IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 045062 https://doi.org/10.1088/2632-2153/ad12e3

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
21 December 2022

REVISED
27 October 2023

ACCEPTED FOR PUBLICATION
6 December 2023

PUBLISHED
29 December 2023

PAPER

FAIR AI models in high energy physics

Javier Duarte[1,*], Haoyang Li[1], Avik Roy[2], Ruike Zhu[2,3], E A Huerta[3,4], Daniel Diaz[1], Philip Harris[5], Raghav Kansal[1], Daniel S Katz[2], Ishaan H Kavoori[1], Volodymyr V Kindratenko[2], Farouk Mokhtar[1,6], Mark S Neubauer[2], Sang Eon Park[5], Melissa Quinnan[1], Roger Rusack[7] and Zhizhen Zhao[2]

[1] University of California San Diego, La Jolla, CA 92093, United States of America
[2] University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States of America
[3] Argonne National Laboratory, Lemont, IL 60439, United States of America
[4] The University of Chicago, Chicago, IL 60637, United States of America
[5] Massachusetts Institute of Technology, Cambridge, MA 02139, United States of America
[6] Halıcıoğlu Data Science Institute, La Jolla, CA 92093, United States of America
[7] The University of Minnesota, Minneapolis, MN 55405, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: jduarte@ucsd.edu

Keywords: FAIR, AI, high energy physics, Higgs boson, ML

Abstract

The findable, accessible, interoperable, and reusable (FAIR) data principles provide a framework for examining, evaluating, and improving how data is shared to facilitate scientific discovery. Generalizing these principles to research software and other digital products is an active area of research. Machine learning models—algorithms that have been trained on data without being explicitly programmed—and more generally, artificial intelligence (AI) models, are an important target for this because of the ever-increasing pace with which AI is transforming scientific domains, such as experimental high energy physics (HEP). In this paper, we propose a practical definition of FAIR principles for AI models in HEP and describe a template for the application of these principles. We demonstrate the template's use with an example AI model applied to HEP, in which a graph neural network is used to identify Higgs bosons decaying to two bottom quarks. We report on the robustness of this FAIR AI model, its portability across hardware architectures and software frameworks, and its interpretability.

1. Introduction

Breakthroughs in machine learning (ML) and artificial intelligence (AI) have had a major impact on a range of scientific disciplines, including high energy physics (HEP), which is the study of the fundamental constituents of matter and their interactions. In HEP, multiple experimental collaborations have used ML techniques extensively to address a broad range of problems. For example, they were integral to the 2012 discovery of the Higgs boson [1, 2] and subsequent observation of its decay to bottom quarks [3, 4] at the CERN Large Hadron Collider (LHC), where they were used to identify in proton–proton collisions the nature and origin of 'jets' of particles produced in the collisions. In another significant application, ML was used to identify in real time about 1000 events of interest from the 40 million background events produced each second at the LHC [5, 6]. To maximize the scientific impact and utility of AI models in HEP, we propose a set of findable, accessible, interoperable, and reusable (FAIR) principles for them.

Our approach is inspired by community-wide initiatives that have produced guiding principles to maximize the reuse and scientific reach of digital assets. Specifically, the FAIR principles were originally introduced [7] as guidelines for the management and stewardship of scientific datasets to optimize their reuse. Recently, the FAIR for Research Software (FAIR4RS) working group has developed an interpretation of the FAIR principles specifically for research software [8–11], and FAIR principles have also been applied in the context of benchmarking and tool development [12], and on the creation of computational frameworks for AI models [13].

While these are important steps, these prior interpretations of FAIR principles are not readily applicable to AI models, which are conceptually and structurally different from data and research software. Elucidating the details needed for a robust and general definition of FAIR principles for AI models requires application-specific benchmarks. To address these challenges, we propose an operational definition of FAIR for HEP AI models, focusing on pre-trained models used to make predictions on HEP data. These principles are intended to promote research reuse and reproducibility, which are known challenges in AI-driven scientific application research [14]. In addition, we present a method to automate the production, standardization, and publication of Python-based FAIR AI models in HEP.

To illustrate our proposed FAIR AI model definition in the context of HEP, we use a FAIR dataset to create and publish a FAIR AI model. Specifically, we use a simulated Higgs boson dataset distributed by the CMS Collaboration [15–17]. This FAIR dataset has been used for ML studies [18], college courses [19, 20] and tutorials [21]. We create a FAIR version of an interaction network (IN) AI model for Higgs boson identification [18], and show how adopting our FAIR principles simplifies porting the model across different hardware architectures and software frameworks and facilitates the study of its interpretability.

This paper is organized as follows: section 2 outlines the methods used, where section 2.1 describes related work and a formulation of FAIR principles for AI models; section 2.2 introduces an AI project template; section 2.3 summarizes how the template maps to FAIR principles, and section 2.4 describes an example of the application of FAIR principles, where we take a previously published AI model in HEP [18] and make it FAIR. Next, section 3 discusses the portability and interpretability of this model, as enabled by the FAIR principles. Finally, section 4 summarizes the paper.

2. Methods

2.1. FAIR principles for AI models in HEP

Substantial work has been done to investigate how to apply the FAIR principles to research software [8–11]. The design, optimization, and training of ML models combine disparate digital assets, including research software, data, libraries and tools, workflows, and an expanding ecosystem of hardware architectures. Depending on the use case, AI models can often be optimized to be faster, more parallel, or better utilize the underlying hardware within different software toolkits. To minimize misinterpretation, the reproducibility and reusability of AI models require details of provenance for the entire discovery cycle. In addition, to execute the AI model on a new dataset, including new data that has not been preprocessed, an exact recipe of the data preparation and preprocessing steps is required, such as the units used to express the data features [22].

Operationally, an AI model is usually instantiated in a software framework, such as Scikit-learn [23], TensorFlow [24], PyTorch [25], XGBoost [26], or ONNX [27], that may be serialized in a file on disk. The storage of models within these formats can vary from low-level hardware-optimized intermediate representations (IRs) to high-level IRs, leading to different inference results or performance. In addition, preparation and prepossessing steps, which can have an impact on the model, can be specified either in separate scripts, or as layers integrated into the model. There are efforts to share such code as open-source GitHub repositories, like Papers With Code [28]. However, it has been observed that these repositories are often incomplete, lacking key information, and not maintained, making the results difficult to reproduce [14, 29, 30]. This has led to the establishment of AI reproducibility challenges [30, 31]. In light of these considerations, we propose the following definition for a FAIR AI model, aimed at meeting the high-level goals of F, A, I, and R (the four foundational principles) in the original FAIR data principles [7] for AI models:

> *An AI model consists of the architecture (computational graph) and a given set of parameters, which can be expressed as source code files or executables needed to run inference (i.e. produce outputs) on a data sample. A FAIR AI model is an AI model that satisfies the properties listed in table 1. In brief, (F) the model and its associated metadata are easy to find for both humans and machines, (A) the model and its metadata are retrievable via standardized protocols, (I) the model interoperates with other models, data, and/or software, and (R) the model is both usable and reusable.*

For an ML model to be FAIR, we stress that, first, the dataset used to train the model must be FAIR, and follow domain-relevant community standards, because the dataset is an essential part of the ML model's provenance. In table 1, we present a set of proposed FAIR AI principles, adapted from the FAIR principles created for research software [10] by the Research Data Alliance (RDA) FAIR4RS working group [8–10, 32, 33]. This set of principles has been given to the RDA FAIR for ML interest group [34] that formed in September 2022. We believe that these guidelines are the minimum criteria for a model to be considered as

Table 1. Proposed FAIR principles for fully trained AI models used for AI-inference only, based on adapting the original FAIR principles by initially replacing data by AI models and then making further changes based on the characteristics of AI models versus datasets and the ways they are developed, shared, searched for, and used. These proposed principles could be further extended for retraining use cases by amending our proposed definition for the 'Reusability' principle.

F: the AI model, and its associated metadata, are easy to find for both humans and machines.
F1. The AI model is assigned a globally unique and persistent identifier.
F2. The AI model is described with rich metadata.
F3. Metadata clearly and explicitly include the identifier of the AI model they describe.
F4. Metadata and the AI model are registered or indexed in a searchable resource.
A: the AI model, and its metadata, are retrievable via standardized protocols.
A1. The AI model is retrievable by its identifier using a standardized communications protocol.
A1.1. The protocol is open, free, and universally implementable.
A1.2. The protocol allows for an authentication and authorization procedure, where necessary.
A2. Metadata are accessible, even when the AI model is no longer available.
I: the AI model interoperates with other models, data, and/or software by exchanging data and/or metadata, and/or through interaction via application programming interfaces (APIs), described through standards.
I1. The AI model reads, writes and exchanges data in a way that meets domain-relevant community standards.
I2. The AI model includes qualified references to other objects, including the (FAIR) data used to train the model.
R: the AI model is both usable (for inference) and reusable (can be understood, built upon, or incorporated into other models and/or software).
R1. The AI model is described with a plurality of accurate and relevant attributes.
R1.1. The AI model is given a clear and accessible license.
R1.2. The AI model is associated with detailed provenance, such as information about the input data preparation and training process.
R2. The AI model includes qualified references to other models and/or software, such as dependencies.
R3. The AI model meets domain-relevant community standards.

FAIR. However, additional criteria may be necessary to truly ensure a shareable, reproducible, and extendable ML model.

A critical challenge to ensure reproducibility is that of backend optimizations. The output of the AI algorithm can be affected by changes in the operation order, operation precision, and parallelization strategy. Currently, frameworks such as PyTorch and ONNX have different IRs, which can lead to different outputs depending on how the model is initialized or compiled. These differences can be substantial even when the same hardware is used [35]. Moreover, specific processor types may have limitations in the bit precision of various operations. Differences in precision can lead to substantial deviations, rendering exact reproducibility across processors nearly impossible. As a consequence, for the purposes of this discussion, we refer to reproducibility as the ability to produce results that are statistically consistent with the aggregate data on a large scale, but when comparing a single inference on the same data, can deviate within a specified tolerance.

2.2. Cookiecutter4FAIR: FAIR AI project template

Software templates can be used to encourage good practices; Cookiecutter Data Science [36] is one such template that is specifically oriented at data science projects. It consists of a logical, reasonably standardized, but flexible project structure hosted on GitHub for performing and sharing data science work. We took inspiration from this and created a fork of this template generator, called `cookiecutter4fair` [37], with additional features to promote the adoption of our FAIR principles. Other tools, like Showyourwork [38], specifically address the issue of reproducibility in science.

