("trim_t2"),
            timeout=10,
        )
    except Exception as e:
        print(f"ERROR: Failed to fetch signal '{signal_name}': {e}")
        raise

    if len(shot_data["signals"]) == 0:
        # Fallback to the first available signal if the requested one wasn't found
        print(f"Signal '{signal_name}' not found, using fallback: '{
                                     app_control_parameters['
                                     data_coordinator'].
                                     all_possible_signals[0]}'")
        fallback_signal = app_control_parameters["data_coordinator"].
                                     all_possible_signals[0]

        try:
            shot_data = app_control_parameters["data_coordinator"].fetch_data_async(
                app_control_parameters["data_coordinator"].data_folder,
                dataset_id=None,
                shot_id=app_control_parameters["shot_id"],
                signals=[fallback_signal],
                global_data_params={},
                trim_1=app_control_parameters.get("trim_t1"),
                trim_2=app_control_parameters.get("trim_t2"),
                timeout=10,
            )
        except Exception as e:
            print(f"ERROR: Fallback signal fetch failed: {e}")
```

```python
            raise

        signal_name = fallback_signal

    if len(shot_data["signals"]) == 0:
        raise ValueError(f"No signals could be fetched. Available signals: {
                                            app_control_parameters['
                                            data_coordinator'].
                                            all_possible_signals}")

    signal = np.array(shot_data["signals"][0]["data"]).flatten()
    times = np.array(shot_data["signals"][0]["times"]).flatten()

    # Create Plotly figure
    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=times,
        y=signal,
        mode='lines',
        line=dict(width=2),
        name=signal_name
    ))

    fig.update_layout(
        title=dict(
            text=f"{signal_name} Over Time",
            font=dict(size=16),
            y=0.95
        ),
        xaxis=dict(title="Time [s]"),
        yaxis=dict(title="Amplitude"),
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)',
        template="plotly_white" if not app_control_parameters["theme_value"] else "
                                            plotly_dark",
        margin=dict(l=50, r=50, t=80, b=50)
    )

    return fig


def get_provider(_):
    """
    Get the complete data provider configuration for the Paper App.

    This function sets up and returns a comprehensive configuration dictionary
    that defines the entire data provider interface for demonstrating signal
    processing pipeline comparisons.

    Args:
        _ (Any): Unused parameter for compatibility with the interface.

    Returns:
        dict: A dictionary containing the full provider configuration.
    """
    # --- Logging Setup ---
    log_file_path = Path(__file__).parent / "paper_data_provider_errors.log"
    logger = logging.getLogger(__name__)
    logger.setLevel(logging.ERROR)

    # Prevent duplicate handlers if get_provider is called multiple times
    if not logger.handlers:
        file_handler = logging.FileHandler(log_file_path)
        formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(
                                            message)s')
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)
    # --- End Logging Setup ---
```

```python
try:
    def fetch_shot_ids_for_dataset_id(data_folder, _=None):
        """
        Fetch all available shot IDs from the data folder.

        For the paper app, we provide synthetic shot IDs.
        """
        # Return some synthetic shot IDs
        return ["synthetic_burst_signal", "synthetic_clean_signal", "
                                          synthetic_noisy_signal"]

    def fetch_data(data_folder, _dataset_id, shot_id, signals, _global_data_params,
                                          data_trim_1=None, data_trim_2=None):
        """
        Fetch and process data for a specific shot ID and signals.

        Generates synthetic signals based on the shot_id parameters.
        """
        if shot_id is None:
            return []
        if signals is None:
            signals = ["original_signal"]

        # Generate different synthetic signals based on shot_id
        if shot_id == "synthetic_burst_signal":
            signal, times = generate_synthetic_signal(N=5000, seed=1)
        elif shot_id == "synthetic_clean_signal":
            # Clean signal without noise
            np.random.seed(1)
            times = np.linspace(0, 10, 5000)
            signal = np.sin(2 * np.pi * 0.6 * times)
            burst_mask = (times > 4.0) & (times < 4.5)
            signal += burst_mask * 5 * np.sin(2 * np.pi * 12 * times)
        elif shot_id == "synthetic_noisy_signal":
            # Very noisy signal
            np.random.seed(42)
            times = np.linspace(0, 10, 5000)
            signal = np.sin(2 * np.pi * 0.6 * times)
            burst_mask = (times > 4.0) & (times < 4.5)
            signal += burst_mask * 5 * np.sin(2 * np.pi * 12 * times)
            signal += 0.5 * np.random.randn(5000)  # More noise
        else:
            # Default to burst signal
            signal, times = generate_synthetic_signal(N=5000, seed=1)

        # Use raw numerical time values (since is_date=False)
        times_array = times

        data_array = []
        errored_signals = []

        for signal_name in signals:
            if signal_name == "Original Signal":
                record = {
                    "data": signal,
                    "data_name": signal_name,
                    "times": times_array,
                    "units": {"Original Signal": "amplitude"},
                    "dims": ("times",)
                }
                data_array.append(record)
            elif signal_name == "Smooth âĘŠ Kaiser Downsample":
                # Pre-compute Pipeline A result for display
                from scipy.signal import savgol_filter, resample_poly
                smoothed = savgol_filter(signal, 31, 3)
                downsampled = resample_poly(smoothed, up=1, down=10, window=('kaiser
                                          ', 2.0))
                stretched = np.repeat(downsampled, 10)[:len(signal)]
                record = {
                    "data": stretched,
```

43

```python
                    "data_name": signal_name,
                    "times": times_array,
                    "units": {"Smooth âĘŠ Kaiser Downsample": "amplitude"},
                    "dims": ("times",)
                }
                data_array.append(record)
            elif signal_name == "Kaiser Downsample âĘŠ Smooth":
                # Pre-compute Pipeline B result for display
                from scipy.signal import savgol_filter, resample_poly
                downsampled = resample_poly(signal, up=1, down=10, window=('kaiser',
                                                    2.0))
                smoothed = savgol_filter(downsampled, 31, 3)
                stretched = np.repeat(smoothed, 10)[:len(signal)]
                record = {
                    "data": stretched,
                    "data_name": signal_name,
                    "times": times_array,
                    "units": {"Kaiser Downsample âĘŠ Smooth": "amplitude"},
                    "dims": ("times",)
                }
                data_array.append(record)
            else:
                errored_signals.append(signal_name)

    return {"id": shot_id, "signals": data_array, "errored_signals":
                                    errored_signals}


# --- Module Initialization ---
data_folder = Path(__file__).parent / "paper_data"

# Available signals
all_possible_signals = ["Original Signal", "Smooth âĘŠ Kaiser Downsample", "
                                    Kaiser Downsample âĘŠ Smooth"]

# --- Configuration Dictionaries ---
custom_smoothing_dictionary = {
    "savitzky_golay_smoothing": {
        "display_name": "Savitzky-Golay Smoothing",
        "parameters": {
            "window_length": {"default": 31, "min": 3, "max": 101, "display_name
                                    ": "Window Length"},
            "polyorder": {"default": 3, "min": 1, "max": 10, "display_name": "
                                    Polynomial Order"}
        },
        "function": savitzky_golay_smoothing,
    },
    "kaiser_downsample_processing": {
        "display_name": "Kaiser Downsample Processing",
        "parameters": {
            "downsample_factor": {"default": 10, "min": 1, "max": 50, "
                                    display_name": "
                                    Downsample Factor"},
            "beta": {"default": 2.0, "min": 0.1, "max": 10.0, "display_name": "
                                    Kaiser Beta"}
        },
        "function": kaiser_downsample_processing,
    },
}

custom_grapher_dictionary = {
    "pipeline_comparison_grapher": {
        "display_name": "Pipeline Comparison",
        "parameters": {
            "downsample_factor": {"default": 10, "min": 1, "max": 50, "
                                    display_name": "
                                    Downsample Factor"},
            "beta": {"default": 2.0, "min": 0.1, "max": 10.0, "display_name": "
                                    Kaiser Beta"},
            "window_length": {"default": 31, "min": 3, "max": 101, "display_name
                                    ": "Window Length"},
```

```python
                            "polyorder": {"default": 3, "min": 1, "max": 10, "display_name": "
                                                            Polynomial Order"},
                            "show_both_plots": {"default": True, "display_name": "Show Both
                                                            Pipelines"}
                    },
                    "function": pipeline_comparison_grapher,
                },
                "simple_signal_plot": {
                    "display_name": "Simple Signal Plot",
                    "parameters": {
                        "signal_name": {
                            "default": "Original Signal",
                            "options": {
                                "Original Signal": "Original Signal",
                                "Smooth âĘŠ Kaiser Downsample": "Pipeline A Result",
                                "Kaiser Downsample âĘŠ Smooth": "Pipeline B Result"
                            },
                            "display_name": "Signal to Plot"
                        }
                    },
                    "function": simple_signal_plot,
                },
            }

            custom_normalizing_dictionary = {}

            # --- Final Provider Assembly ---
            data_coordinator_info = {
                "fetch_data": fetch_data,
                "dataset_id": "paper_signals",
                "fetch_shot_ids_for_dataset_id": fetch_shot_ids_for_dataset_id,
                "all_possible_signals": all_possible_signals,
                "custom_smoothing_options": custom_smoothing_dictionary,
                "custom_normalizing_options": custom_normalizing_dictionary,
                "spline_path": "spline_parameters.csv",
                "auto_label_function_dictionary": {},
                "all_labels": ["Burst", "Clean", "Noise"],
                "custom_grapher_dictionary": custom_grapher_dictionary,
                "is_date": False,
                "trim_data": trim_data,
                "data_folder": data_folder,
                "layout_options": {}
            }
            return data_coordinator_info
        except Exception as e:
            logger.error("Exception occurred in get_provider:")
            logger.error(traceback.format_exc())
            raise  # Re-raise the exception to be caught by the caller
```

```python
"""
Paper App Utilities

This module contains utility functions for signal processing used in the paper app,
including the signal processing pipelines from the research paper:
- Savitzky-Golay smoothing
- Kaiser windowed downsampling
- Pipeline comparison functions

The app demonstrates the non-commutative nature of smoothing and downsampling operations
                                                            .
"""

import numpy as np
import pandas as pd
from scipy.signal import resample_poly, savgol_filter
from datetime import timezone
import matplotlib
matplotlib.use('Agg')  # Use non-interactive backend to avoid GUI warnings
import matplotlib.pyplot as plt
import plotly.graph_objects as go
```

```python
import io
import base64
from pathlib import Path


def parse_value(value):
    """
    Parse a value to its most appropriate data type (datetime, float, int, or original).

    This function attempts to convert input values in the following priority order:
    1. Datetime (with automatic unit detection for timestamps)
    2. Float
    3. Integer
    4. Original value (if all conversions fail)

    Args:
        value: Input value to parse (can be string, number, or datetime)

    Returns:
        Parsed value in most appropriate type, or None for empty strings
    """
    # First, try to parse as datetime
    try:
        # Determine unit based on timestamp length
        if isinstance(value, (int, float)):
            timestamp_str = str(value)
            length = len(timestamp_str)

            if length == 10:
                unit = "s"  # Seconds
            elif length == 13:
                unit = "ms"  # Milliseconds
            elif length == 16:
                unit = "us"  # Microseconds
            elif length == 19:
                unit = "ns"  # Nanoseconds
            else:
                # Default to milliseconds if unsure
                unit = "ms"

            # Use UTC timezone to avoid timezone conversion issues
            dt = pd.to_datetime(value, unit=unit, utc=True)
        else:
            dt = pd.to_datetime(value, utc=True)  # For strings or already datetime
                                                  #              objects

        if pd.isna(dt):  # Check if the result is NaT
            raise ValueError
        return dt.to_pydatetime()
    except (ValueError, TypeError):
        pass

    # If not int, try to convert to float
    try:
        return float(value)
    except (ValueError, TypeError):
        pass

    # If not datetime, try to convert to int
    try:
        return int(value)
    except (ValueError, TypeError):
        pass

    # If all conversions fail, return the original value
    if isinstance(value, str) and len(value) == 0:
        return None
    return value
```

```python
def find_nearest_datetime_index(arr, target_datetime):
    """
    Find the index of the nearest datetime in an array to a target datetime.