2.2.1. Usage

The project template is designed to be used with the `cookiecutter` [39] program, a command-line utility that creates projects from project templates using the Jinja2 [40] templating engine, and that can be installed via `pip`. A new FAIR AI project can be made with the command `cookiecutter` https://github.com/FAIR4HEP/cookiecutter4fair. The first argument corresponds to the project template that is hosted on GitHub. After asking the user for the project name, repository name, author name, author ORCID,

```
.
├── LICENSE ..................................... License for reusing code
├── Makefile ..... Makefile with commands like make data or make train
├── CITATION.cff ........................... Standardized citation metadata
├── README.md ..... Top-level README for developers using this project
├── data
│   ├── processed ............ The final, canonical data sets for modeling
│   └── raw ....................... Original, FAIR, and immutable data dump
├── Dockerfile ................. For building a containerized environment
├── docs ................................ Sphinx project for documentation
├── models ... Trained and serialized models, model predictions, or model
│   summaries
├── notebooks ........................................... Jupyter notebooks
├── references ..... Data dictionaries, manuals, and other explanatory
│   materials
├── reports ................. Generated analysis as HTML, PDF, LaTeX, etc
│   └── figures ............. Generated graphics and figures for reporting
├── requirements.txt ........... Dependencies for reproducing the analysis
│   environment
├── setup.py ....... Makes project pip installable so src can be imported
└── src ............................... Source code for use in this project
    ├── __init__.py ............................... Makes src a Python module
    ├── data ........................... Scripts to download or generate data
    │   └── make_dataset.py
    ├── features ...... Scripts to turn raw data into features for modeling
    │   └── build_features.py
    ├── models ............... Scripts to train models and make predictions
    │   ├── predict_model.py
    │   └── train_model.py
    └── visualization ..................... Scripts to create visualizations
        └── visualize.py
            └── tox.ini ................................ Tox file with settings
```

Figure 1. Folder hierarchy of the `cookiecutter4fair` v1.0.0 [37] project template. The main Python source code is contained in `src`. The `docs` folder contains a Sphinx project for generating documentation.

description of the project, chosen license, DOI for the input data, DOI for the code (if available), and whether to include a template Dockerfile, `cookiecutter` will create the template structure as shown in figure 1.

The questions that the repository asks the user upon project creation can be found and modified in the file `cookiecutter.json`. The `Makefile` contains commands that allow the user to do various things with their project, such as downloading the data, setting up the test environment, converting the dataset, and training and evaluating the model. It also contains global variables obtained from `cookiecutter.json`. This procedure makes it explicit that the analysis operations are a directed acyclic graph (DAG).

If the data is hosted on Zenodo [41], the user can download the data from the DOI link by invoking `make sync_data_zenodo`, which uses the `zenodo_get` command line utility [42] to download the data. The `Dockerfile` can be built and run to provide a Python environment for the project to work, which installs the dependencies specified in `requirements.txt`. When the Docker image is built, it can be run interactively with the command `docker run -d -t <image name>`. The pre-project and post-project scripts are automatically run before and after the project directory is generated and provide additional

flexibility. After the project template has been generated, the user can organize their source code and documentation in order to follow the FAIR principles.

2.2.2. Design considerations based on FAIR principles

2.2.2.1. Findable

There are many ways to ensure findability for AI models once they are created and published. Simple ways include uploading it to GitHub, GitLab, or BitBucket. Several efforts aim to create 'model commons,' hubs in which models can be shared. Among these are DLHub [43, 44], OpenML [45], MLCommons [46], AI Model Share [47], and Hugging Face [48]. If a publication or arXiv preprint is associated with the software, the code repository can also be linked to it via Papers With Code [28]. However, this does not really support the findability principle.

To improve findability, Zenodo [41] can be leveraged to generate a DOI for the repository, as well as to store metadata. Recently, Hugging Face also enabled the ability to generate DOIs for both data sets and models [49]. Ideally, we would like a way to search all these repositories at the same time. This would require that they each expose a machine accessible search mechanism, ideally using a common standard, and that there is a way to perform a federated search across the full set of repositories.

2.2.2.2. Accessible

Accessibility is another place where standardization is needed. Specifically, we need a standard, open, free, protocol for retrieving a model from an identifier. Then the various model repositories would need to implement the server side of this protocol, and community members would likely then implement the client side of the protocol in common tools in Python, R, and other programming languages.

2.2.2.3. Interoperable

To ensure interoperability, the metadata describing the AI model must thoroughly document all aspects of its structure, training, and inputs, including any prepossessing needed for the raw data and a provenance of the data. To enable machine interoperability, standardized APIs, such as those associated with DLHub, Hugging Face, or NVIDIA Triton Server, can be used [50].

2.2.2.4. Reusable

To enable reusability, it is important to specify the software, tools, and dependencies needed to seamlessly invoke an AI model to extract knowledge from datasets in a given computing environment. This process should be hardware agnostic. This may be accomplished by using container solutions, such as Docker [51] or Apptainer [52].

Reusability for inference only requires fully trained ML models. In this context, a trained ML model may be reusable as the backbone to develop another model or to fine-tune it to perform a different task, e.g. the WaveNet model [53], originally developed for text-to-speech and music generation has been adapted for classification and regression tasks in astrophysics [54, 55]. Recent approaches based on 'foundation models,' [56] in which large models (sometimes containing up to 10^9 parameters) are pre-trained on unlabeled datasets and subsequently fine-tuned for downstream tasks, illustrate the need for reusability at large scale. These approaches envision the creation of a small collection of general-purpose AI models that may be reused for a large class of tasks.

2.2.2.5. Other considerations

Optimally deploying models on a given hardware processor often involves modifying the internal structure of the model to better utilize the hardware resources. These optimizations correspond to transformations of IRs, specified, e.g. in ONNX or the more flexible multi-level IR (MLIR) [57]. These transformations can change the numerical output values of models, affecting their reproducibility. There has been limited broad scale acceptance of a standard IR for AI models. In place of this, appropriate metadata describing the hardware used and any hardware-specific optimizations is needed to ensure the model can be reliably reproduced.

In some ways, a higher standard than FAIR is full reproducibility. To ensure reproducibility requires clearly communicating the details of the full end-to-end AI cycle encompassing data collection and curation, API selection for model R&D, hyperparameter optimization, design of domain-inspired loss functions, distributed training schemes, optimizers, random/frozen initialization of weights, data split choices for training, validation, testing and quantization, data loaders, hardware used, and hardware-specific optimizations, among other details. The diverse and rather disparate portfolio of available choices, and the different levels of AI and computing skills of end users, may mean that full reproducibility is not possible. In this article, we propose a minimum and achievable standard of FAIR principles in the context of AI models used for inference.

Table 2. Map between existing capabilities of the `coookiecutter4fair` AI project template and our proposed FAIR principles for AI models. The ∗ symbol indicates that the process is not yet fully automated and requires additional manual steps.

Principle	GitHub repository	Zenodo upload	DLHub upload	Docker or Apptainer image	License
Findable	✓				
Accessible		✓	∗		
Interoperable				✓	
Reusable			∗	✓	✓

2.3. Mapping to FAIR principles

Table 2 summarizes how the features of the `coookiecutter4fair` AI project template map to the proposed FAIR principles for AI models. Most aspects are fully automated, such as the creation of a license file and `Dockerfile` for creating an environment. Some aspects are partially automated, such as uploading the model to Zenodo. In particular, the GitHub–Zenodo bridge can be enabled from the Zenodo web interface, which automates the generation of an updated entry for each new release on GitHub. The `coookiecutter4fair` repository template populates a `CITATION.cff` file [58] with citation metadata, which can then be used by Zenodo. Finally, other aspects are not fully automated, but require some additional manual steps, such as uploading the model to DLHub as described above.

2.4. FAIR implementation of $H \to b\bar{b}$ IN

The Higgs boson is a linchpin of the standard model (SM) of particle physics. It is a byproduct of the mechanism that generates masses for all elementary particles. Studying its properties, such as its production and decay rates, is one of the overarching goals of the CERN LHC program, and any deviations measured with respect to the SM may give a hint to elusive new physics. The Higgs boson most commonly decays (about 58% of the time) to a bottom quark–antiquark pair ($b\bar{b}$). Traditionally, this is a difficult decay of the Higgs boson to study because there is a large background consisting of jets produced through the strong interactions. These are known as quantum chromodynamics (QCD) multijet events. ML models, especially graph neural networks (GNNs) [18, 59], have been shown to dramatically improve the rejection of this background, while retaining high $H \to b\bar{b}$ detection efficiency thus enabling the study of this decay mode. In this section, we provide a concrete example of implementing one such model, which is an IN model described in [18], following our recommendations for a FAIR AI model.

The data structure in HEP is defined around the concepts of events. These are discrete moments where all the particles arising from a single proton–proton collision are measured by a detector and recorded. Each event is independent of all the other events. A dataset may consist of several millions of events. To identify events with a $H \to b\bar{b}$ decay and separate them from the much larger QCD background, several salient features are illustrated in figure 2. At the LHC, for each event particle candidates are reconstructed from detector measurements and clustered into cone-shaped jets, attempt capture most of the energy from a single particle produced in the collision, such as Higgs boson. Charged particles produced in the collision are detected and the momenta and direction are measured in a tracking detector. These tracks are collected to form jets. There is a special class of jets from bottom quarks where the particles travel a measurable distance from the collision vertex before decaying to other particles, forming a so-called secondary vertex (SV). It is this class of jets that we are searching for when we search for $H \to b\bar{b}$ decays.

2.4.1. IN model

The IN model was first proposed [60] in order to explore the evolution of physical dynamics and was later adapted for the task of jet classification; in this case differentiating $H \to b\bar{b}$ jets from QCD jets [18]. The dataset for training, validation, and testing is derived from the CMS open simulated dataset with 2016 conditions that is available from the CERN Open Data Portal [15]. It consists of jets, decomposed into constituent charged particle tracks, and SVs, labeled as either $H \to b\bar{b}$ signal or QCD background. More information on the dataset can be found in Chen *et al* [16]. Figure 3 shows the IN model architecture and table 3 provides the values of the model hyperparameters as well as input data dimensions for the baseline model. For a detailed description of the model and chosen hyperparameters, see Moreno *et al* [18].