    Args:
        arr (array-like): Array of datetime values to search
        target_datetime (datetime): Target datetime to find nearest match for

    Returns:
        int or None: Index of nearest datetime, or None if array is empty or error
                                                occurs
    """
    if target_datetime is None:
        return None

    # Convert array to numpy array if it's not already
    arr = np.array(arr)

    if len(arr) == 0:
        return None

    # Check if the array contains timezone-aware datetimes
    try:
        sample_time = pd.Timestamp(arr[0])
        arr_has_timezone = sample_time.tzinfo is not None
    except:
        arr_has_timezone = False

    # Check if target datetime has timezone
    target_has_timezone = hasattr(target_datetime, "tzinfo") and target_datetime.tzinfo
                                        is not None

    # Make timezone consistent between array and target
    if target_has_timezone and not arr_has_timezone:
        # Remove timezone from target to match array
        target_datetime = target_datetime.replace(tzinfo=None)
    elif not target_has_timezone and arr_has_timezone:
        # Add UTC timezone to target to match array
        target_datetime = target_datetime.replace(tzinfo=timezone.utc)

    # Convert target_datetime to nanoseconds since epoch
    try:
        target_ns = int(target_datetime.timestamp() * 1e9)
    except (AttributeError, TypeError, ValueError) as e:
        print(f"Error converting target datetime to timestamp: {e}")
        return None

    try:
        # Convert datetime64 array to nanoseconds since epoch
        arr_ns = arr.astype("datetime64[ns]").astype(np.int64)

        # Find the index of the nearest value
        nearest_index = np.abs(arr_ns - target_ns).argmin()
        return nearest_index
    except Exception as e:
        print(f"Error finding nearest datetime index: {e}")
        return None


def savitzky_golay_smoothing(_shot_name, _signal_name, raw_signal, _times, parameters):
    """
    Apply Savitzky-Golay smoothing to a 1D array.

    This function smooths data using a Savitzky-Golay filter which fits local
                                        polynomials
    to preserve features better than simple moving averages.

    Args:
        _shot_name (str): The name of the shot being analyzed (unused)
        _signal_name (str): The name of the signal (unused)
```

```
    raw_signal (array-like): Input data to smooth
    _times (array-like): Time array of the shot (unused)
    parameters (dict): Dictionary containing smoothing parameters

Parameters Dictionary Keys:
    - "window_length": int, Window size for Savitzky-Golay filter (default: 31, must
                                            be odd)
    - "polyorder": int, Polynomial order for fitting (default: 3)

Returns:
    numpy.array: Smoothed data with same length as input
"""
# Convert input to numpy array
data = np.array(raw_signal)

# Extract parameters with defaults
window_length = int(parameters.get("savitzky_golay_smoothing_window_length",
                        parameters.get("window_length", 31)))
polyorder = int(parameters.get("savitzky_golay_smoothing_polyorder",
                        parameters.get("polyorder", 3)))

# Ensure window_length is odd and valid
if window_length % 2 == 0:
    window_length += 1

if window_length < 1:
    window_length = 1

if window_length > data.shape[0]:
    window_length = data.shape[0] if data.shape[0] % 2 == 1 else data.shape[0] - 1
    if window_length < 1:
        window_length = 1

# Ensure polyorder is valid
if polyorder >= window_length:
    polyorder = window_length - 1

if polyorder < 0:
    polyorder = 0

# Apply Savitzky-Golay filter
try:
    smoothed = savgol_filter(data, window_length, polyorder)
except ValueError as e:
    print(f"Savitzky-Golay filter error: {e}")
    # Fallback to original data if filtering fails
    smoothed = data

return smoothed


def kaiser_downsample_processing(_shot_name, _signal_name, raw_signal, times, parameters
                                            ):
    """
    Apply Kaiser windowed downsampling to a 1D array.

    This function downsamples data using a polyphase FIR filter with Kaiser windowing.
    The low beta value deliberately relaxes stop-band attenuation to make aliasing
                                            visible.

    Args:
        _shot_name (str): The name of the shot being analyzed (unused)
        _signal_name (str): The name of the signal (unused)
        raw_signal (array-like): Input data to downsample
        times (array-like): Time array of the shot
        parameters (dict): Dictionary containing downsampling parameters

    Parameters Dictionary Keys:
        - "downsample_factor": int, Downsampling factor (default: 10)
        - "beta": float, Kaiser window parameter (default: 2.0)
```

```python
    Returns:
        tuple: (downsampled_data, downsampled_times)
    """
    # Convert input to numpy arrays
    data = np.array(raw_signal)
    time_array = np.array(times)

    # Extract parameters with defaults
    factor = int(parameters.get("kaiser_downsample_processing_downsample_factor",
                                parameters.get("downsample_factor", 10)))
    beta = float(parameters.get("kaiser_downsample_processing_beta",
                                parameters.get("beta", 2.0)))

    # Ensure factor is valid
    if factor < 1:
        factor = 1

    if factor > data.shape[0]:
        factor = data.shape[0]

    # Apply polyphase FIR downsampling with Kaiser window
    try:
        downsampled_data = resample_poly(data, up=1, down=factor, window=('kaiser', beta
                                                                           ))

        # Downsample times accordingly
        downsampled_times = time_array[::factor][:len(downsampled_data)]

        # If lengths don't match, truncate the longer one
        min_len = min(len(downsampled_data), len(downsampled_times))
        downsampled_data = downsampled_data[:min_len]
        downsampled_times = downsampled_times[:min_len]

    except Exception as e:
        print(f"Kaiser downsampling error: {e}")
        # Fallback to simple decimation
        downsampled_data = data[::factor]
        downsampled_times = time_array[::factor]

    return downsampled_data, downsampled_times


def trim_data(shot_data, trim_1=None, trim_2=None):
    """
    Trim shot data based on time range specifications.

    Args:
        shot_data (dict): Shot data dictionary containing signals
        trim_1 (str, optional): Start time for trimming (various formats accepted)
        trim_2 (str, optional): End time for trimming (various formats accepted)

    Returns:
        dict: Modified shot_data with trimmed signals
    """
    signals = []
    for signal in shot_data["signals"]:
        data = signal["data"]
        times = signal["times"]
        index1 = None
        index2 = None

        # Check if times array has timezone info
        has_timezone = False
        if len(times) > 0:
            try:
                # Convert back to pandas datetime temporarily to check timezone
                sample_time = pd.Timestamp(times[0])
                has_timezone = sample_time.tzinfo is not None
            except:
```

```python
                has_timezone = False

        if trim_1 is not None and len(trim_1) != 0:
            # Use parse_value to handle different timestamp formats
            trim1 = parse_value(trim_1)
            # Make timezone consistent with times array
            if hasattr(trim1, "tzinfo") and trim1.tzinfo is not None:
                if not has_timezone:
                    trim1 = trim1.replace(tzinfo=None)
            else:
                if has_timezone:
                    trim1 = trim1.replace(tzinfo=timezone.utc)

            index1 = find_nearest_datetime_index(times, trim1)

        if trim_2 is not None and len(trim_2) != 0:
            # Use parse_value to handle different timestamp formats
            trim2 = parse_value(trim_2)
            # Make timezone consistent with times array
            if hasattr(trim2, "tzinfo") and trim2.tzinfo is not None:
                if not has_timezone:
                    trim2 = trim2.replace(tzinfo=None)
            else:
                if has_timezone:
                    trim2 = trim2.replace(tzinfo=timezone.utc)

            index2 = find_nearest_datetime_index(times, trim2)

        if index1 is not None and index2 is not None:
            data = data[index1:index2]
            times = times[index1:index2]
        if index1 is not None and index2 is None:
            data = data[index1:]
            times = times[index1:]
        if index1 is None and index2 is not None:
            data = data[:index2]
            times = times[:index2]

        signal["data"] = data
        signal["times"] = times
        signals.append(signal)

    shot_data["signals"] = signals
    return shot_data


def matplotlib_to_plotly_base64(fig_mpl):
    """
    Convert matplotlib figure to base64-encoded PNG for embedding in Plotly.

    Args:
        fig_mpl (matplotlib.figure.Figure): Matplotlib figure to convert

    Returns:
        str: Base64-encoded PNG data URL ready for use in Plotly layout images
    """
    buf = io.BytesIO()
    fig_mpl.savefig(buf, format='png', bbox_inches='tight', dpi=150)
    buf.seek(0)
    img_base64 = base64.b64encode(buf.read()).decode('utf-8')
    buf.close()
    plt.close(fig_mpl)  # Important: close the matplotlib figure to free memory
    return f"data:image/png;base64,{img_base64}"


def generate_synthetic_signal(N=5000, seed=1):
    """
    Generate the synthetic signal with ultrahigh frequency burst as described in the
                                                paper.
```

```
    Args:
        N (int): Number of samples (default: 5000)
        seed (int): Random seed for reproducibility (default: 1)

    Returns:
        tuple: (signal, time_array) where signal is the synthetic data and time_array is
                                            the time values
    """
    np.random.seed(seed)
    x = np.linspace(0, 10, N)

    # Slow sine + short, high freq, high amplitude burst + mild noise
    signal = np.sin(2 * np.pi * 0.6 * x)
    burst_mask = (x > 4.0) & (x < 4.5)
    signal += burst_mask * 5 * np.sin(2 * np.pi * 12 * x)   # 12Hz inside 4-4.5s
    signal += 0.1 * np.random.randn(N)

    return signal, x
```

Appendix C. Incorporating Custom Autolabelers into dFL

Similarly to Appendix B, this autolabeling script can be integrated into the `data provider`. This autolabeler detects and marks events in a signal when its value exceeds a user-specified threshold. It scans the selected time-series, identifies contiguous regions above the threshold, and generates labeled intervals that extend slightly before and after the excursion to capture context. These automatically generated labels can then be used dFL for anomaly tagging, event detection, or downstream model training.

```
def perform_threshold_autolabeling(
    dataset_id, shot_id, data_coordinator, additional_parameters, trim_1=None, trim_2=
                                            None
):
    """Detect events when signal values exceed a threshold."""
    if shot_id is None:
        return []

    # Extract parameters from the configuration
    signals_to_autolabel = additional_parameters["threshold_signals"]
    threshold = additional_parameters["threshold_autolabel_threshold"]

    # Fetch time-series data for the specified signals
    shot_data = data_coordinator.fetch_data_async(
        data_coordinator.data_folder, dataset_id, shot_id,
        signals_to_autolabel, {}, trim_1=trim_1, trim_2=trim_2
    )

    if len(shot_data["signals"]) == 0:
        return []

    signal = shot_data["signals"][0]
    signal_data = signal["data"]
    times = signal["times"]

    # Implement threshold crossing detection
    labels = []
    start = None
    finish = None

    for value_index, value in enumerate(signal_data):
        if value > threshold and start is None:
            start = times[value_index]
        if value < threshold and start is not None:
            finish = times[value_index]
        if start is not None and finish is not None:
            # Create label with temporal bounds and classification
            label = {}
            for possible_label in data_coordinator.all_labels:
```

```
            label[possible_label] = None
        label["Anomaly"] = True
        label["T1"] = pd.Timestamp(start) - pd.Timedelta(seconds=10)
        label["T2"] = pd.Timestamp(finish) + pd.Timedelta(seconds=10)
        labels.append(label)
        start = None
        finish = None


    return labels
```

Appendix D. Preprocessing with Fetch Data

dFL allows for a user to supply their own script to fetch the data given parameters from the dFL GUI, such as the data trim, and the signals requested. The user can additionally make any adjustments they want to in the data, in addition to any custom filters, such as normalizations, smoothings, resamplings, etc. defined independently in the script (e.g. see Appendix B).

The following is an example of taking data from a saved file, in this case a `parquet` file, and fetching additional magnetics data in case the user has requested a signal relevant to a spectrogram.

```
def fetch_data(data_folder, dataset_id, shot_id, signals, global_data_params,
                                    data_trim_1=None, data_trim_2=None):
    """
    Fetch and process data for a specific shot ID and signals.

    This function is the core data loader for the TokSearch offline provider. It
                                    handles:
    1. Loading standard signals from a main parquet file.
    2. Loading specialized 'modespec' probe signals from a separate pickle file.
    3. Applying numerical trimming based on time values (not indices).
    4. Consolidating data into a standardized shot dictionary.