As discussed in Moreno *et al* [18], graphs are natural data structures to describe jets because they are permutation invariant (i.e. there is no preferred order to the constituents of the jet), they can accommodate variable-sized objects (i.e. jets may be composed of a few or many constituents), and they can describe entities as nodes (i.e. constituents) and their relations as edges. This network was trained on graph data structures based on up to $N_p = 30$ particle tracks, each with $P = 60$ features, and up to $N_v = 5$ SVs, each with

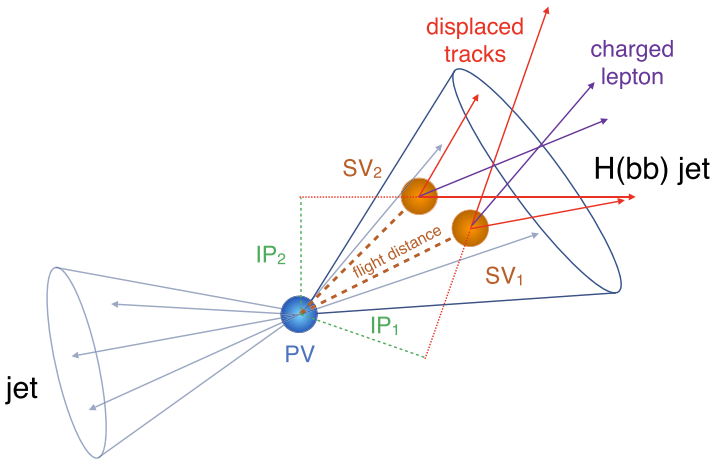

Figure 2. Illustration of a $H \to b\bar{b}$ jet with two secondary vertices (SVs) from the decay of two bottom hadrons resulting in charged-particle tracks (including a low-energy, or soft, lepton) that are displaced with respect to the primary collision vertex (PV), and hence have a large impact parameter (IP) value.

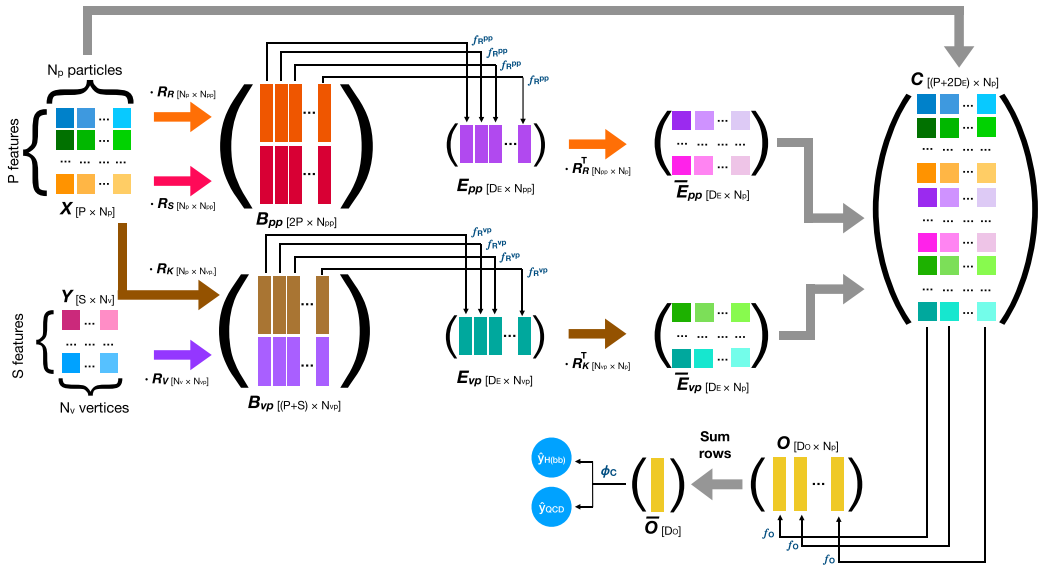

Figure 3. Network architecture and dataflow in the IN model [18]. The choice of model hyperparameters and input data dimensions for the baseline model is given in the accompanying table.

Table 3. The choice of IN model hyperparameters and input data dimensions for the baseline model.

Hyperparameter	Value
$(P, N_\mathrm{p}, S, N_\mathrm{v})$	$(30, 60, 14, 5)$
No. of hidden layers	3
Hidden layer dimension	60
(D_E, D_O)	(20, 24)
Activation	ReLU

$S = 14$ features, associated with the jet. The physical description of each feature is given in appendix C of Moreno *et al* [18].

Two input graphs are used: a fully-connected directed graph with $N_\mathrm{pp} = N_\mathrm{p}(N_\mathrm{p} - 1)$ edges between the particle tracks and a separate graph with $N_\mathrm{vp} = N_\mathrm{v}N_\mathrm{p}$ connections between the particle tracks and the SVs. The node level feature space of the fully connected track graph is transformed to edge level features via two interaction matrices, identified as $R_{R[N_\mathrm{p} \times N_\mathrm{pp}]}$ and $R_{S[N_\mathrm{p} \times N_\mathrm{pp}]}$, where the former accounts for how each node receives information from other nodes and the latter encodes the information about each node sending

information to other nodes. The track–vertex graph is transformed by similarly defined interaction matrices: $R_{K[N_p \times N_{vp}]}$ and $R_{V[N_v \times N_{vp}]}$. The feature spaces of these graphs are transformed via nonlinear functions, respectively called f_R^{pp} and f_R^{vp}, to obtain two D_E dimensional internal state representations of these graphs. These nonlinear functions are approximated by fully connected multilayer perceptrons (MLPs).

These internal state representations, respectively given by $E_{pp[D_E \times N_{pp}]}$ and $E_{vp[D_E \times N_{vp}]}$ matrices, are transferred back to the particle tracks by transforming them with R_R^T and R_K^T matrices. These transformed particle level representations are given by matrices $\bar{E}_{pp[D_E \times N_p]}$ and $\bar{E}_{vp[D_E \times N_p]}$ respectively. Concatenating these particle-level internal state representations with the original track features creates a feature space with a dimension of $(P + 2D_E)$ for each of the N_p tracks. The function f_O, represented by a trainable dense MLP, creates the post-interaction D_O dimensional internal representation that is stored in the matrix $O_{[D_O \times N_p]}$. Finally, these track-level internal representations are summed to obtain a D_O dimensional state vector $\bar{O}$ and linearly combined to produce a two-dimensional output, which is transformed to individual class probabilities via a softmax function.

2.4.2. FAIR implementation

We created a FAIR implementation of the AI model hosted on GitHub and Zenodo [61]. The repository was initialized using the template described in section 2.2.

2.4.2.1. Features

The repository includes a dataset processing script that converts the raw data from the CERN Open Data portal. It also has training and prediction scripts to reproduce the published results. As described above, `Makefile` contains all of these commands, which codifies the analysis as a DAG.

In addition, two `Dockerfiles` that can create reproducible environment for either CPU-based or GPU-based model training and inference are included in the repository. These images are prebuilt and hosted on DockerHub. We also automated documentation generation, training and inference workflows, Docker container building, with continuous integration through GitHub Actions. Finally, a DOI is generated using the Zenodo–GitHub bridge, in which a new DOI is minted for each new release of the software on GitHub.

2.4.2.2. Deployment to DLHub

We have made the trained ML model accessible [62] and reusable for inference by making it publicly available via DLHub [43, 63]. DLHub provides a custom software development kit (SDK) called `dlhub_sdk` that allows users to package and preserve a trained model with necessary dependencies, including packages with specific versions, custom modules, and serialized data and model files. Once a model has been published, its dedicated API can be used to run remote inference tasks using `funcX`, a fire-and-forget remote function execution that elastically deploys workers and containers across nodes in clouds, clusters, and supercomputers [64]. The process of making a model available is simplified with a notebook template made available by DLHub developers. This notebook requires the user to implement the inference code as a function that is executed during model calls, and to declare model-specific dependencies and associate metadata. The notebook template is accompanied with a document template with necessary information about the model. The prescription of using these templates is user friendly: once both templates are filled out and the notebook successfully runs, they can be sent to the DLHub developers who streamline the process of depositing and curating the model. The published model includes a DOI, list of authors, point of contact, relevant information about input and output data type and shape, and instructions to run the ML model with a sample test set. DLHub's SDK also allows users to explore the model's metadata, which encompasses dependencies and libraries used to create and containerize the model, and information about the tasks performed by the model, e.g. classification or regression.

3. Results

3.1. Portability and performance across platforms

In this section, we examine the portability and extensibility of the IN model, a GNN used for the jets classification task. In section 3.1.1, we reproduce the training and evaluation of the IN model with the same hyperparameters and dataset as Moreno *et al* [18]. Section 3.2 retrains the model with different training–validating splits on different servers to test the reproducibility of the results under different conditions. In sections 3.3 and 3.3.1, we explore the model's portability across software frameworks and hardware platforms. We convert the model from PyTorch to TensorRT, using ONNX as the intermediate format, and evaluate the model's inference speed and compatibility of results. We also create an Apptainer container [52] to improve the model's portability across platforms, and evaluate the model's inference performance within the container.

Table 4. The IN model's performance in this work and as reported in the original publication. In this work, we repeat the training ten times varying the random seed used for initialization and data shuffling, and report the mean and standard deviation of the validation accuracy and the AUC. We also report the one-sided (upper tail) p-value for the original model given the distribution of our trials. We find the reported performance of the original model is consistent (p-value $> 5\%$) with our reproduction.

	Validation accuracy	AUC
IN: this work	0.9545 ± 0.0005	0.9898 ± 0.0002
IN: original model [18]	0.9550	0.9900
p-value (consistency)	12.69%	9.27%

3.1.1. Reproducibility

In this subsection we provide details of training the benchmark experiments of the IN model with the same data input and hyperparameters setting as used by Moreno *et al* [18]. The training samples are saved in 57 HDF5 files, each of which contains about 100k jets. We use 52 of them for training and 5 for validation. The testing dataset is saved as a set of NumPy array files (one feature per file), where each file contains 600k jets.

There are several differences in our experiment setting compared to Moreno *et al*. For the training platform, we use the hardware qccelerated learning (HAL) GPU cluster at the National Center for Supercomputing Applications (NCSA) [65] as a remote GPU cluster and train on the NVIDIA V100 GPU, while Moreno *et al* trained their model on one NVIDIA GeForce GTX 1080 GPU. For the data splitting, we take the first five HDF5 files as validation data and the rest as training data. Moreno *et al* split the data into training, validation, and test samples, with 80%, 10%, and 10% of the data respectively. In our training process, each epoch takes about 450 s to finish. The training terminates following the early stopping condition when the validation loss failed to improve for eight epochs. As a first check, table 4 shows a comparison of our training results and the results from Moreno *et al*. We repeat the training ten times varying the random seed used for initialization and data shuffling, and report the mean and standard deviation of the validation accuracy and the area under the curve (AUC). We also report the one-sided (upper tail) p-value for the original model given the distribution of our trials. We find the reported performance of the original model is consistent (p-value $> 5\%$) with our reproduction.

3.2. Robustness

There are a variety of methods to quantify the stability of AI models. Smart data samplers may be developed to expose ML models to novel information at every training epoch. This may be a particularly challenging task if the parameter space is largely unknown, and the optimizer, loss function, and architecture do not encode domain information to properly constrain the ML model during the training stage. Even if the method used to sample the parameter space under consideration during the training stage is suboptimal, the ML model may eventually converge and attain optimal performance, even if the training stage takes longer. The performance of the fully trained model, however, should not be uniquely determined by the method used to split the training, validation, and test sets. In fact, an optimal model should be robust to the selection of training, validation, and test sets, unless the information contained in these datasets is not representative of the phenomena that it aims to describe.

In view of these considerations, we have explored three different data split approaches to handle the HDF5 files that contain the jet data used to produce a new version of the IN model in this article, namely:

(i) Use k-fold cross-validation at the file level. In this approach, the data are split into folds, each containing five files. For training purposes, we select a k-fold as validation data and the rest as training data. We iterate over the entire dataset, and then calculate the average score of all training rounds.
(ii) Randomly select five files as the validation set and the rest as the training set.
(iii) Save the entire dataset as one NumPy array on disk and use the split function in Scikit-learn to randomly split the dataset to create training and validation sets.

We explored these approaches using the IN model in the HAL cluster. Our findings are summarized in table 5. Briefly, the IN model is robust to any of the different methods used to train it, which furnishes evidence for its stability and reliability.

3.3. Portability across hardware platforms

To demonstrate the portability of our IN model implementation across different hardware architectures, we used the HAL and DGX systems at NCSA and the ThetaGPU supercomputer at the Argonne Leadership Computing Facility. The specifications of each of these platforms are summarized in table 6.

Our IN model implementation is produced using a CMS dataset with a suitable format to fit the model's input data size and type. Each file in the dataset includes 10^5 data points. Table 6 provides results for each of

Table 5. Stability of IN model against different training methods in the HAL GPU cluster.

Training epochs	Valid. accuracy	Valid. loss	Train. (valid.) time/epoch (s)	AUC
Method 1: cross validation				
55	0.9541	0.1207	468.8 (23.0)	0.9897
Method 2: random split on file names				
71	0.9555	0.117 57	445.4 (28.6)	0.9901
Method 3: random split on data points				
71	0.955 46	0.117 60	420.4 (20.5)	0.9901

Table 6. Specifications of the DGX, HAL, and ThetaGPU systems.

	HAL System	DGX system	ThetaGPU system
Operation system	ppc64le RHEL 8.4 Linux	x86_64 CentOS 7.4 Linux	x86_64 GNU/Linux
Package versions	Python 3.7.10 PyTorch 1.7.1 Torchvision 0.8.2 Tqdm 4.59.0 Scikit-learn 0.24.2 h5py 3.2.1	Python 3.6.8 PyTorch 1.9.1 Torchvision 0.10.1	Python 3.8.10 PyTorch 1.10.0 Torchvision 0.11.1
GPU	NVIDIA V100	NVIDIA A100	NVIDIA A100

Table 7. IN model portability is showcased using three different data split methods across three different high performance computing platforms.

Platform	Training epochs	Valid. accuracy	Valid. loss	Train. (valid.) time/epoch (s)	AUC
Method 1: cross validation					
HAL	55	0.9541	0.1207	468.8 (23.0)	0.9897
DGX	86	0.9563	0.1160	260.3 (13.0)	0.9905
ThetaGPU	67	0.9555	0.1178	238.5 (10.4)	0.9901
Method 3: random split on data points					
HAL	71	0.95 546	0.11 760	420.4 (20.5)	0.9901
DGX	150	0.9572	0.1131	236.9 (10.9)	0.9908
ThetaGPU	106	0.9563	0.1151	233.97 (10.8)	0.9905

the three training methods described in the previous section, in each of the three high performance computing platforms used for this exercise. Our findings indicate that our IN model implementation is hardware agnostic.

Table 7 summarizes our key findings. We can see that the AUC of the receiver operating characteristic (ROC) curve the validation accuracy are stable at around 99 and 95.5%, respectively. These results are robust to data split methods, and agnostic to the underlying hardware used.

3.3.1. Portability across software frameworks

Here we explore the portability of the AI model across software frameworks that are extensively used for AI research, optimal assembly of software and hardware solutions, and containers.

3.3.1.1. ONNX and TensorRT conversion

Software frameworks such as PyTorch and TensorFlow are extensively used in the AI community. ONNX has emerged as a tool to ease the portability of models developed across software frameworks, and to optimize AI models for accelerated inference using tools such as NVIDIA TensorRT. ONNX has also become a common standard to share and publish ML models. Thus, we have quantified the performance of our IN model in three different implementations: PyTorch, TensorRT and ONNX. The metrics used for this study are inference accuracy, running time, and AUC score.

Table 8. Inference results, produced in the ThetaGPU supercomputer, for different frameworks using partial test data, all test data, and all test data within an Apptainer container.

Framework	Accuracy	Time/batch (s)	AUC
Inference using partial test data			
PyTorch	0.9735	0.0011	0.9915
ONNX	0.9735	0.0008	0.9915
TensorRT	0.9735	0.0099	0.9915
Inference using all test data			
PyTorch	0.975 07	0.001 23	0.991 03
ONNX	0.975 07	0.012 64	0.991 03
TensorRT	0.975 07	0.012 31	0.991 03
Inference on all test data using Apptainer			
PyTorch	0.975 69	0.000 98	0.991 22
ONNX	0.975 07	0.012 24	0.991 03
TensorRT	0.975 06	0.012 32	0.991 03

We carried out these experiments on the ThetaGPU supercomputer using Python 3.6.3, ONNX 1.10.1, PyTorch 1.9.1, and TensorRT 8.2.1.8. For inference, we considered a CMS test set consisting of 180k test events/samples, and then quantified the performance and reliability of our three IN model implementations using the first 10k events in the test data. We set the batch size to 1 for these comparisons. The output of the IN model in these experiments is an array with two values that indicate the probability for the classification of two types of jets. The results of these studies are summarized in table 8.

We also tested these three different implementations using all 1800k events, using a batch size equal to 128. The results of these two experiments are reported in table 8. Inference with the ONNX model is done on the GPU while the data are stored on the host side. Thus, before inference, the data need to be copied from host to device. The time/batch column refers to the time used to run one batch, including the data transfer between the two sides (device, host) and the inference part in the GPU device. When we increase the batch size from 1 to 128, the running time becomes larger because the time taken to transfer the data increases.

For the second case, using a subset of the test set, we can see that when converting from PyTorch to TensorRT, the inference accuracy and AUC score are similar, and the running time of ONNX and TensorRT is shorter due to the accelerating effect of these two formats. When we used the entire test set, the running time of ONNX and TensorRT increase because we use a larger batch size.

3.3.1.2. GPU utilization and throughput

Since NVIDIA TensorRT was developed to optimize AI models for accelerated inference, we have quantified the interplay between batch size, GPU utilization, throughput, and inference accuracy. In this context, throughput corresponds to the number of inferred events per second. In practice, throughput is calculated by computing the total number of inferences divided by total time, or batch size divided by the average running time per batch. Here, running time corresponds to the time taken to complete the analysis of one batch, including data transfer between device–host and the inference part at the GPU. In our experiments, we increased the batch size from 100 to 2400 with a step size equal of 200, while from 2400 to 4200, we used a step size of 400. For each batch size, we run ten times and draw a boxplot of the throughput. Our findings are summarized in figure 4. At a glance, we see that GPU utilization saturates at 100% for a batch size of 1000, while throughput peaks (35k inferred events per second) at a batch size of 1200. These findings exhibit the realm of applicability of TensorRT, i.e. for large scale ML inference workflows.

3.4. Model interpretability

In recent years, advances in explainable AI (XAI) [66] have made it possible to identify novel connections between an AI model's inputs, architecture, optimization, and predictions [67–69]. A substantial subset of XAI methods have been developed to analyze computer vision models where an intuitive reasoning can be extracted from human-annotated datasets to validate XAI techniques. However, in other data structures, like large tabular data or relational data constructs like graphs, the use of XAI methods is still quite new [70, 71]. These XAI techniques have been harnessed across disciplines to quantify the reliability of AI models for science [72–75]. Recently, the scope of XAI has been expanded to include AI applications within HEP [76–79]. In HEP, XAI has been used to understand the output of AI models used in high energy detectors [80], including parton showers at the LHC [81], deep neural network (DNN)-based classification

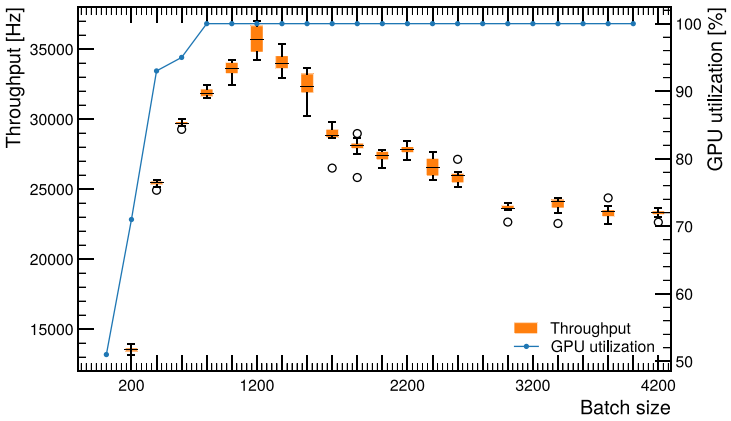

Figure 4. GPU utilization (shown as a blue line) and throughput (shown as box-and-whisker plots) as a function of batch size. GPU utilization saturates at 100% for a batch size of 1000, while throughput peaks at 35k inferred events per second for a batch size of 1200. For the box-and-whisker throughput plots, ten runs are performed with a given batch size. The black line represents the median value of the throughput, the orange box represents the range from the first quartile to the third quartile, and the whiskers extend an additional distance of 1.5× the interquartile range. The white circles represent the outliers.

of jets [82, 83], and particle-based global event description algorithms [84]. Learnable randomness injection [79] provides interpretability by identifying a subset of HEP detector hits in a particle cloud that is the most relevant to the prediction results. This method can also identify whether the existence or specific geometry of a point is important.