    Args:
        data_folder (Path): Directory containing data files
        dataset_id (str): Dataset ID
        shot_id (str): Unique identifier for the data shot
        signals (list[str]): List of signal names to extract
        global_data_params (dict): Global data parameters for additional project-
                                    wide customizations
        data_trim_1 (float, optional): Start time for data trimming
        data_trim_2 (float, optional): End time for data trimming

    Returns:
        dict: Shot data dictionary with signals and errors.
    """
    if shot_id is None:
        return []
    if signals is None:
        signals = [all_possible_signals[0]] if all_possible_signals else []

    df = pd.read_parquet(Path(data_folder) / f"{shot_id}.parquet")
    main_times = df["times"].to_numpy()
    data_array = []
    errored_signals = []

    for signal in signals:
        # Handle special 'modespec' signals from pickle file
        modespec_signals = ["mpi66m307d", "mpi66m340d", "mpi66m307e", "mpi66m340e"]
        if signal in modespec_signals:
            file_path_probe = Path(data_folder) / "probe_data" / f"spectrum_{shot_id
                                    }.parquet"

            if os.path.exists(file_path_probe):
                probe_df = pd.read_parquet(file_path_probe)
                if signal in probe_df.columns:
                    probe_signal = probe_df[signal]
                    probe_times = np.array(probe_df["times"])
                else:
```

```
                continue

            index_start, index_end = find_trim_indices(
                arr=probe_times, trim_beginning=data_trim_1, trim_end=
                                                        data_trim_2
            )

            times = probe_times[index_start:index_end]
            signal_data = np.array(probe_signal)[index_start:index_end]

            record = {"data": signal_data, "data_name": signal, "times": times,
                                            "units": {}, "dims": ("
                                            times",)}
            data_array.append(record)
        else:
            errored_signals.append(signal)

    # Handle standard signals from main parquet file
    elif signal in df.columns.to_list():
        index_start, index_end = find_trim_indices(
            arr=main_times, trim_beginning=data_trim_1, trim_end=data_trim_2
        )
        signal_data = df[signal].to_numpy()[index_start:index_end]
        times = main_times[index_start:index_end]

        record = {"data": signal_data, "data_name": signal, "times": times, "
                                        units": {}, "dims": ("times"
                                        ,)}
        data_array.append(record)
    else:
        errored_signals.append(signal)

shot_dictionary = {"id": shot_id, "signals": data_array, "errored_signals":
                                    errored_signals}
return shot_dictionary
```

The data could additionally come from a remote SQL table or database source such as TokSearch. The shots that are available for selection from another user-supplied function, in this case, simply the filenames of the files of shot data:

```
def fetch_shot_ids_for_dataset_id(data_folder, dataset_id=None):
    """
    Fetch all available shot IDs from the data folder.

    Shot IDs are derived from parquet filenames (without extension).
    """
    return [file.stem for file in data_folder.glob("*.parquet")]
```

The signals available come from a list supplied to the dictionary that is returned in the `get_provider` method, such that the `get_provider` method returns the following:

```
# --- Final Provider Assembly ---
    data_coordinator_info = {
        "fetch_data": fetch_data,
        "dataset_id": "plasma_shots",
        "fetch_record_ids_for_dataset_id": fetch_shot_ids_for_dataset_id,
        "all_possible_signals": all_possible_signals,
        "all_labels": ["BBQH", "QH", "WPQH", "ELMy H"],
        "spline_path": "spline_parameters.csv",
        "custom_normalizing_options": custom_normalizing_options,
        "custom_smoothing_options": custom_smoothing_options,
        "auto_label_function_dictionary": plasma_mode_autolabeling_dictionary,
        "custom_grapher_dictionary": custom_grapher_dictionary,
        "is_date": False, # Important: This provider uses numerical time, not datetime
        "trim_data": trim_data,
        "data_folder": data_folder,
        "layout_options": layout_options,
        "label_export_hook": label_export_hook,
        "data_export_hook": data_export_hook,
        "manual_label_hook": manual_label_hook,
```

```
    }
    return data_coordinator_info
```

Appendix E. Upsampling in dFL

Upsampling involves inserting additional data points between existing input values using interpolation techniques. Given a discrete set of data points $\{(x_i, y_i)\}_{i=1}^N$, interpolation constructs a continuous function $f(x)$ such that $f(x_i) = y_i$ for known points and estimates intermediate values for unknown points. dFL supports the following interpolation-based methods for *upsampling* and gap-filling. These techniques estimate values at intermediate time points between existing samples, increasing the effective sampling rate or regularizing irregularly sampled data for downstream processing. Each method differs in smoothness, monotonicity guarantees, computational complexity, and available free parameters.

1. **Linear Interpolation** [74] : connects adjacent samples with straight-line segments, producing a continuous but non-differentiable function at the knots.

   - *Advantages*: computationally inexpensive; memory-efficient; never overshoots.
   - *Limitations*: corners at sample points; not suitable for high-curvature data.

2. **Quadratic Interpolation** [121] : fits piecewise quadratic polynomials through consecutive triplets of points, ensuring C^1 continuity.

   - *Advantages*: smoother than linear interpolation with modest computational overhead.
   - *Limitations*: can produce mild overshoot and oscillations in regions of high curvature.

3. **Cubic Spline Interpolation** [122, 123]: constructs C^2-continuous cubic polynomials between points, minimizing global curvature under specified boundary conditions.

   - *Advantages*: very smooth; widely used in scientific computing and graphics.
   - *Limitations*: no monotonicity guarantee; may exhibit ringing near steep gradients.

4. **Piecewise Cubic Hermite Interpolation (PCHIP, Mono-PCHIP)** [124, 125]: uses cubic Hermite polynomials with slope adjustments to preserve monotonicity where present.

   - *Advantages*: shape-preserving; no overshoot; ideal for cumulative or strictly monotonic data.
   - *Limitations*: only C^1-continuous.

5. **Akima Interpolation (Mono-Akima)** [126, 127]: employs locally weighted cubic polynomials with slopes estimated from neighboring intervals; can be made monotonic.

   - *Advantages*: robust to outliers; minimal ringing; suitable for nonuniformly spaced data.
   - *Limitations*: higher computational cost than PCHIP.

Upsampling Method Selection Considerations: the optimal choice of interpolation method used for upsampling depends critically on the smoothness requirements, noise characteristics, and application context. Linear interpolation offers maximum computational efficiency and minimal implementation overhead but may introduce abrupt slope changes that degrade the quality of highly smooth or curvature-sensitive signals. Quadratic and cubic spline methods provide progressively smoother approximations, with cubic splines achieving C^2 continuity at the expense of increased computational cost and potential ringing near steep gradients. Shape-preserving methods such as PCHIP and Akima are preferable when monotonicity is essential or when the data exhibit sharp transitions or outliers. In practice, selecting an appropriate method involves balancing computational constraints against the fidelity requirements of the downstream analysis or control task.

Appendix F. Downsampling in dFL

Downsampling in dFL involves dividing the signal into contiguous *blocks* and replacing each block with a single representative value. dFL provides five methods for downsampling time-series data, described below. In all methods, the first step is to specify the desired number of regularly spaced samples L_{target} in the downsampled signal. This determines the number of output blocks $M = L_{\text{target}}$.

Let the original signal be

$$\mathbf{x} = \{(t_1, x_1), (t_2, x_2), \ldots, (t_N, x_N)\}, \quad t_1 < t_2 < \cdots < t_N.$$

The nominal block size is

$$b_{\text{nom}} = \left\lfloor \frac{N}{M} \right\rfloor,$$

for floor funciton $\lfloor \cdot \rfloor$. To ensure M complete blocks, any remainder samples ($N \bmod M$) are discarded from the *start* of the signal. The remaining samples are split contiguously into

$$B_m = \{(t_{i_{m,1}}, x_{i_{m,1}}), \ldots, (t_{i_{m,b_m}}, x_{i_{m,b_m}})\}, \quad m = 1, \ldots, M,$$

where b_m is the actual number of samples in block m. When the sampling grid is irregular, b_m may vary across blocks even though M is fixed.

All block-based downsampling methods in dFL operate by mapping each block B_m to a single representative value y_m according to a rule determined by the chosen method. All methods use the entire block (b_m) to compute statistics within that block.

dFL supports the following block-based and statistical downsampling techniques:

1. **Block Average** [70, 128]: for each block B_m, compute the arithmetic mean over all b_m samples.

   - *Free parameters*: target length L_{target} of downsampled signal.
   - *Advantages*: simple anti-aliasing for decimation; preserves average energy for white-noise signals.
   - *Limitations*: blurs sharp peaks and transients; sensitive to block boundaries.

2. **Block Max** [129]: for each block B_m, compute the maximum value over a sliding window of length b_m.

   - *Free parameters*: target length L_{target} of downsampled signal.
   - *Advantages*: captures extreme peaks; useful for overload or fault detection.
   - *Limitations*: ignores average behavior; sensitive to outliers.

3. **Block Min** [130]: for each block B_m, compute the minimum value over the window of length b_m.

   - *Free parameters*: target length L_{target} of downsampled signal.
   - *Advantages*: captures extreme valleys; complements Block Max for envelope estimation.
   - *Limitations*: ignores peak behavior; sensitive to negative spikes.

4. **Maximum-Likelihood of Gaussian Mixture** [131–133]: for each block B_m, fit Gaussian mixture models (GMMs) to all b_m samples ($k = b_m$), testing component counts $i \in \{1, \ldots, n_{\max}\}$. Select the optimal model via the Bayesian Information Criterion (BIC) and return the maximum-likelihood value as the representative.

   In more detail, MLE-GMM is a probabilistic downsampling method that estimates a representative value for each block B_m by fitting a Gaussian Mixture Model (GMM) to the blockâĂŹs data and selecting the number of mixture components that best explains the local distribution.

   Given a block

   $$B_m = \{x_{i_{m,1}}, \ldots, x_{i_{m,b_m}}\},$$

   the procedure is as follows:

(a) Fit a Gaussian Mixture Model (GMM) with i components to the b_m samples in the block, with i selected from a predefined set $\{1, 2, \ldots, n_{\max}\}$.

(b) Determine the optimal number of mixture components i_0 by minimizing the Bayesian Information Criterion (BIC):

$$\mathrm{BIC} = -2\log\mathcal{L} + p\log b_m,$$

where $\mathcal{L}$ is the maximized likelihood of the model given the data in B_m, and p is the number of free parameters in the GMM with i components.

(c) Using the selected model with i_0 components, compute the *maximum-likelihood estimate* (MLE) of the distribution:

$$\hat{x}_m = \arg\max_x P(x \mid \theta_{i_0}),$$

where $P(x \mid \theta_{i_0})$ is the probability density function of the fitted GMM with parameters θ_{i_0}.

(d) Assign $\hat{x}_m$ as the representative value for block B_m. Repeating this process for $m = 1, \ldots, M$ produces the downsampled signal $\{\hat{x}_1, \ldots, \hat{x}_M\}$.

The MLE-GMM method is particularly effective for blocks whose local distributions are multimodal or strongly non-normal. By fitting a mixture model and selecting the number of components via BIC, this approach preserves complex statistical characteristics, such as variance, skewness, and clustering structure, that would be lost under simple averaging. Empirical results indicate that reliable fitting requires at least $b_m \geq 30$ samples; for smaller blocks, the method reverts to a block mean.

Although MLE-GMM can deliver a more faithful representation of local structure, it is computationally expensive because it requires running the Expectation-Maximization (EM) algorithm for multiple candidate models per block and is sensitive to initialization. When the local distribution is far from a Gaussian mixture, or when block sizes are small, the method may yield suboptimal results. Nonetheless, for domains where accurate local distribution modeling is essential and sufficient data per block is available, MLE-GMM is a powerful probabilistic downsampling tool.

- *Free parameters*: target length L_{target}; maximum mixture components $n_{\max}$.
- *Advantages*: preserves multimodal structure; adapts complexity to local distribution via BIC.
- *Limitations*: computationally expensive (EM per block); sensitive to initialization; unreliable for $b_m < 30$ (falls back to block mean).

5. **Importance Downsampling** [46, 134]: for each block B_m, the b_m sample points are used to selects values in each block so that chosen statistical central moments (zero, variance, skewness, etc.) match those of the original data and the samples remain close to their original time stamps.

- *Free parameters*: target length L_{target}; set of central moments to preserve, e.g.,

$$\{\mu_1, \mu_2, \ldots, \mu_r\} = \{1, 2, \ldots, r\}.$$

- *Advantages*: explicitly preserves user-selected moments; temporal proximity constraint; adapts to local variability.
- *Limitations*: requires at least $k+1$ samples per block; computationally heavy/expensive (i.e. a nonlinear optimization per block), sensitive to initial guesses.

Downsampling Method Selection Considerations: selecting an appropriate downsampling method in dFL depends on the signalâĂŹs characteristics, the analysis goals, and the downstream use case. For smooth, continuous signals where preserving the average trend is most important (e.g., low-frequency behavior for model fitting), the Block Average method offers a computationally inexpensive, anti-aliasing-friendly choice, but may obscure short-duration peaks. Conversely, Block Max and Block Min are well-suited for envelope extraction, overload detection, or fault monitoring, but will sacrifice statistical representativeness and can exaggerate the influence of single-sample outliers. The Maximum-Likelihood Gaussian Mixture method should be reserved for blocks with sufficient sample count ($b_m \gtrsim 30$) and suspected multimodal structure, as it is computationally expensive and sensitive to initialization, but excels at

preserving distributional shape in complex data. Importance Downsampling provides a principled way to preserve selected statistical central moments while retaining temporal locality, making it valuable for maintaining statistical fidelity in irregularly spaced data; however, its nonlinear optimization cost can be prohibitive for large datasets or real-time pipelines.

In all cases, the user should be mindful of aliasing risks when high-frequency content is present and large downsampling ratios are applied: while some dFL resampling methods implicitly smooth the data, block-based statistical methods without prior low-pass filtering may allow high-frequency components to fold into lower frequencies. In irregular-grid contexts, smoothing prior to downsampling may not be well-defined, so users may instead rely on the `Resample` stage of the pipeline to impose a regular grid before applying explicit smoothing. Care should also be taken when the data contain sharp transients or sparse events of interest—aggressive downsampling can irreversibly lose such features unless a peak-preserving method is chosen.

Appendix G. Smoothing in dFL

Appendix G.1. Moving Average Approaches

Moving averages are among the most widely used smoothing techniques for reducing high-frequency noise and stabilizing time-series signals. Two common variants supported in dFL are the *simple moving average* (SMA) and the *exponentially weighted moving average* (EMA), each with distinct weighting schemes and temporal sensitivities.

Simple Moving Average (SMA): given a sequence $\{x_t\}$, the SMA of window length N is

$$\mathrm{SMA}_t = \frac{1}{N} \sum_{i=0}^{N-1} x_{t-i}.$$

This true sliding window assigns equal weight to all samples within the window and ignores older points. Larger N yields smoother results but increases lag by $(N-1)/2$ samples, while smaller N preserves detail but is more susceptible to noise. Its mathematical transparency and deterministic behavior make it a useful baseline, though uniform weighting can blur sharp features and attenuate peaks.

Exponentially Weighted Moving Average (EMA): the EMA replaces the uniform window with an exponentially decaying weight profile:

$$\mathrm{EMA}_t = \alpha\, x_t + (1 - \alpha)\, \mathrm{EMA}_{t-1}, \quad \alpha = \frac{2}{N+1},$$

where N is the *span* parameter controlling the effective smoothing window. Although the update is a convex combination of the latest value and the previous EMA, unrolling the recursion reveals that past samples receive weights proportional to $(1-\alpha)^k$, which iteratively decay exponentially; i.e., older samples never vanish completely but their influence decays exponentially, imparting a strong recency bias. The EMA offers $O(1)$ memory and CPU cost, making it well suited for streaming or real-time systems. Its primary trade-off lies in tuning: too small an N chases noise, too large an N lags important events.

Comparative Considerations: the SMAâĂŹs fixed-length, equal-weight window is preferable when a stable, interpretable filter is needed for exploratory analysis or when the noise level is relatively stationary. The EMA excels when responsiveness to recent changes is essential, such as in dashboards or real-time tokamak control loops, though its fixed decay rate limits adaptation to seasonal or cyclic structure. Both methods suppress high-frequency variance, but the EMAâĂŹs asymmetry in weighting recent points makes it better at tracking evolving plasma states, whereas the SMA provides a cleaner low-pass baseline.

Appendix G.2. Savitzky-Golay Smoothing

When the objective is to reduce noise without sacrificing fine-scale structure, the *Savitzky-Golay* (SG) filter offers a principled alternative to simple moving averages. Rather than averaging, SG smoothing performs a local polynomial regression within a sliding window of odd length m, fitting a polynomial