3.4.1. Evaluating feature importance

Identifying feature importance has been a significant component of XAI methods and has been thoroughly studied in the context of classification models [85]. In standard feature selection tasks, a reasonable subset of the features that excels in some model performance metric is chosen. Although it is conceptually different from feature ranking in *post-hoc* model interpretation, the latter usually also relies on minimizing a model's performance loss [86]. One of the most useful model analysis tool of a binary classification is the ROC curve, and the corresponding AUC serves as a scalar metric for evaluating model performance. AUC-based feature ranking has been widely used in the AI literature [87–89]. We adapt those same principles for our model interpretation studies. One strategy for evaluating a feature's contribution in making predictions is to investigate the model's performance when that feature is *masked*, e.g. by replacing it with a population-wide average value or a zero value, whichever is contextually relevant to the model's relationship with the training dataset.

In order to identify the features that play the most important role in the IN model's decision-making process, we first train the model with its default settings, which we call the *baseline* model. During the training, for any event where certain input tracks or SVs are absent for a given jet, its corresponding entries are marked with zeros. Hence, we mask one feature at a time for all input tracks or SVs by replacing the corresponding entries by zero values. We obtain predictions from the trained model and evaluate the AUC score. The change observed in the AUC score when masking each of the features is presented in figure 5. It shows that while the model has been trained to take into account the entire feature space, there are 14 track features and 4 SVs features that, if removed one at a time, reduce the model's AUC score by less than 0.05%. Inspired by computer vision studies, we propose that the input features that cause the largest change in the AUC score may be regarded as the features that play the most important role in the model's decision-making process.

The weak dependence of the model on many of its input features indicates that the model can learn the jet classification task from a subset of input features. To further investigate the cumulative impact of removing these *unimportant* features, we mask multiple features at the same time based on a few arbitrary thresholds for the change in AUC score compared to the baseline. The set of masked features includes every track and SV feature that causes a change in AUC score below that threshold when independently masked. To compare how individual predictions vary on average, we compute the model fidelity score [71, 90], defined as

$$F(\mathcal{M}_1, \mathcal{M}_2) = 1 - \frac{1}{N}\sum_{i=0}^{N-1} \left|\hat{y}_i^1 - \hat{y}_i^2\right|. \tag{1}$$

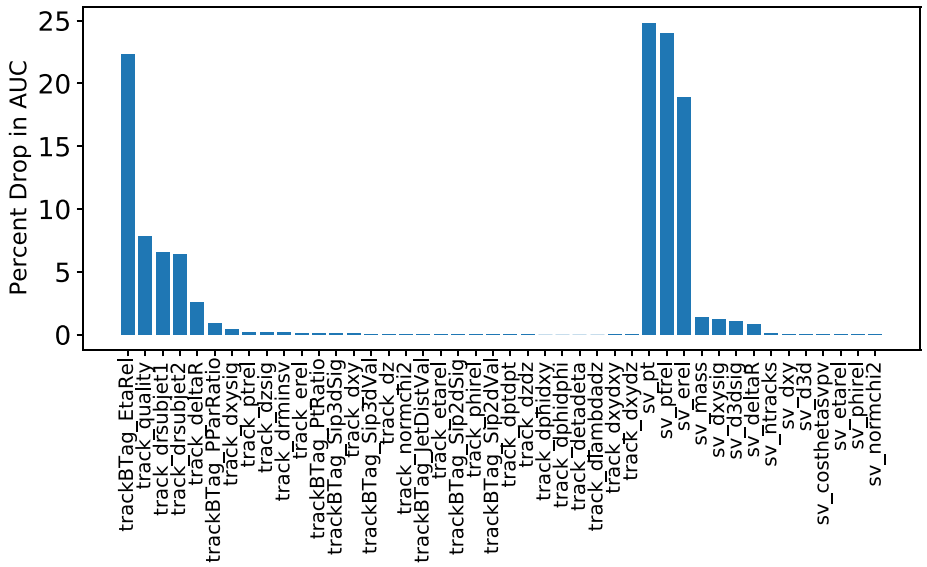

Figure 5. Change in AUC score with respect to a baseline model when each of the tracks and secondary vertex (SV) features are individually masked during inference.

Table 9. The table shows the performance of a baseline model when multiple features are simultaneously masked based on AUC score drop threshold. ΔP (ΔS) represents the number of particle (secondary vertices) features that have been masked. The fidelity score, see equation (1), is measured with respect to the baseline model.

Threshold (%)	ΔP	ΔS	AUC (%)	Fidelity (%)
0	0	0	99.02	100
0.001	8	2	99.02	99.69
0.005	9	2	99.05	99.46
0.01	11	3	98.99	98.85
0.05	14	4	98.81	97.12
1.00	25	8	80.49	70.83

Here, $\mathcal{M}_1$ and $\mathcal{M}_2$ are two different models and the corresponding classifier scores for the i-th data sample are respectively given by $\hat{y}_i^1$ and $\hat{y}_i^2$. The results are summarized in table 9. The model's performance, both in terms of AUC and fidelity scores, remains very close to the baseline even when masking up to 14 particle track and 4 SV features.

While the AUC and fidelity scores allow determining which features play important roles in the IN's decision making process, we can inspect the importance of these features for individual tracks and vertices by the layerwise relevance propagation (LRP) technique [91, 92]. The LRP technique propagates the classification score predicted by the network backwards through the layers of the network and attributes a partial relevance score to each input. The original LRP method has been developed for simple MLP networks. Variants of this method have been explored to propagate relevance across convolutional neural networks [82, 93] and GNNs [84, 94].

Since some of the input features show a high degree of correlation with each other, we use the LRP-γ method described by Montavon *et al* [92], which is designed to skew the LRP score distributions to nodes with positive weights in the network and thus, avoiding propagation of large but mutually canceling relevance scores. In order to apply the LRP method for the IN model, we propagate scores across (i) the aggregation of internal representation of track features obtained from the aggregator network

$$O_{[D_O \times N_p]} \quad \rightarrow \quad \bar{O}_{[D_O]}, \tag{2}$$

and (ii) the interaction matrices that send edge-level representations to the individual particle tracks

$$E_{pp[D_E \times N_{pp}]} \quad \rightarrow \quad \bar{E}_{pp[D_E \times N_p]} \tag{3}$$

$$E_{vp[D_E \times N_{vp}]} \quad \rightarrow \quad \bar{E}_{vp[D_E \times N_p]}. \tag{4}$$

The relevance scores for the output, $O_{[D_O \times N_p]}$, of the f_O function can be obtained as

$$r_{kn} = \bar{r}_k \left(\frac{o_{kn}}{\sum_m o_{km}} \right), \tag{5}$$

where $\bar{r}$ represents the LRP scores for the summed internal representation. On the other hand, the relevance scores $\bar{R}_{kn}$ for the track level internal representations in $\bar{E}_{\mathrm{pp}[D_E \times N_p]}$ can be propagated to edge level representations, $E_{\mathrm{pp}[D_E \times N_{\mathrm{pp}}]}$, using the relation

$$r_{km} = e_{km} \sum_n \left(\frac{\bar{r}_{kn}}{\bar{e}_{kn}} \right) (R_R)_{nm}, \tag{6}$$

where R_R is the receiver matrix for particle–particle interactions. A similar expression allows translating the relevance scores of the track level representation in $\bar{E}_{\mathrm{pv}[D_E \times N_p]}$ to track-vertex edge representations in $E_{\mathrm{vp}[D_E \times N_{\mathrm{vp}}]}$ using the the receiver matrix R_K for vertex–particle interactions.

We show the average scores attributed to the different features for QCD and $H \to b\bar{b}$ jets in figure 6. When compared with the change in AUC score by individual features in figure 5, the track and SV features with largest relevance scores are also the features that individually cause the largest drop in AUC score. We additionally observe that the track features are generally assigned larger relevance scores for QCD jets and SV features play a more important role in identifying the $H \to b\bar{b}$ jets. This behavior is also justified from a physics standpoint, since the presence of high energy SVs is an important signature for jets from b quarks because of its relatively longer lifetime. This is also illustrated in figure 6, where the cumulative relevance score for each track and vertex is shown. The tracks and vertices are ordered according to their relative energy and our results show that the higher energy tracks and vertices are generally attributed with higher relevance scores for both jet classes. However, feature representing relative track energy, `track_erel`, itself does not carry notable relevance weight. On the other hand, the relevance attributed to `sv_pt`, which is strongly correlated with `sv_erel`, is very large.

We also note that while the SV features `sv_ptrel` and `sv_erel` are assigned relatively low relevance scores, masking them independently leads to very large drops in the AUC score. This *apparent* discrepancy can be explained by the very high correlation between these variables, each of which also displays a very large correlation (correlation coefficient of 0.85) with `sv_pt`, as shown in figure 7. Because the LRP-γ method skews the relevance distribution between highly correlated features, it suppresses the LRP scores for those two variables while assigning a large relevance score to the variable `sv_pt`.

We make an additional observation regarding the importance attributed to the feature called `track_quality`. This feature is a qualitative tag denoting the track reconstruction status, and has an almost identical, doubly peaked distribution for both jet categories. In figure 7, the peak at 0 represents absent tracks. With such an underlying distribution, this variable does not contribute to the classifier's ability to distinguish the jet categories. However, the large relevance score associated with it, along with the large drop in AUC score upon masking this feature, indicates that the classifier's class-predictive output for each class somehow receives a large contribution from the numerical embedding used to represent this feature and eventually gets canceled by the softmax operation.