of degree $p < m$ to the samples and replacing the central point with the fitted value. Formally, for a window $\{x_{t-m'}, \ldots, x_{t+m'}\}$ with $m = 2m' + 1$, the smoothed value is

$$y_t = \sum_{i=-m'}^{m'} c_i \, x_{t+i},$$

where the coefficients $\{c_i\}$ are precomputed from the least-squares fit of the polynomial basis to the window positions.

The SG filterâĂŹs defining strength is *shape preservation*: by approximating the signal locally with a low-order polynomial, it maintains the heights, widths, and positions of peaks, valleys, and inflection points, even while attenuating high-frequency noise. This makes it particularly effective for signals where higher-order features (e.g., slopes, curvatures, or spectral line shapes) carry critical physical meaning. Additionally, the same fitted polynomial can be analytically differentiated to yield smooth, low-noise first or second derivatives, a capability invaluable in spectroscopy, plasma diagnostics, and other derivative-sensitive analyses.

Key Parameters and Trade-offs

- **Window length** (m): controls the degree of smoothing; larger m yields stronger noise suppression but risks over-smoothing fine features.

- **Polynomial degree** (p): determines the local modelâĂŹs flexibility; higher p captures curvature but increases the risk of oscillatory artifacts in noisy data.

A poor choice of (m, p) can lead to under-smoothing, artificial oscillations, or distortion near signal edges, where incomplete windows require padding or asymmetric fits. SG filters also tend to overshoot across step discontinuities, making them ill-suited for piecewise-constant or heavily quantized data.

The SG filter is well-suited for applications where both noise suppression and local feature fidelity are essential; such as magnetic probe traces, reflectometry profiles, or line-integrated density measurements in tokamaks, especially in the broader context of data harmonization, where retaining physically significant morphology is as important as removing stochastic fluctuations. Its efficiency (coefficients are reusable for a fixed (m, p)) makes it practical for high-throughput or streaming workflows, provided its parameters are tuned to the underlying physical and statistical structure of the data.

Appendix G.3. Convolutional Smoothing

Convolutional smoothing generalizes the moving-average concept by allowing an arbitrary weighting kernel to be applied to the data. Given a discrete signal $\{x_t\}$ and a kernel $h[i]$ of width $2m + 1$ satisfying $\sum_{i=-m}^{m} h[i] = 1$, the smoothed output is

$$y_t = \sum_{i=-m}^{m} h[i] \, x_{t+i}.$$

By choosing different kernel shapes, convolutional smoothing can be tuned to match the statistical and spectral characteristics of the data, making it highly adaptable for diverse signal environments.

A widely used special case is the *Gaussian convolutional filter*, in which the weights follow a discrete Gaussian profile

$$g(i) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{i^2}{2\sigma^2}\right),$$

where σ controls the effective width of the smoothing window. The Gaussian kernelâĂŹs smooth, positive, and rapidly decaying form provides excellent time-domain fidelity with no oscillatory ringing, making it particularly well suited for attenuating high-frequency noise while preserving global signal shape. In practice, σ serves as a single, interpretable scale parameter: larger values yield heavier smoothing, while smaller values retain more fine detail.

Gaussian Convolution Advantages and Trade-offs

- **Minimal artifacts:** no Gibbs ringing, even in the presence of sharp transitions.

- **Edge handling:** reflective or symmetric padding reduces boundary bias.

- **Computational flexibility:** efficient implementations exist both in the direct time domain and via FFT for very long signals.

However, it is worth noting that excessive σ blurs sharp edges and diminishes local detail. Single-scale kernels may also be inadequate for signals with noise spanning multiple characteristic scales, requiring either multi-pass filtering or wavelet-based alternatives.

Gaussian convolution is an excellent choice when the goal is to remove stochastic, high-frequency noise while preserving smooth, continuous structure, particularly in scenarios where physical interpretability of the signalâĂŹs overall morphology is critical. In the context of data harmonization, its non-parametric nature and minimal artifact profile make it an effective first-pass denoiser for heterogeneous signals prior to more specialized processing. Applications span both time-series domains (e.g., diagnostic traces in tokamaks) and spatial domains (e.g., imaging data from diagnostics or simulations), benefiting from its balance between robustness and computational efficiency.

Appendix G.4. Butterworth Low-Pass Filter

The *Butterworth low-pass filter* is a classical digital filter design valued for its maximally flat pass-band response, eliminating ripple in the retained frequency range while providing rapid attenuation beyond the specified cutoff. For a given filter order n and normalized cutoff frequency ω_c, its magnitude response is

$$\left|H(e^{j\omega})\right| = \frac{1}{\sqrt{1 + \left[\frac{\tan(\omega/2)}{\tan(\omega_c/2)}\right]^{2n}}},$$

where $\omega_c = \pi\, \texttt{normal\_cutoff}$ and

$$\texttt{normal\_cutoff} = \frac{\texttt{low\_end\_cutoff}}{0.5\,\texttt{sampling\_freq}}, \quad 0 < \texttt{normal\_cutoff} < 1.$$

Here, `low_end_cutoff` is the user-selected cutoff fraction relative to the Nyquist limit $f_{\text{Nyq}} = \frac{1}{2}\,\texttt{sampling\_freq}$.

In *non-causal* mode—achieved by forward-and-reverse-filtering the data (e.g., via filtfilt-style implementations [135])—the Butterworth filter becomes **zero-phase**, ensuring that all features remain temporally aligned without phase distortion. This property is essential in scientific contexts, such as tokamak diagnostics or experimental waveform analysis, where physical interpretability depends on accurate timing relationships between features.

Advantages and Limitations

- **Flat pass-band:** minimal amplitude distortion for all frequencies below cutoff.

- **Steep attenuation:** rapid suppression of frequencies above cutoff.

- **Zero-phase option:** maintains temporal alignment when run forward and backward.

- **Broad applicability:** widely used in physics, audio engineering, and control systems.

While effective at rejecting out-of-band noise, the Butterworth filter permanently removes spectral content above its cutoff, which may include legitimate high-frequency features if the cutoff is set too low. Edge artifacts may arise in short signals due to filter transients, necessitating windowing or padding prior to application. Moreover, because the cutoff is fixed, the method assumes stationarity in the underlying spectral structure; it is less effective for signals with time-varying noise profiles.

Appendix G.5. Smoothing Method Selection Considerations

Selecting an appropriate smoothing method is a critical step in the *data harmonization* pipeline, as it directly impacts the preservation of physically meaningful structures and the suppression of noise in both experimental and simulated fusion datasets. The optimal choice depends on the statistical properties of the signal, the spectral separation between noise and features, the intended downstream analysis, and computational constraints. The four smoothing techniques discussed above span a spectrum of trade-offs in temporal responsiveness, shape preservation, and frequency selectivity.

Ultimately, smoothing method selection should be guided by:

- **Signal morphology:** Are sharp transitions, peaks, or derivatives physically meaningful and thus in need of preservation?

- **Noise spectrum:** Is noise spectrally separated from the signal, or does it overlap with relevant frequencies?

- **Adaptivity needs:** Should the filter respond rapidly to evolving states (favoring EMA) or enforce stability (favoring SMA or Gaussian)?

- **Downstream requirements:** Will the smoothed data feed into derivative estimation, feature extraction, machine learning models, or cross-diagnostic fusion?

- **Computational constraints:** Is the filtering performed in real-time, on large archives, or within iterative analysis loops?

No single method is universally optimal. In practice, dFL's design encourages an *iterative and data-driven* selection process, leveraging exploratory visualization and quantitative diagnostics to ensure that smoothing serves the dual goals of noise suppression and preservation of the underlying plasma physics.

Appendix H. Normalization in dFL

Appendix H.0.1. Standard Score Normalization

A widely used normalization strategy in both scientific data analysis and machine learning is the *standard score* or *z-score* standardization. Given a raw value x, the standard score is defined as

$$z = \frac{x - \mu}{\sigma},$$

where μ is the mean and σ is the standard deviation of the dataset (or of the specific feature in a multivariate setting). This transformation expresses each measurement in units of standard deviation relative to the feature's mean, thereby producing a rescaled variable with zero mean and unit variance.

From a *data harmonization* perspective, z-score normalization removes arbitrary offsets and scales, enabling direct comparison between heterogeneous variables, even those measured in entirely different units, without distorting their statistical structure. In multimodal fusion datasets, for example, plasma density, temperature, and magnetic field strength can differ by orders of magnitude in their raw numerical values, yet may exhibit comparable relative deviations from their respective means. Standardization equalizes these scales, allowing algorithms to operate on a balanced feature space where no single variable dominates due to its absolute magnitude.

The technique is especially valuable when the distribution of each variable is approximately Gaussian and when relative variability, rather than absolute magnitude, is of primary interest. In such cases, the z-score preserves the shape of the distribution, ensuring that the transformed feature retains its underlying statistical relationships. This is critical for distance-based methods (e.g., k-nearest neighbors), gradient-based optimizers in neural networks, and linear models (e.g., regression, PCA, SVMs), all of which benefit from features that are centered and scaled. In fact, standardization can reduce optimizer ill-conditioning, improve numerical stability, and shorten training convergence time in high-dimensional problems.

However, the z-score is not without limitations. Because μ and σ are global statistics, the transformation is sensitive to outliers, which can skew both the center and the scale, reducing interpretability for the bulk of the data. Moreover, z-scores produce unbounded outputs, which may be suboptimal for

models or activation functions expecting inputs in a fixed range. For streaming or time-evolving datasets, μ and σ must be recomputed or incrementally updated to maintain consistency, adding computational overhead.

Despite these caveats, z-score normalization remains a robust, versatile default for a wide range of fusion data workflows, particularly when aiming to integrate variables from disparate diagnostics into a common, physically meaningful scale. In anomaly detection, the interpretability of z-scores—where values above, e.g., $|z| > 3$ indicate strong deviations from nominal behavior—provides a direct statistical basis for identifying events of interest. Its balance of theoretical grounding, computational simplicity, and interpretive clarity makes it a core element of the *data harmonization* pipeline.

Appendix H.0.2. Feature Scaling (Min-Max Normalization)

An alternative to standardization is *feature scaling*, also known as *Min-Max normalization*, which linearly maps the range of each variable to a fixed interval (in dFL between $[0, 1]$) via:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}},$$

where $x_{\min}$ and $x_{\max}$ denote, respectively, the minimum and maximum observed values of the feature. This transformation preserves the relative ordering of the data while compressing it into a uniform range, ensuring that all features have equal numerical bounds regardless of their original units or magnitudes.

In the context of data harmonization, Min-Max scaling is especially valuable when integrating multi-modal datasets in which variables differ not only in units but also in numerical span. Without rescaling, features with large numerical ranges dominate distance-based measures such as Euclidean or cosine similarity, biasing clustering, regression, or classification models. By mapping all features to a common range, min–max scaling allows algorithms such as k-nearest neighbors, support vector machines, and neural networks to treat each variable with equal weight in geometric computations.

A key strength of Min-Max scaling is its simplicity; preserving the shape of the original distribution (apart from a linear stretch and shift), maintaining interpretability by bounding all values, and producing numerically stable input for optimization in models that are sensitive to input scale, particularly those employing bounded activation functions such as sigmoid or tanh. This boundedness can accelerate convergence in deep networks and improve the conditioning of gradient-based learning.

However, because the transformation depends directly on $x_{\min}$ and $x_{\max}$, it is highly sensitive to outliers, which can cause the majority of the data to be compressed into a narrow subinterval of $[0, 1]$, reducing effective resolution. It also lacks centering, meaning the mean of the transformed variable is not guaranteed to be zero, which may be suboptimal for algorithms that perform best with zero-centered features. In streaming or time-evolving datasets, min–max scaling requires ongoing tracking of extrema to avoid range drift, adding operational complexity. For features with intrinsically small ranges, rescaling may also reduce numerical precision.