We have found that the two previously mentioned SV features, along with `track_quality`, have no discernible impact on the IN model's ability to tell the jet categories apart by retraining the model without these variables. The model that was trained without these variables, along with the 11 (3) track (SV) features that report a change in AUC of less than 0.01%, converged with an AUC score of 99.00%. In the absence of these redundant features, we observed some differences in the relative distribution of the relevance scores. Thus, we are better able to understand which features play a more important role in the identification of $H \to b\bar{b}$ or QCD jets, respectively. These physics-informed validation of model explanation pinpoints two major drawbacks of the existing XAI methods. First, explanations for models trained with highly correlated input features can be inconsistent across approaches and second, treating categorical and continuous variables on equal footing in XAI methods might lead to misleading attribution of feature importance.

3.4.1.1. Inspecting the activation layers

Here we aim to gain new insights on the IN model's decision-making process at the layer level. As the IN processes the input, it is passed through three different MLPs that approximate arbitrary nonlinear functions

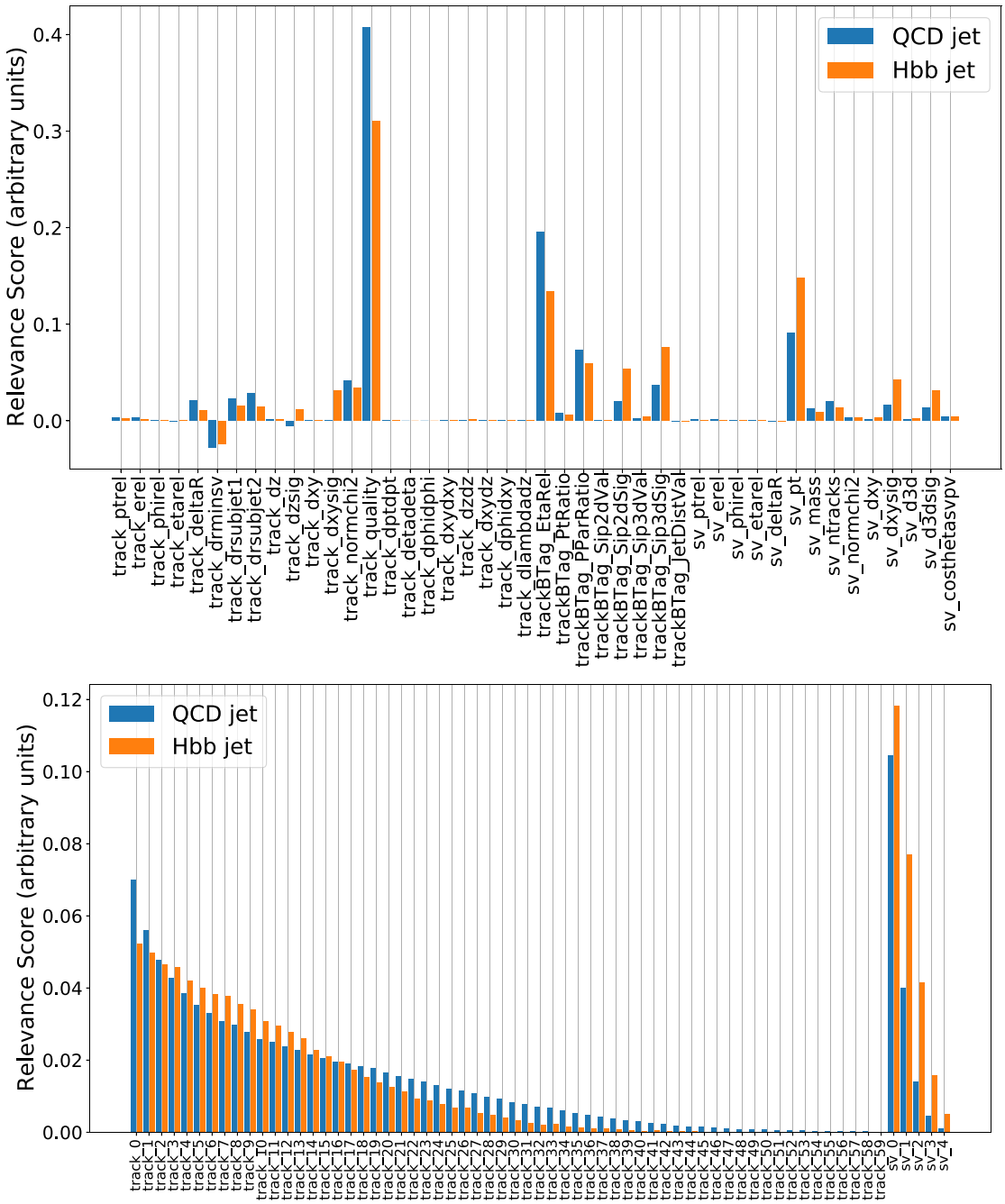

Figure 6. Average relevance scores attributed to input track and secondary vertex features (upper) and individual tracks and secondary vertices (lower). The tracks and secondary vertices (SVs) are ordered according to their relative energy with respect to the jet energy.

identified as f_R^{pp}, f_R^{np}, and f_O. In order to explore the activity of each neuron and compare it with the activity of neurons in the same layer, we define relative neural activity (RNA) [83] as

$$\text{RNA}(j,k;\mathcal{S}) = \frac{\sum_{i=1}^{N} a_{j,k}(s_i)}{\max_j \sum_{i=1}^{N} a_{j,k}(s_i)} \tag{7}$$

where $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ represents a set of samples over which the RNA score is evaluated. The quantity $a_{j,k}(s_i)$ is the activation of jth neuron in the kth layer when the input to the network is s_i. When summed over all the samples in the evaluation set $\mathcal{S}$, this represents the cumulative neural response of a node, which is normalized with respect to the largest cumulative neural response in the same layer to obtain the RNA score.

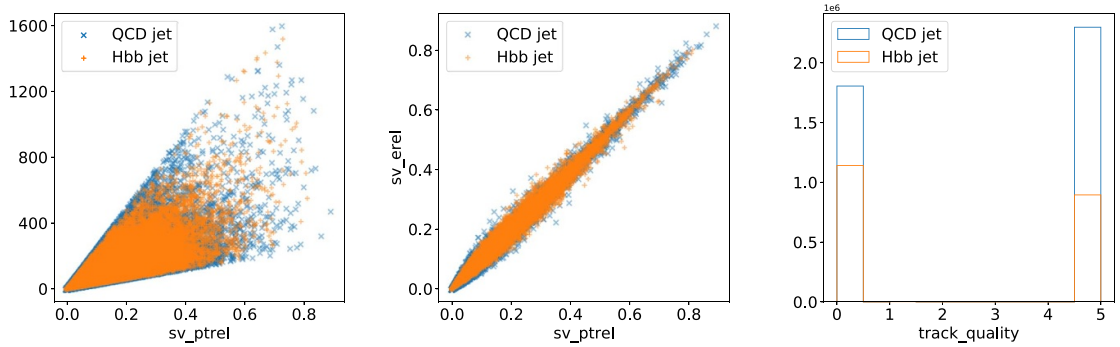

Figure 7. Scatter plots of `sv_ptrel` and `sv_pt` (left) `sv_ptrel` and `sv_erel` (middle), and distribution of the categorical variable `track_quality` (right). `sv_ptrel` and `sv_erel` represent the relative transverse momentum and energy of the secondary vertex with respect those of the jet. `sv_pt` is the transverse momentum of the secondary vertex. `track_quality` is a categorical variable to represent the quality of track reconstruction where the peak at 0 represents absent tracks.

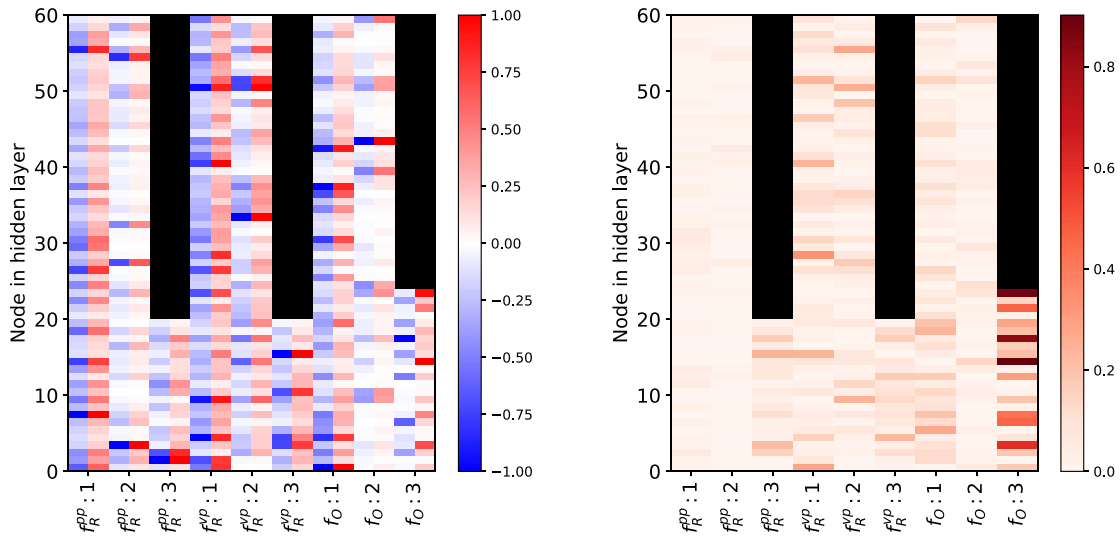

Figure 8. 2D map of relative neural activity (RNA) score for different nodes of the activation layers (left). To simultaneously visualize the scores for QCD and $H \to b\bar{b}$ jets, we project the RNA scores of the former as negative values. 2D map of absolute difference in RNA score for QCD and $H \to b\bar{b}$ jets for different nodes of the activation layers (right). In both figures, the labels associated with the horizontal axis entries represent the nonlinear function and the layer associated with it.

Hence, in each layer, there will be at least one node with an RNA score of 1. Since the neurons are activated with ReLU activation in the IN model, the RNA score will be strictly between 0 and 1.

At a qualitative level, this study aims to identify which neurons are most actively engaged when the IN model produces an output. Since the MLPs in the IN model consist of only fully-connected layers, each layer takes all the activations from the previous layer as inputs. As all nodes within a given layer are subject to the same set of inputs, we can reliably estimate how strongly they perceive and transfer that information to the next layer by looking at their activation values. For the same reason, we normalize the cumulative activation of a node with respect to the largest aggregate in the same layer.