When applied to fusion data, Min-Max scaling is well suited for harmonizing diagnostics whose absolute magnitudes differ but where bounded, order-preserving normalization is desired. This is common in preparing inputs for neural network surrogates of tokamak plasma states, where stable, bounded gradients improve training robustness. In such workflows, Min-Max normalization offers a low cost, model-friendly preprocessing step, provided that extreme values are handled appropriately, either through prior outlier mitigation or by adopting more robust scaling variants.

Appendix H.0.3. Multiplicative Scalar Scaling

Multiplicative scalar scaling is the simplest normalization strategy, applying a uniform multiplicative factor to all samples in a dataset:

$$x' = \lambda x,$$

where λ is a user-defined scaling constant. This transformation preserves all relative relationships within the data, ioncluding ratios, ordering, and distributional shape, while shifting the overall magnitude to a desired range or scale. When λ is chosen such that the rescaled values fall within a fixed interval (e.g., $[-1, 1]$ or $[0, 1]$), scalar scaling serves as a lightweight alternative to more complex normalization procedures.

From a physical sciences perspective, multiplicative scaling is particularly relevant when adjusting data to match desired unit conventions or when harmonizing measurements from different sensors that are

already on comparable scales but differ in absolute magnitude. In fusion research workflows, for example, λ might be chosen to convert magnetic field measurements from millitesla to tesla, or to scale diagnostic traces into dimensionless form for nondimensional analysis. Crucially, because the transformation is linear and isotropic, it does not distort temporal or spatial patterns, making it ideal when preserving the full physical context of the waveform or field is essential.

Its computational triviality—requiring only a single multiplication per value—makes scalar scaling well suited for high-throughput, real-time, or streaming environments where processing latency is a constraint. Furthermore, it can be applied uniformly across large, homogeneous datasets without the need to compute statistics such as means, standard deviations, or extrema, avoiding potential biases introduced by nonstationary data.

However, scalar scaling does not address differences in variance between features, does not center the data, and leaves the influence of outliers unchanged. The choice of λ is critical: a poor selection can either fail to address numerical conditioning issues or inadvertently amplify noise. Moreover, in heterogeneous, multimodal datasetsâĂŤcommon in fusion experimentsâĂŤassigning different λ values to each feature can become logistically burdensome and risk introducing inconsistencies across experiments or analysis pipelines.

When applied judiciously, multiplicative scalar scaling is an effective and low-cost normalization method, especially in physics-driven contexts where the preservation of physical units, relative proportions, and geometric structure is paramount. It is best viewed as a targeted tool for rapid magnitude adjustment rather than a comprehensive solution for feature standardization in complex, mixed-scale datasets.

Appendix H.0.4. Box-Cox Transformation (Power Normalization)

The *Box-Cox transformation* is a parametric, power-based normalization technique designed to stabilize variance and reduce skew in strictly positive data. To handle datasets containing zeros or negative values, a constant *shift parameter* δ is added before transformation, ensuring all shifted values are strictly positive. The general form is:

$$x' = \begin{cases} \dfrac{(x + \delta)^{\lambda} - 1}{\lambda}, & \lambda \neq 0, \\ \ln(x + \delta), & \lambda = 0, \end{cases}$$

where x is the original data value, $\delta > -\min(x)$ is chosen so that $x + \delta > 0$ for all samples, and λ is a continuous shape parameter controlling the degree and type of transformation. Special cases include $\lambda = 1$ (identity transform), $\lambda = 0.5$ (square-root), and $\lambda = 0$ (logarithmic). In practice, λ is often estimated via maximum likelihood to make the transformed distribution as close to Gaussian as possible, improving the performance of models that assume normality.

By applying a monotonic, non-linear mapping to $x + \delta$, the Box-Cox method can simultaneously correct for heteroscedasticity (non-constant variance) and reduce skew, producing more symmetric features. This is beneficial for algorithms sensitive to distributional shape, such as linear regression, ARIMA forecasting, and Gaussian-based classifiers. The transformation is invertible, allowing the data to be mapped back to the original physical unitsâĂŤan important property when interpretability or physical meaning must be preserved.

The shift parameter δ extends the applicability of Box-Cox to datasets that would otherwise violate its strict positivity requirement. However, its choice can influence the resulting transformation, particularly in the presence of extreme values. Additionally, λ estimation adds computational overhead, and both λ and δ can be biased by outliers. Because the transformation is non-linear, it may alter local ordering of values, affecting distance-based metrics.

In the context of fusion energy datasets, the Box-Cox transformation with shift is especially valuable for diagnostic signals that exhibit multiplicative noise or heavy-tailed distributionsâĂŤfor example, line-integrated density traces, fluctuation amplitudes, or power spectra with large dynamic ranges. By stabilizing variance and normalizing distribution shape, this method enhances the statistical robustness of downstream analyses, including principal component analysis, surrogate modeling, and anomaly detection.

Appendix H.0.5. Robust Scaling (Median-IQR)

Robust scaling is a normalization technique specifically designed to mitigate the influence of extreme values and non-Gaussian distributions. It centers each feature by its median and scales by its interquartile range (IQR), defined as $Q_3 - Q_1$:

$$x' = \frac{x - \text{median}(x)}{Q_3 - Q_1}.$$

Here, Q_1 and Q_3 denote the first and third quartiles, respectively, so that the transformation expresses each value in âĂIJIQR unitsâĂİ relative to the featureâĂŹs central tendency. Unlike mean-variance scaling, which can be heavily distorted by a small number of extreme observations, robust scaling relies entirely on rank-based statistics that are insensitive to outliers.

This property makes the method highly effective for datasets with heavy tailed distributions, skewness, or significant intermittent spikes; conditions encountered in some experimental fusion diagnostics, where transient electromagnetic interference, sensor drift, or environmental noise can produce erratic high-amplitude readings. By removing the leverage of such extremes, robust scaling can produce stable, comparable feature magnitudes without requiring outlier removal or preprocessing heuristics.

Although robust scaling offers broad distributional adaptability, it does not bound the transformed data, and its units (IQRs) are less intuitively interpretable than standard deviations. In perfectly Gaussian data, z-score normalization achieves slightly lower variance and may yield marginally better statistical efficiency. Moreover, computing exact Q_1 and Q_3 quantiles on massive datasets can be computationally expensive, though streaming quantile approximation methods make it feasible for real-time or high-throughput workflows.

In *data harmonization* pipelines for fusion data, robust scaling is best viewed as a cautious default for mixed-quality or partially vetted datasets—tempering occasional glitches, dropouts, and saturation without letting rare extremes dominate. For most well-calibrated diagnostics (magnetics, interferometry, Thomson, bolometry), which exhibit near-Gaussian, light-tailed noise after standard preprocessing, z-score (or Min-Max for bounded ranges) often yields more interpretable and statistically efficient features. Robust scaling remains useful for burst-prone channels in some contexts (e.g., filterscopes) or during early integration of heterogeneous sources; once signals are model-processed and QCâĂŹd, conventional scaling may be preferred.

Normalization Method Selection Considerations: normalization aligns heterogeneous variables for meaningful comparison, fusion, and modeling. The optimal method depends on distributional shape, outlier presence, and downstream model requirements. All natively supported normalizations are invertible mappings, but new normalizations may be readily integrated into the dFL GUI, as detailed in the dFL documentation. Additionally, normalization may be performed at ingestion using the Fetch Data approach discussed in Appendix D.

When data quality is high and distributions are near-normal, z-score normalization remains a strong default. For bounded output on clean data, Min-Max scaling integrates seamlessly with many machine learning models. Scalar multiplication is optimal for rapid, unit-preserving adjustments across already harmonized features. When variable skewness must be reduced—whether strictly positive or extended to non-positive ranges via a constant shift—the Box-Cox transformation offers a principled, tunable path toward Gaussianization and variance stabilization. In the presence of outliers or heavy-tailed distributions, robust scaling provides stability without distorting relationships within the bulk of the data. In practice, hybrid workflows often combine methods, e.g., robust scaling followed by Min-Max or Box-Cox followed by z-score, tailored to the statistical and physical structure of the dataset, and the requirements of the modeling target. Examples of how to easily implement serial normalizations into dFL are provided in the dFL documentation.

Appendix I. A Note on Units and Axis Labeling

A subtle limitation of the present dFL implementation is the absence of explicit x and y axis unit labeling in its visualization components. While time is generally understood to be the abscissa (the independent variable, usually t, though not necessarily), the ordinate (dependent variable, often y or $f(t)$) remains unlabeled, reflecting the fact that many harmonization operations—particularly normalization and standardization—transform raw measurements into dimensionless or non-intuitive scales.

This practice is commonplace in machine learning and artificial intelligence pipelines, where preserving statistical comparability across heterogeneous feature spaces often supersedes the preservation of absolute units. By contrast, in traditional physics workflows, units are non-negotiable: they are essential for dimensional consistency, physical interpretation, and reproducibility, and errors in units can propagate catastrophically.

A concrete illustration arises when considering electron temperature T_e measured in electronvolts (eV) alongside electron density n_e measured in m^{-3}. Typical core tokamak values might span $T_e \sim 1$–$10\,\mathrm{keV}$ and $n_e \sim 10^{19}$–$10^{20}\,\mathrm{m}^{-3}$. If these raw values are ingested directly into an ML pipeline without normalization, the density feature dominates numerically by more than fifteen orders of magnitude relative to temperature. In effect, the units themselves act as an implicit statistical weight, biasing learning algorithms toward density-driven structure while suppressing temperature variations that may be equally or more relevant for physical inference. Such imbalances reduce the utility of otherwise powerful algorithms, as the optimization landscape becomes warped by unit-induced scaling disparities rather than by underlying correlations or dynamics. Normalization or variance-based scaling thus serves not only as a numerical convenience, but as an essential step in mitigating hidden biases introduced by standard (or conventional) units. In addition, dFL supports multimodal visualizations (in part to further develop intuition and understanding of how data-driven algorithms, for example, see and operate on the underlying data spaces) where signals with disparate base units (e.g., temperatures, magnetic fields, and densities) are shown together on a common axis. In such cases, plotting in absolute units is rarely practical, since signals must be rescaled or normalized to avoid one variableâĂŹs dynamic range obscuring the others, making normalization a prerequisite for multimodal visualization.

Transformation-aware units: not all preprocessing operations treat units equally. Resampling or trimming typically preserves units, while z-scoring or PCA whitening explicitly destroys them. Transformations such as logarithms or Box-Cox mappings produce re-scaled units that remain interpretable, but only if metadata tracking the mappings. Without automated unit propagation, these subtleties can be lost.

Operational and human factors: in practice, units often act as guardrails. Operators and diagnosticians often catch errors (e.g., kiloamps vs. megaamps, keV vs. eV) by their units alone. When stripped away, silent misinterpretations become more likely, especially in archival datasets where future users may not recall the original dimensionality of a signal. Unit metadata thus provides a natural audit trail.

Implications for PCS and control: in real-time plasma control, units are frequently inseparable from actuator guardrails. For example, actuator constraints such as "do not exceed 5 MW of injected ECH power" cannot be expressed in normalized space. Feeding unitless features directly into a plasma control system (PCS) therefore introduces risk unless a unit-aware mapping layer exists. In principle, dFL can serve as such an intermediate layer; where signals may be normalized for ML pipelines while retaining the ability to "round-trip" back into physical units when interfaced with PCS, MPC, or physics validation.

Relation to data fusion mathematics: Bayesian or statistical data fusion explicitly requires noise covariance matrices R_i expressed in physical units, since weights are determined by inverse variances. If signals are unitless or inconsistently scaled, these weights lose physical meaning and produce biased or non-optimal estimators. Correct unit handling is therefore not simply cosmetic, but mathematically essential for heteroscedastic fusion across modalities.

Provenance and metadata standards: the International Atomic Energy Agency's IMAS framework and tools such as OMAS already mandate explicit unit metadata, but these are rarely surfaced into harmonization and visualization layers. For dFL, the lack of unit propagation is thus not a theoretical impossibility but an implementation gap. Proper unit handling requires metadata-rich datasets where units are explicitly stored, propagated, and updated under transformation, including transparent annotations of when units are lost or remapped.

Potential solutions: future dFL extensions plan to integrate unit-handling libraries (e.g., schema-aware extensions of `pint` [136]) to track dimensional consistency automatically. This would enable hybrid visualization modes: (*i*) *absolute mode*, where raw units are preserved; (*ii*) *relative mode*, where all signals are normalized; and (*iii*) *hybrid mode*, where multiple axes or dual scales expose both. Planned unit-aware metadata will also support heteroscedastic data fusion where per-modality variances R_i are interpreted in physical units, while still allowing unit-stripped versions for ML workflows requiring dimensionless feature vectors and multimodal visualization.

For now, if units must be explicitly preserved, the easiest workaround is to simply rename input features to include their base units (e.g., `density_in_m^-3`); or, if needing to preserve unit structures

through transformations, to use a custom implementation of `pint` through the `Fetch Data` field of the dFL `data coordinator`. Howveer, while future dFL releases plan to include automated unit propagation and guardrails against common errors, ultimate responsibility for correct unit management will remain with the practitioner. Even the most sophisticated metadata systems cannot substitute for principled scientific use. Users must remain vigilant in tracking and validating transformations, ensuring dimensional consistency, and interpreting outputs in physically meaningful terms. In this sense, dFL can provide scaffolding and reminders, but it will never be able to fully insulate users from mistakes introduced by careless or unprincipled handling of units.

References

[1] S. J. Zweben, J. Terry, D. Stotler, R. J. Maqueda, Invited review article: Gas puff imaging diagnostics of edge plasma turbulence in magnetic fusion devices, Review of Scientific Instruments 88 (4) (2017).

[2] F. Jenko, W. Dorland, M. Kotschenreuther, B. N. Rogers, Electron temperature gradient driven turbulence, Physics of Plasmas 7 (5) (2000) 1904–1910. doi:10.1063/1.874014.

[3] M. Kotschenreuther, W. Dorland, M. Beer, G. Hammett, Quantitative predictions of tokamak energy confinement from first-principles simulations with kinetic effects, Physics of Plasmas 2 (6) (1995) 2381–2389.

[4] E. Strait, E. Fredrickson, J.-M. Moret, M. Takechi, Chapter 2: magnetic diagnostics, Fusion Science and technology 53 (2) (2008) 304–334.

[5] D. Ponce-Marquez, B. Bray, T. Deterly, C. Liu, D. Eldon, Thomson scattering diagnostic upgrade on diii-d, Review of Scientific Instruments 81 (10) (2010).

[6] L. Bertalot, R. Barnsley, M. Direz, J. Drevon, A. Encheva, S. Jakhar, Y. Kashchuk, K. Patel, A. Arumugam, V. Udintsev, et al., Fusion neutron diagnostics on iter tokamak, Journal of Instrumentation 7 (04) (2012) C04012.

[7] C. Petty, Sizing up plasmas using dimensionless parameters, Physics of Plasmas 15 (8) (2008).

[8] R. Dumont, Magnetic confinement fusion-plasma theory: Heating and current drive, Encyclopedia of nuclear energy (2021).

[9] W. Wehner, M. Lauret, E. Schuster, J. R. Ferron, C. Holcomb, T. C. Luce, D. A. Humphreys, M. L. Walker, B. G. Penaflor, R. D. Johnson, Predictive control of the tokamak q profile to facilitate reproducibility of high-q min steady-state scenarios at DIII-D, in: 2016 IEEE Conference on Control Applications (CCA), IEEE, 2016, pp. 629–634.

[10] J. Vega, A. Murari, S. Dormido-Canto, G. A. Rattá, M. Gelfusa, Disruption prediction with artificial intelligence techniques in tokamak plasmas, Nature Physics 18 (7) (2022) 741–750.

[11] R. Buttery, S. Günter, G. Giruzzi, T. Hender, D. Howell, G. Huysmans, R. La Haye, M. Maraschek, H. Reimerdes, O. Sauter, et al., Neoclassical tearing modes, Plasma Physics and Controlled Fusion 42 (12B) (2000) B61.

[12] C. Ham, A. Kirk, S. Pamela, H. Wilson, Filamentary plasma eruptions and their control on the route to fusion energy, Nature Reviews Physics 2 (3) (2020) 159–167.

[13] C. Michoski, T. A. Oliver, D. R. Hatch, A. Diallo, M. Kotschenreuther, D. Eldon, M. Waller, R. Groebner, A. O. Nelson, A gaussian process guide for signal regression in magnetic fusion, Nuclear Fusion 64 (3) (2024) 035001.

[14] Z. Liu, Y. Huang, M. Wu, Z. Luo, Y. Wang, K. Wu, D. Chen, J. Huang, S. Wang, H. Lian, et al., Plasma electron density profile tomography for east based on integrated data analysis, Nuclear Fusion 64 (12) (2024) 126006.

[15] R. Anirudh, R. Archibald, M. S. Asif, M. M. Becker, S. Benkadda, P.-T. Bremer, R. H. Budé, C.-S. Chang, L. Chen, R. Churchill, et al., 2022 review of data-driven plasma science, IEEE Transactions on Plasma Science (2023).

[16] A. Pavone, A. Merlo, S. Kwak, J. Svensson, Machine learning and Bayesian inference in nuclear fusion research: an overview, Plasma Physics and Controlled Fusion 65 (5) (2023) 053001.

[17] P. W. Hatfield, J. A. Gaffney, G. J. Anderson, S. Ali, L. Antonelli, S. Başeğmez du Pree, J. Citrin, M. Fajardo, P. Knapp, B. Kettle, et al., The data-driven future of high-energy-density physics, Nature 593 (7859) (2021) 351–361.

[18] H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac, et al., Scientific discovery in the age of artificial intelligence, Nature 620 (7972) (2023) 47–60.

[19] S. V. Ryzhkov, Magneto-inertial fusion and powerful plasma installations (a review), Applied Sciences 13 (11) (2023) 6658.

[20] G. D. Conway, A. I. Smolyakov, T. Ido, Geodesic acoustic modes in magnetic confinement devices, Nuclear Fusion 62 (1) (2021) 013001.

[21] S. C. Hoi, D. Sahoo, J. Lu, P. Zhao, Online learning: A comprehensive survey, Neurocomputing 459 (2021) 249–289.

[22] O. Gheibi, D. Weyns, F. Quin, Applying machine learning in self-adaptive systems: A systematic literature review, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 15 (3) (2021) 1–37.

[23] R. M. Churchill, J. Choi, R. Kube, C.-S. Chang, S. Klasky, Machine learning for the complex, multi-scale datasets in fusion energy, in: Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI: 17th Smoky Mountains Computational Sciences and Engineering Conference, SMC 2020, Oak Ridge, TN, USA, August 26-28, 2020, Revised Selected Papers 17, Springer, 2020, pp. 269–284.

[24] D. R. Hatch, C. Michoski, D. Kuang, B. Chapman-Oplopoiou, M. Curie, M. Halfmoon, E. Hassan, M. Kotschenreuther, S. M. Mahajan, G. Merlo, et al., Reduced models for ETG transport in the tokamak pedestal, Physics of Plasmas 29 (6) (2022).

[25] M. S. Osman, A. M. Abu-Mahfouz, P. R. Page, A survey on data imputation techniques: Water distribution system as a use case, IEEE Access 6 (2018) 63279–63291.

[26] K. Olofsson, D. Humphreys, R. La Haye, Event hazard function learning and survival analysis for tearing mode onset characterization, Plasma Physics and Controlled Fusion 60 (8) (2018) 084002.

[27] K. E. J. Olofsson, C. Akcay, X. Sun, B. S. Sammuli, R. Nazikian, Large-scale tearing-mode hazard function analysis with standard matched equilibrium reconstructions, Plasma Physics and Controlled Fusion 67 (6) (2025) 065039. doi:10.1088/1361-6587/ade009.
URL https://dx.doi.org/10.1088/1361-6587/ade009

[28] C. Cheng, et al., A general primer for data harmonization, Scientific Data 11 (152), defines data harmonization broadlyâĂŤcombining heterogeneous datasets for comparability and discusses conceptual, technical, and logistical dimensions (2024).

[29] F. Castanedo, A review of data fusion techniques, The Scientific World Journal 2013 (2013) e531672, open-access review summarizing broad definitions and methods for data fusion, including JDL and Dasarathy frameworks. doi:10.1155/2013/531672.

[30] M. Herschel, R. Diestelkämper, H. B. Lahmar, A survey on provenance: What for? what form? what from?, The VLDB Journal 26 (6) (2017) 881–906, comprehensive review synthesizing provenance concepts across databases, workflows, and applications. doi:10.1007/s00778-017-0486-1.

[31] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al., The fair guiding principles for scientific data management and stewardship, Scientific data 3 (1) (2016) 1–9.

[32] R. D. Peng, Reproducible research in computational science, Science 334 (6060) (2011) 1226–1227.

[33] M. Souibgui, Y. Rjoub, et al., Data quality in etl process: A preliminary study, Procedia Computer Science 151 (2019) 694–701. doi:10.1016/j.procs.2019.04.158.

[34] F. E. White, Data fusion lexicon, Tech. rep. (1991).

[35] D. L. Hall, J. Llinas, An introduction to multisensor data fusion, Proceedings of the IEEE 85 (1) (2002) 6–23.

[36] S. D. Pinches, L. Abadie, K. Affonin, X. Bonnin, M. De Bock, O. e. Hoenen, Integrated modelling & analysis suite: Developments to address iter needs, in: Proceedings of the 28th IAEA Fusion Energy Conference, Nice, France, 2020, iAEAâĂŘCNâĂŘ286/TH/P2âĂŘ22. URL https://nucleus.iaea.org/sites/fusionportal/Shared%20Documents/FEC%202020/fec2020-preprints/preprint1165.pdf

[37] O. Meneghini, S. Smith, L. L. Lao, O. Izacard, Q. Ren, J. M. e. Park, Integrated modeling applications for tokamak experiments with OMFIT, Nuclear Fusion 55 (8) (2015) 083008. doi:10.1088/0029-5515/55/8/083008.

[38] G. Staebler, J. Kinsey, R. Waltz, A theory-based transport model with comprehensive physics, Physics of Plasmas 14 (5) (2007).

[39] J. Kates-Harbeck, A. Svyatkovskiy, W. Tang, Predicting disruptive instabilities in controlled fusion plasmas through deep learning, Nature 568 (2019) 526–531.

[40] M. Xu, D. Mazon, M. Barbarino, W. Biel, R. Churchill, R. Fischer, K. Fujii, P. Jain, A. Murari, S. D. Pinches, et al., Summary of the 5th iaea technical meeting on fusion data processing, validation and analysis (fdpva), Nuclear Fusion (2025).

[41] P. De Vries, G. Arnoux, A. Huber, J. Flanagan, M. Lehnen, V. Riccardo, C. Reux, S. Jachmich, C. Lowry, G. Calabro, et al., The impact of the iter-like wall at JET on disruptions, Plasma Physics and Controlled Fusion 54 (12) (2012) 124032.

[42] S. Wang, Y. Wang, Q. Ma, X. Wang, N. Yan, Q. Yang, G. Xu, J. Tang, Multi-modal fusion based q-distribution prediction for controlled nuclear fusion, arXiv preprint arXiv:2410.08879 (2024).

[43] T. A. Oliver, C. Michoski, S. Langendorf, A. LaJoie, Automated Bayesian high-throughput estimation of plasma temperature and density from emission spectroscopy, Review of Scientific Instruments 95 (7) (2024).

[44] M. Grandini, E. Bagli, G. Visani, Metrics for multi-class classification: an overview, arXiv preprint arXiv:2008.05756 (2020).

[45] I. Lin, O. Loyola-González, R. Monroy, M. A. Medina-Pérez, A review of fuzzy and pattern-based approaches for class imbalance problems, Applied Sciences 11 (14) (2021) 6310.

[46] D. Faghihi, V. Carey, C. Michoski, R. Hager, S. Janhunen, C.-S. Chang, R. D. Moser, Moment preserving constrained resampling with applications to particle-in-cell methods, Journal of Computational Physics 409 (2020) 109317.

[47] Z. Lu, S. Jia, G. Li, S. Jing, Neutron image denoising method based on adaptive new wavelet threshold function, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 1059 (2024) 169006.

[48] M. Farge, K. Schneider, Wavelet transforms and their applications to MHD and plasma turbulence: a review, Journal of Plasma Physics 81 (6) (2015) 435810602.

[49] S. C. Jardin, Review of implicit methods for the magnetohydrodynamic description of magnetically confined plasmas, Journal of Computational Physics 231 (3) (2012) 822–838. `doi:10.1016/j.jcp.2011.03.033`.

[50] I. K. Charidakos, et al., Action principles for extended magnetohydrodynamic models, Physics of Plasmas 21 (9) (2014) 092118. `doi:10.1063/1.4896336`.

[51] C. R. Sovinec, A. H. Glasser, T. A. Gianakon, D. C. Barnes, R. A. Nebel, S. E. Kruger, D. D. Schnack, S. J. Plimpton, A. Tarditi, M. Chu, Nonlinear magnetohydrodynamics simulation using high-order finite elements, Journal of Computational Physics 195 (1) (2004) 355–386. `doi:10.1016/j.jcp.2003.10.004`.

[52] T. Bernard, F. Halpern, M.âĹŔFrancisquez, G. Hammett, A. Marinoni, Plasma edge and scrape-off layer turbulence in gyrokinetic simulations of negative triangularity plasmas, Plasma Physics and Controlled Fusion 66 (1) (2024) 15017. `doi:10.1088/1361-6587/ad8186`.

[53] A. Bottino, A. Stier, M. Boesl, T. Hayward-Schneider, A. Bergmann, D. Coster, S. Brunner, G. D. Giannatale, L. Villard, Particle-in-cell methods in edge plasma physics: the picls code, arXiv preprint.

[54] K. Nordlund, C. BjÃűrkas, T. Ahlgren, A. Lasa, et al., Multiscale modelling of plasmaâĂŞwall interactions in fusion reactor conditions, Journal of Physics D: Applied Physics 47 (22) (2014) 224018. `doi:10.1088/0022-3727/47/22/224018`.

[55] A. Lasa, S. Blondel, M. A. Cusentino, B. D. Wirth, et al., Development of multi-scale computational frameworks to solve fusion materials science challenges, Journal of Nuclear Materials 594 (2024) 154003. `doi:10.1016/j.jnucmat.2024.154003`.

[56] S. Wiesen, M. Groth, M. Wischmeier, S. Brezinsek, A. Järvinen, F. Reimold, L. Aho-Mantila, J. Contributors, E. M. Team, A. U. Team, A. C.-M. Team, Plasma edge and plasma-wall interaction modelling: Lessons learned from metallic devices, Nuclear Materials and Energy 12 (2017) 3–17. `doi:10.1016/j.nme.2017.03.033`.

[57] P. Robbe, S. Blondel, T. A. Casey, A. Lasa, K. Sargsyan, B. D. Wirth, H. N. Najm, Global sensitivity analysis of a coupled multiphysics model to predict surface evolution in fusion plasmaâĂŞsurface interactions, Computational Materials Science 226 (2023) 112229. `doi:https://doi.org/10.1016/j.commatsci.2023.112229`.
URL `https://www.sciencedirect.com/science/article/pii/S0927025623002239`

[58] P. Snyder, R. Groebner, J. Hughes, T. Osborne, H. Wilson, X. Xu, A first-principles predictive model of the pedestal height and width: Development, testing and iter optimization with the eped model, Nuclear Fusion 51 (10) (2011) 103016. `doi:10.1088/0029-5515/51/10/103016`.