Figure 8 (left) shows the (NAP) diagram for the baseline model, showing the RNA scores for the different activation layers. The scores are separately evaluated for QCD and $H \to b\bar{b}$. To simultaneously visualize these scores, we project the RNA scores of the former as negative values. The NAP diagram clearly shows that the network's activity level is quite sparse. In some layers, more than half of the nodes show RNA scores less than 0.2. This implies that while some nodes are playing very important roles in propagating the necessary information, other nodes do not participate as much. We additionally observe that right until the very last layer of the aggregator network f_O, the same nodes show the largest activity level for both jet categories. This is better illustrated in figure 8 (right), where the absolute difference in RNA scores for the two jet categories are mapped. For most nodes in every layer but the very last one, the difference in RNA scores is very close to zero. However, different nodes are activated in the last layer for the two jet categories, indicating an effective

Table 10. The performance of a baseline and ablated models. ΔP represents the number of particle track features that have been dropped and h is the number of nodes in the hidden layers. The fidelity score is measured with respect to a baseline model. Sparsity is measured by the fraction of activation nodes with an RNA score less than 0.2.

ΔP, ΔS	h, D_E, D_O	Parameters	AUC score (%)	Fidelity (%)	Sparsity
0, 0 (baseline)	60, 20, 24	25 554	99.02	100	0.56
12, 5	32, 16, 16	8498	98.87	96.93	0.52
	32, 8, 8	7178	98.84	96.79	0.48
	16, 8, 8	2842	98.62	96.12	0.40

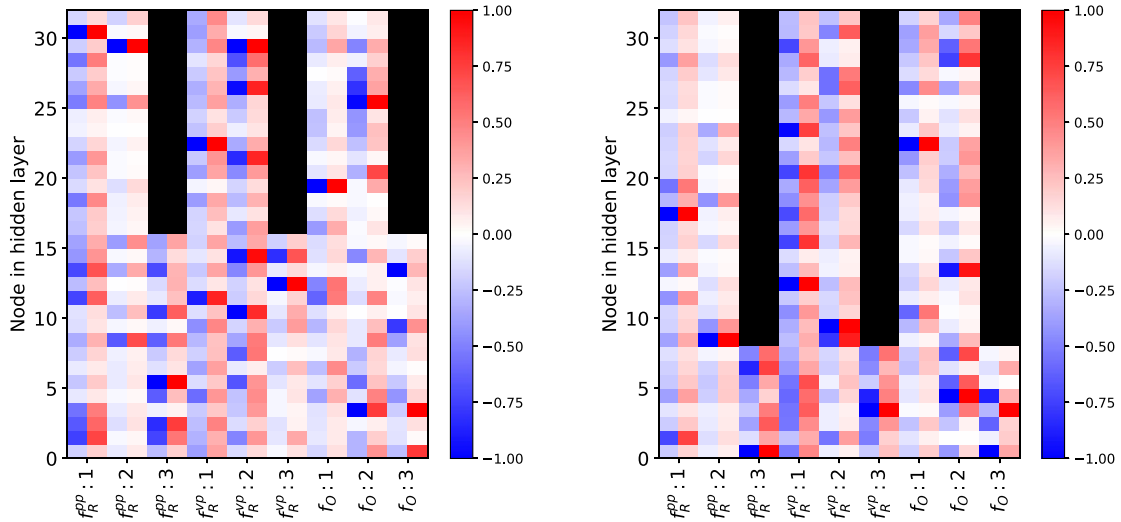

Figure 9. Neural activation pattern diagrams for the IN model where the features `sv_ptrel`, `sv_erel`, `track_quality` along with the additional 11 (3) particle track (secondary vertex) features associated with a change in AUC of 0.01%. In both models, the number of nodes in hidden layers is 32 while $D_E = D_O = 16$ (left) or $D_E = D_O = 8$ (right).

disentanglement of the jet category information in this layer. However, even in this layer, the activity level appears to be sparse—only a few nodes showing large activation for each category.

3.4.2. Model reoptimization

The studies presented in sections 3.4.1 and 3.4.1.1 suggest that the baseline IN model can be made simpler by reducing both the number of input features it relies on and the number of trainable parameters. To explore this observation, we trained alternate variants of the IN models where the features `sv_ptrel`, `sv_etrel`, and `track_quality` were dropped along with additional 11 track and 3 SV features that reduce the AUC less than 0.01%, as shown in figure 5.

The details and performance metrics of these models are given in table 10. It should be noted that the ablated models presented here represent neither an exhaustive list of such choices nor any result of some rigorous optimization. These results demonstrate that a simpler IN model may be developed without compromising the quality of its performance. As can be seen from the results in table 10, both AUC score and fidelity of the alternate models are very close to that of the baseline model, though the number of trainable parameters is significantly lower.

Figure 9 shows the NAP diagrams for the model with 15 (5) dropped track (vertex) features with 32 nodes per hidden layer where the internal representation dimensions D_E and D_O are set to 16 and 8 for the left and right figures, respectively. Sparsity of the latter, as measured by the number of activation nodes with RNA < 0.2, is noticeably lower than the baseline model though the former has increased sparsity. With reduced size for the post interaction internal space representation, the alternate models do not completely disentangle the jet classes at the output stage of f_O.

4. Discussion and conclusion

We have proposed a practical definition of FAIR principles for ML and AI models in experimental HEP. To promote adherence to these principles, we have introduced a FAIR AI project template and demonstrated how to implement this template with a model to identify Higgs bosons decaying to bottom quarks. We studied the robustness of this FAIR AI model and its portability across hardware architectures and software

frameworks, and reported new insights on the interpretability of AI predictions, by studying the interplay between FAIR datasets and AI models.

These studies represent a step toward a FAIR ecosystem of data and AI models to enable and streamline automated AI-driven scientific discovery across disciplines [95]. Future work in this area will need to address many outstanding issues, such as providing documentation in a machine-readable way, as well as the development of standardized APIs for federating searching, accessing, and interoperating AI models hosted on different platforms, such as GitHub, DLHub, AI Model Share, and Hugging Face. We also stress that the FAIR principles outlined in this paper are by no means an exhaustive prescription for shareable, reproducible, and extendable scientific AI research. Nonetheless, we recommend the adoption of this FAIR AI model standard to advance HEP research.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://doi.org/10.7483/OPENDATA.CMS.JGJX.MS7Q [15].

Acknowledgments

This research is supported by DE-SC0021258, DE-SC0021395, DE-SC0021225 and DE-SC0021396 from the Office of Advanced Scientific Computing Research (ASCR) within US Department of Energy (DOE) Office of Science, by the FAIR Data Program of the DOE, Office of Science, ASCR, under Contract Number DE-AC02-06CH11357, and by Laboratory Directed Research and Development funding from Argonne National Laboratory, provided by the Director, Office of Science, of the DOE under Contract No. DE-AC02-06CH11357. It used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, and resources supported by the US National Science Foundation's Major Research Instrumentation program, Grant #1725729, as well as the University of Illinois at Urbana-Champaign. We thank Nikil Ravi, Pranshu Chaturvedi, and Huihuo Zheng for expert support creating and deploying ONNX and TensorRT engines, and Apptainer containers in the ThetaGPU supercomputer.

Author contributions statement

J D conceptualized some of the original Cookiecutter template ideas, supervised, and organized the effort. I H K and HL developed the `cookiecutter4fair` project template. D S K provided expertise on FAIR for other types of objects, such as software, and other disciplines, such as computer science. R Z and A R studied the portability and interpretability of the IN model and VVK advised on computing platforms and tools used in this study. M S N advised on the use of ONNX for portability and interpretability of the IN model. E A H guided activities on the use of DLHub and the deployment and use of ML models on disparate high performance computing platforms. All authors contributed to writing and reviewing the work and the manuscript.

Conflict of interests

The authors declare no competing interests.

ORCID iDs

Javier Duarte https://orcid.org/0000-0002-5076-7096
Haoyang Li https://orcid.org/0000-0003-2599-4948
Avik Roy https://orcid.org/0000-0002-0116-1012
E A Huerta https://orcid.org/0000-0002-9682-3604
Daniel Diaz https://orcid.org/0000-0001-6834-1176
Philip Harris https://orcid.org/0000-0001-8189-3741
Raghav Kansal https://orcid.org/0000-0003-2445-1060
Daniel S Katz https://orcid.org/0000-0001-5934-7525
Volodymyr V Kindratenko https://orcid.org/0000-0002-9336-4756
Farouk Mokhtar https://orcid.org/0000-0003-2533-3402
Mark S Neubauer https://orcid.org/0000-0001-8434-9274
Sang Eon Park https://orcid.org/0000-0003-3225-0007

Melissa Quinnan https://orcid.org/0000-0003-2902-5597
Roger Rusack https://orcid.org/0000-0002-7633-749X