[59] B. Khaleghi, A. Khamis, F. O. Karray, S. N. Razavi, Multisensor data fusion: A review of the stateâĂŘofâĂŘtheâĂŘart, Information Fusion 14 (1) (2013) 28–44. `doi:10.1016/j.inffus.2011.08.001`.

[60] A. Kline, et al., Multimodal machine learning in precision health: A scoping review, npj Digital Medicine 5 (2022) 1âĂŞ14. `doi:10.1038/s41746-022-00712-8`.

[61] C. Cheng, L. Messerschmidt, I. Bravo, M. Waldbauer, R. Bhavikatti, C. Schenk, V. Grujic, T. Model, R. Kubinec, J. Barceló, A general primer for data harmonization, Scientific data 11 (1) (2024) 152.

[62] Y. Nan, J. Del Ser, S. Walsh, et al., Data harmonisation for information fusion in digital healthcare: A state-of-the-art systematic review, meta-analysis and future research directions, Information Fusion 82 (2022) 99âĂŞ122. `doi:10.1016/j.inffus.2022.01.001`.

[63] N. E. I. Hamda, A. Hadjali, M. Lagha, Multisensor data fusion in iot environments in dempsterâĂŞshafer theory setting: An improved evidence distance-based approach, Sensors 23 (11) (2023) 5141. `doi:10.3390/s23115141`.

[64] E. Koksalmis, Sensor fusion based on dempsterâĂŞshafer theory of evidence, Intelligent Systems 35 (5) (2020) 843–859. `doi:10.1002/int.22237`.

[65] A. M. Hanea, et al., Mathematically aggregating expertsâĂŹ predictions of possible futures, Risk Analysis 41 (4) (2021) 765–787. `doi:10.1111/risa.13523`.

[66] E. Nazari, [other authors], Decision fusion in healthcare and medicine: A narrative review, BMC Medical Informatics and Decision Making 22 (1) (2022) 1–24. `doi:10.1186/s12911-022-01893-w`.

[67] B. S. Sammuli, J. L. Barr, N. W. Eidietis, E. E. J. Olofsson, S. M. Flanagan, M. Kostuk, D. A. Humphreys, Toksearch: A search engine for fusion experimental data, Fusion Engineering and Design 129 (4 2018). `doi:10.1016/j.fusengdes.2018.02.003`.

[68] Toksearch documentation.
URL `https://ga-fdp.github.io/toksearch/latest/`

[69] B. Sammuli, Enhancing fusion research with toksearch: Updates and integration into the fusion data platform, in: 14th Technical Meeting on Control Systems, Data Acquisition, Data Management and Remote Participation in Fusion Research, 2024, presented at the 14th Technical Meeting on Control Systems, Data Acquisition, Data Management and Remote Participation in Fusion Research.
URL `https://conferences.iaea.org/event/377/contributions/31647/attachments/17876/30195/sammuli_iaea_tm_2024_v0.pdf`

[70] S. W. Smith, The Scientist & EngineerâĂŹs Guide to Digital Signal Processing, California Technical Publishing, 1997.
URL `https://www.dspguide.com/pdfbook.htm`

[71] H. Wickham, Tidy data, Journal of Statistical Software 59 (10) (2014).
URL `https://www.jstatsoft.org/article/view/v059i10`

[72] S. Van Buuren, S. Van Buuren, Flexible imputation of missing data, Vol. 10, CRC press Boca Raton, FL, 2012.

[73] pandas development team, `Series.interpolate` documentation, `https://pandas.pydata.org/docs/reference/api/pandas.Series.interpolate.html` (2024).

[74] SciPy Developers, Scipy interpolation guide âĂŤ linear interpolation, accessed: 2025-08-11 (2025).
URL `https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html#linear-interpolation`

[75] J. Van Der Donckt, J. Van Der Donckt, M. Rademaker, S. Van Hoecke, Minmaxlttb: Leveraging minmax-preselection to scale lttb, in: 2023 IEEE Visualization and Visual Analytics (VIS), IEEE, 2023, pp. 21–25.

[76] P. Virtanen, R. Gommers, T. E. Oliphant, et al., SciPyÂă1.0: Fundamental algorithms for scientific computing in Python, Nature Methods 17 (3) (2020) 261–272, function cited: `scipy.signal.savgol_filter` (Savitzky–Golay smoother), tested with SciPy 1.12.0. `doi:10.1038/s41592-019-0686-2`.

[77] P. Virtanen, R. Gommers, T. E. Oliphant, et al., SciPyÂă1.0: Fundamental algorithms for scientific computing in Python, Nature Methods 17 (3) (2020) 261–272, function cited: `scipy.signal.resample_poly` with Kaiser window (polyphase FIR), tested with SciPy 1.12.0. `doi:10.1038/s41592-019-0686-2`.

[78] R. E. Crochiere, L. R. Rabiner, Multirate Digital Signal Processing, Prentice Hall, Englewood Cliffs, NJ, 1983.

[79] A. J. Koomthanam, S. Serebryakov, A. Tripathy, G. Nayak, M. Foltin, S. Bhattacharya, Integrated framework for trustworthy artificial intelligence pipelines, IEEE Internet Computing 28 (3) (2024) 55–61. `doi:10.1109/MIC.2024.3377170`.

[80] Hewlett Packard Enterprise, CMF: CMF library helps to collect and store information associated with ML pipelines, https://github.com/HewlettPackard/cmf, accessed: [Insert date of access] (2021).

[81] D. Humphreys, A. Kupresanin, M. D. Boyer, J. Canik, C. Chang, E. C. Cyr, R. Granetz, J. Hittinger, E. Kolemen, E. Lawrence, et al., Advancing fusion with machine learning research needs workshop report, Journal of Fusion Energy 39 (2020) 123–155.

[82] F. H. O'Shea, S. Joung, D. R. Smith, R. Coffee, Automatic identification of edge localized modes in the DIII-D tokamak, APL Machine Learning 1 (2) (2023).

[83] C. Truong, L. Oudre, N. Vayatis, Selective review of offline change point detection methods, Signal Processing 167 (2020) 107299.

[84] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, D. B. Rubin, Bayesian Data Analysis, 3rd Edition, CRC Press, 2013.

[85] S. J. Qin, Statistical process monitoring: basics and beyond, Journal of Chemometrics: A Journal of the Chemometrics Society 17 (8-9) (2003) 480–502.

[86] A. Bustos, E. Ascasíbar, A. Cappa, R. Mayo-García, Automatic identification of MHD modes in magnetic fluctuation spectrograms using deep learning techniques, Plasma Physics and Controlled Fusion 63 (9) (2021) 095001.

[87] D. Meng, et al., A survey of multi-view representation learning, IEEE Transactions on Knowledge and Data Engineering 33 (10) (2020) 1990–2010.

[88] N. M. Ferraro, S. Jardin, A. Kleiner, C. Liu, P. SINHA, C. CLAUSER, B. Lyons, C. ZHAO, A. Wingen, A. M. WRIGHT, et al., The m3d-c1 code as a tool for design validation and whole-device modeling, Tech. rep., Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States) (2023).

[89] L. Shi, E. Valeo, B. Tobias, G. Kramer, L. Hausammann, W. Tang, M. Chen, Synthetic diagnostics platform for fusion plasmas, Review of Scientific Instruments 87 (11) (2016).

[90] G. Staebler, J. Kinsey, R. Waltz, Gyro-landau fluid equations for trapped and passing particles, Physics of Plasmas 12 (10) (2005).

[91] C. Bourdelle, J. Citrin, B. Baiocchi, A. Casati, P. Cottier, X. Garbet, F. Imbeaux, J. Contributors, Core turbulent transport in tokamak plasmas: bridging theory and experiment with qualikiz, Plasma Physics and Controlled Fusion 58 (1) (2015) 014036.

[92] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Computing Surveys 41 (3) (2009) 1–58.

[93] E. S. Page, Continuous inspection schemes, Biometrika 41 (1/2) (1954) 100–115.

[94] D. C. Montgomery, Statistical Quality Control: A Modern Introduction, 6th Edition, John Wiley & Sons, 2009.

[95] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning: With Applications in R, Springer, 2013.

[96] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[97] J. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning, Springer, 2001.

[98] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3) (1995) 273–297.

[99] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[100] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, Vol. 30, 2017.

[101] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nature Reviews Physics 3 (6) (2021) 422–440.

[102] P. Snyder, H. Wilson, J. Ferron, L. Lao, A. Leonard, T. Osborne, A. Turnbull, D. Mossessian, M. Murakami, X. Xu, Edge localized modes and the pedestal: A model based on coupled peeling–ballooning modes, Physics of Plasmas 9 (5) (2002) 2037–2043.

[103] F. Porcelli, D. Boucher, M. Rosenbluth, Model for the sawtooth period and amplitude, Plasma Physics and Controlled Fusion 38 (12) (1996) 2163.

[104] G. Conway, J. Schirmer, S. Klenge, E. Holzhauer, W. Suttrop, Plasma rotation profile measurements using doppler reflectometry, Plasma Physics and Controlled Fusion 46 (6) (2004) 951.

[105] C. De Boor, A Practical Guide to Splines, Vol. 27, Springer-Verlag, 2001. doi:10.1007/978-1-4612-6333-3.

[106] P. Holoborodko, Smooth noise robust differentiators, Consulted on 7 (2008) (2008) 2015.

[107] R. Colchin, D. Hillis, R. Maingi, C. Klepper, N. Brooks, The filterscope, Review of scientific instruments 74 (3) (2003) 2068–2070.

[108] F. H. OâĂŹShea, S. Joung, D. R. Smith, R. Coffee, Automatic identification of edge localized modes in the diii-d tokamak, APL Machine Learning 1 (2) (2023) 026102. doi:10.1063/5.0134001.

[109] J. Song, S. Joung, Y.-C. Ghim, S.-H. Hahn, J. Jang, J. Lee, Development of machine learning model for automatic elm-burst detection without hyperparameter adjustment in kstar tokamak, Nuclear Engineering and Technology 55 (1) (2023) 100–108. doi:10.1016/j.net.2022.08.026.

[110] Z. A. Xing, D. Eldon, A. O. Nelson, M. A. Roelofs, W. J. Eggert, O. Izacard, A. S. Glasser, N. C. Logan, O. Meneghini, S. P. Smith, R. Nazikian, E. Kolemen, Cake: Consistent automatic kinetic equilibrium reconstruction, Fusion Engineering and Design 161 (2020) 112163. doi:10.1016/j.fusengdes.2020.112163.
URL https://doi.org/10.1016/j.fusengdes.2020.112163

[111] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016) 785–794doi:10.1145/2939672.2939785.

[112] K. H. Burrell, M. E. Austin, D. P. Brennan, J. C. DeBoo, E. J. Doyle, C. M. Greenfield, R. J. Groebner, J. Kim, R. A. Moyer, T. H. Osborne, W. A. Peebles, C. L. Rettig, T. L. Rhodes, H. Reimerdes, D. M. Thomas, M. R. Wade, J. G. Watkins, W. P. West, Quiescent H-mode plasmas in the DIII-D tokamak, Plasma Physics and Controlled Fusion 44 (12A) (2002) A253–A263. doi:10.1088/0741-3335/44/12A/316.

[113] K. H. Burrell, E. J. Doyle, P. Gohil, C. M. Greenfield, R. J. Groebner, C. C. Petty, T. H. Osborne, M. R. Wade, W. P. West, L. Zeng, Quiescent H-mode plasmas with strong edge rotation in the cocurrent direction, Physical Review Letters 102 (2009) 155003. doi:10.1103/PhysRevLett.102.155003.

[114] X. Chen, K. H. Burrell, C. Chrystal, P. B. Snyder, W. M. Solomon, R. J. Buttery, T. H. Osborne, J. M. Hanson, E. Wang, G. R. McKee, Z. Yan, M. E. Austin, D. Eldon, J. Rhodes, R. M. Groebner, R. Nazikian, S. P. Smith, J. Ferron, A. M. Garofalo, Recent progress of the reactor-relevant intrinsically ELM-stable quiescent H-mode on the DIII-D tokamak, in: Proc. 29th IAEA Fusion Energy Conference (IAEA-FEC 2023), IAEA-CN-316-1992, 2023.
URL https://conferences.iaea.org/event/316/contributions/27812/

[115] K. H. Burrell, P. B. Snyder, W. M. Solomon, C. Chrystal, M. E. Austin, R. J. Buttery, X. Chen, E. J. Doyle, T. H. Osborne, C. C. Petty, D. M. Thomas, L. Schmitz, R. E. Waltz, Discovery of stationary operation of quiescent H-mode plasmas with net-zero neutral beam injection torque and high energy confinement on DIII-D, Physics of Plasmas 23 (5) (2016) 056103. doi:10.1063/1.4944031.

[116] X. Chen, K. H. Burrell, C. Chrystal, P. B. Snyder, W. M. Solomon, R. J. Buttery, T. H. Osborne, J. M. Hanson, E. Wang, G. R. McKee, Z. Yan, M. E. Austin, D. Eldon, J. Rhodes, R. M. Groebner, Stationary QH-mode plasmas with high and wide pedestal at low rotation on DIII-D, Nuclear Fusion 57 (2) (2017) 022007. `doi:10.1088/0029-5515/57/2/022007`.

[117] D. Whyte, A. Hubbard, J. Hughes, B. Lipschultz, J. Rice, E. Marmar, M. Greenwald, I. Cziegler, A. Dominguez, T. Golfinopoulos, et al., I-mode: an h-mode energy confinement regime with l-mode particle transport in alcator c-mod, Nuclear Fusion 50 (10) (2010) 105005.

[118] A. Leonard, Plasma detachment in divertor tokamaks, Plasma Physics and Controlled Fusion 60 (4) (2018) 044001.

[119] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, É. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[120] J. Bai, F. Lu, K. Zhang, Onnx: Open neural network exchange, in: Proceedings of the 1st Workshop on Machine Learning Systems, 2019, pp. 1–1.

[121] J. Stoer, R. Bulirsch, Introduction to Numerical Analysis, 3rd Edition, Springer, 2002. `doi:10.1007/978-0-387-21738-3`.

[122] C. de Boor, A Practical Guide to Splines, Springer, 2001. `doi:10.1007/978-1-4612-6333-3`.

[123] SciPy Developers, Scipy documentation âĂŤ cubic ('cubic') interpolation, `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html`, accessed: 2025-08-11 (2025).

[124] F. N. Fritsch, R. E. Carlson, Monotone piecewise cubic interpolation, SIAM Journal on Numerical Analysis 17 (2) (1980) 238–246. `doi:10.1137/0717021`.

[125] SciPy Developers, Scipy documentation âĂŤ `PchipInterpolator`, `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.PchipInterpolator.html`, accessed: 2025-08-11 (2025).

[126] H. Akima, A new method of interpolation and smooth curve fitting based on local procedures, Journal of the ACM 17 (4) (1970) 589–602. `doi:10.1145/321607.321609`.

[127] SciPy Developers, Scipy documentation âĂŤ `Akima1DInterpolator`, `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.Akima1DInterpolator.html`, accessed: 2025-08-11 (2025).

[128] SciPy Developers, Scipy documentation âĂŤ `uniform_filter1d`, accessed: 2025-08-11 (2025). URL `https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.uniform_filter1d.html`

[129] SciPy Developers, Scipy documentation âĂŤ `maximum_filter1d`, accessed: 2025-08-11 (2025). URL `https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.maximum_filter1d.html`

[130] SciPy Developers, Scipy documentation âĂŤ `minimum_filter1d`, accessed: 2025-08-11 (2025). URL `https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.minimum_filter1d.html`

[131] G. J. McLachlan, D. Peel, Finite Mixture Models, Wiley, 2000. URL `https://www.wiley.com/en-us/Finite+Mixture+Models-p-9780471006268`

[132] K. P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012. URL `https://mitpress.mit.edu/9780262018029/machine-learning/`

[133] scikit-learn developers, scikit-learn user guide âĂŤ gaussian mixture and bic/aic model selection, accessed: 2025-08-11 (2025).
URL https://scikit-learn.org/stable/modules/mixture.html#model-selection

[134] T. Paananen, A. Vehtari, Implicitly adaptive importance sampling: Importance weighted moment matching, Statistics and Computing (2021).

[135] F. Gustafsson, Determining the initial states in forward-backward filtering, IEEE Transactions on Signal Processing 44 (4) (1996) 988–992.

[136] H. E. Grecco, Pint: Operate and manipulate physical quantities in pythonAvailable at https://github.com/hgrecco/pint (2013).
URL https://pint.readthedocs.io/