References

[1] CMS Collaboration 2012 Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC *Phys. Lett.* B **716** 30
[2] ATLAS Collaboration 2012 Observation of a new particle in the search for the standard model Higgs boson with the ATLAS detector at the LHC *Phys. Lett.* B **716** 1
[3] CMS Collaboration 2018 Observation of Higgs boson decay to bottom quarks *Phys. Rev. Lett.* **121** 801
[4] ATLAS Collaboration 2018 Observation of $H \to b\bar{b}$ decays and VH production with the ATLAS detector *Phys. Lett.* B **786** 59
[5] Duarte J *et al* 2018 Fast inference of deep neural networks in FPGAs for particle physics *J. Instrum.* **13** 07027
[6] CMS Collaboration 2020 The phase-2 upgrade of the CMS level-1 trigger *CMS Technical Design Report* CERN-LHCC-2020-004. CMS-TDR-021 (available at: https://cds.cern.ch/record/2714892)
[7] Wilkinson M D *et al* 2016 The FAIR guiding principles for scientific data management and stewardship *Sci. Data* **3** 160018
[8] Katz D S *et al* 2021 A fresh look at FAIR for research software (arXiv:2101.10883)
[9] Katz D S, Gruenpeter M and Honeyman T 2021 Taking a fresh look at FAIR for research software *Patterns* **2** 100222
[10] Chue Hong N P *et al* 2022 FAIR principles for research software (FAIR4RS principles) (available at: https://zenodo.org/records/6623556#.YqCJTJNBwlw)
[11] Barker M *et al* 2022 Introducing the FAIR principles for research software *Sci. Data* **9** 622
[12] Verma G *et al* 2021 HPCFAIR: enabling FAIR AI for HPC applications *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)* p 58
[13] Ravi N, Chaturvedi P, Huerta E A, Liu Z, Chard R, Scourtas A, Schmidt K J, Chard K, Blaiszik B and Foster I 2022 FAIR principles for AI models with a practical application for accelerated high energy diffraction microscopy *Sci. Data* **9** 657
[14] Haibe-Kains B *et al* 2020 Transparency and reproducibility in artificial intelligence *Nature* **586** E14
[15] CMS Collaboration and Duarte J 2019 Sample with jet, track and secondary vertex properties for Hbb tagging ML studies (HiggsToBBNTuple_HiggsToBB_QCD_RunII_13TeV_MC) *CERN Open Data Portal* (available at: http://opendata.cern.ch/record/12102)
[16] Chen Y *et al* 2022 A FAIR and AI-ready Higgs boson decay dataset *Sci. Data* **9** 31
[17] McCauley T 2019 Open data at CMS: status and plans *Proc. 7th Annual Conf. on Large Hadron Collider Physics (PoS(LHCP2019))* vol 350 p 260
[18] Moreno E A , Nguyen T Q, Vlimant J-R, Cerri O, Newman H B, Periwal A, Spiropulu M, Duarte J M and Pierini M 2020 Interaction networks for the identification of boosted $H \to b\bar{b}$ decays *Phys. Rev.* D **102** 012010
[19] Benelli G *et al* 2022 Data science and machine learning in education *2022 Snowmass Summer Study* (arXiv:2207.09060)
[20] Duarte J and Wurthwein F 2021 Jupyter notebooks for particle physics and machine learning, UCSD data science capstone particle physics domain (available at: https://zenodo.org/records/4768816)
[21] Duarte J, McCormack P and Rankin D 2022 IAIFI summer school tutorials (available at: https://zenodo.org/records/6954199)
[22] Hanisch R *et al* 2022 Stop squandering data: make units of measurement machine-readable *Nature* **605** 222
[23] Pedregosa F *et al* 2011 Scikit-learn: machine learning in Python *J. Mach. Learn. Res.* **12** 2825 (available at: www.jmlr.org/papers/v12/pedregosa11a.html)
[24] Abadi M *et al* 2015 TensorFlow: large-scale machine learning on heterogeneous systems (available at: www.tensorflow.org/)
[25] Paszke A *et al* 2019 PyTorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32, ed H Wallach (Curran Associates, Inc)
[26] Chen T and Guestrin C 2016 XGBoost *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (ACM)
[27] Bai J *et al* 2017 Open neural network exchange (available at: https://github.com/onnx/onnx)
[28] Meta AI Research 2022 Papers With Code (available at: https://paperswithcode.com)
[29] Wattanakriengkrai S, Chinthanet B, Hata H, Kula R G, Treude C, Guo J and Matsumoto K 2022 GitHub repositories with links to academic papers: public access, traceability and evolution *J. Syst. Softw.* **183** 111117
[30] Pineau J *et al* 2021 Improving reproducibility in machine learning research (a report from the NeurIPS 2019 Reproducibility Program) *J. Mach. Learn. Res.* **22** 1
[31] Sinha K *et al* 2022 ML reproducibility challenge 2022 (available at: https://paperswithcode.com/rc2022)
[32] Katz D S 2021 Defining FAIR for machine learning (ML) (available at: www.rd-alliance.org/defining-fair-machine-learning-ml)
[33] Katz D S 2022 FAIR software and FAIR ML models (available at: https://zenodo.org/records/6647819)
[34] Psomopoulos F and Katz D S 2022 FAIR4ML) IG charter (available at: www.rd-alliance.org/group/fair-machine-learning-fair4ml-ig/case-statement/fair-machine-learning-fair4ml-ig-charter)
[35] PyTorch Team 2022 PyTorch GitHub Issue #87398: model outputs different values after ONNX export (available at: https://github.com/pytorch/pytorch/issues/87398#issuecomment-1338230472)
[36] Driven data 2022 *Cookiecutter Data Science* (available at: https://drivendata.github.io/cookiecutter-data-science/)
[37] FAIR4HEP 2022 Cookiecutter4fair: v1.0.0 (available at: https://zenodo.org/records/7306229)
[38] Luger R *et al* 2021 Mapping stellar surfaces III: an efficient, scalable, and open-source doppler imaging model (arXiv:2110.06271)
[39] Greenfeld A R *et al* 2022 Cookiecutter (available at: https://github.com/cookiecutter/cookiecutter)
[40] Pallets 2022 Jinja (available at: https://github.com/pallets/jinja/)
[41] European Organization For Nuclear Research and OpenAIRE 2013 *Zenodo* (available at: www.zenodo.org/)
[42] Völgyes D 2020 Zenodo_get: a downloader for Zenodo records *Zenodo* (available at: https://zenodo.org/records/10049960)
[43] Li Z *et al* 2021 DLHub: simplifying publication, discovery and use of machine learning models in science *J. Parallel. Distrib. Comput.* **147** 64
[44] Chard K , Lidman M, McCollam B, Bryan J, Ananthakrishnan R, Tuecke S and Foster I 2016 Globus Nexus: a platform-as-a-service provider of research identity, profile and group management *Future Gener. Comput. Syst.* **56** 571
[45] Vanschoren J, van Rijn J N, Bischl B and Torgo L 2013 OpenML: networked science in machine learning *SIGKDD Explorations* **15** 49
[46] MLCommons 2022 MLCommons (available at: https://mlcommons.org)
[47] AI Model Share Project 2022 AI model share platform (available at: www.modelshare.org/)

[48] Wolf T et al 2020 Transformers: state-of-the-art natural language processing *Conf. on Empirical Methods in Natural Language Processing* (Association for Computational Linguistics) p 38
[49] Luccioni S, Bouchot S, Akiki C and Leroy A 2022 Introducing DOI: the digital object identifier to datasets and models (available at: https://huggingface.co/blog/introducing-doi)
[50] NVIDIA 2022 NVIDIA Triton Inference Server (available at: https://developer.nvidia.com/nvidia-triton-inference-server)
[51] Merkel D 2014 Docker: lightweight Linux containers for consistent development and deployment *Linux J.* **2014** 2
[52] Kurtzer G M, Sochat V, Bauer M W and Gursoy A 2017 Singularity: scientific containers for mobility of compute *PLoS One* **12** e0177459
[53] van den Oord A et al 2016 WaveNet: a generative model for raw audio *9th ISCA Speech Synthesis Workshop* p 125
[54] Huerta E A et al 2021 Accelerated, scalable and reproducible AI-driven gravitational wave detection *Nat. Astron.* **5** 1062,
[55] Khan A, Huerta E A and Kumar P 2022 AI and extreme scale computing to learn and infer the physics of higher order gravitational wave modes of quasi-circular, spinning, non-precessing black hole mergers *Phys. Lett.* B **835** 137505
[56] Bommasani R et al 2021 On the opportunities and risks of foundation models (arXiv:2108.07258)
[57] Lattner C et al 2021 MLIR: scaling compiler infrastructure for domain specific computation *2021 IEEE/ACM Int. Symp. on Code Generation and Optimization (CGO)* p 2
[58] Druskat S et al 2021 Citation file format (available at: https://zenodo.org/records/5171937)
[59] Qu H and Gouskos L 2020 ParticleNet: jet tagging via particle clouds *Phys. Rev.* D **101** 056019
[60] Battaglia P W et al 2016 Interaction networks for learning about objects, relations and physics *Advances in Neural Information Processing Systems* vol 29, ed D Lee (Curran Associates, Inc) p 12
[61] Duarte J M, Li B, Roy A and Zhu R 2022 Hbb interaction network: v0.1.1 (available at: https://github.com/FAIR4HEP/hbb_interaction_network)
[62] Moreno E A, Nguyen T Q, Vlimant J-R, Cerri O, Newman H B, Periwal A, Spiropulu M, Duarte J M, Pierini M, Zhu R, Roy A, Huerta E A 2022 FAIR interaction network model for Higgs boson detection *The Data and Learning Hub for Science (DLHub)* (available at: www.dlhub.org/)
[63] Chard R et al 2019 DLHub: model and data serving for science *2019 IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)* (IEEE) p 283
[64] Chard R et al 2020 funcX: a federated function serving fabric for science *Proc. 29th Int. Symp. on High-Performance Parallel and Distributed Computing (HPDC '20)* (Association for Computing Machinery) p 65
[65] Kindratenko V et al 2020 HAL: computer system for scalable deep learning *Practice and Experience in Advanced Research Computing* (ACM) p 41
[66] Miller T 2019 Explanation in artificial intelligence: insights from the social sciences *Artif. Intell.* **267** 1
[67] Gunning D, Stefik M, Choi J, Miller T, Stumpf S and Yang G-Z 2019 XAI—explainable artificial intelligence *Sci. Robot.* **4** eaay7120
[68] Linardatos P, Papastefanopoulos V and Kotsiantis S 2020 Explainable AI: a review of machine learning interpretability methods *Entropy* **23** 18
[69] Vilone G and Longo L 2020 Explainable artificial intelligence: a systematic review (arXiv:2006.00093)
[70] Sahakyan M, Aung Z and Rahwan T 2021 Explainable artificial intelligence for tabular data: a survey *IEEE Access* **9** 135392
[71] Yuan H, Yu H, Gui S and Ji S 2020 Explainability in graph neural networks: a taxonomic survey (arXiv:2012.15445)
[72] Zhang Q-S and Zhu S-C 2018 Visual interpretability for deep learning: a survey *Front. Inf. Technol. Electron. Eng.* **19** 27
[73] Khan A , Huerta E A, Wang S, Gruendl R, Jennings E and Zheng H 2019 Deep learning at scale for the construction of galaxy catalogs in the Dark Energy Survey *Phys. Lett.* B **795** 248
[74] Khan A et al 2018 Deep transfer learning at scale for cosmology (available at: www.youtube.com/watch?v=8-jcf1TZNdA)
[75] Khan A, Huerta E A and Zheng H 2022 Interpretable AI forecasting for numerical relativity waveforms of quasicircular, spinning, nonprecessing binary black hole mergers *Phys. Rev.* D **105** 024024
[76] Neubauer M S and Roy A 2022 Explainable AI for high energy physics *2022 Snowmass Summer Study.* (arXiv:206.06632)
[77] Shanahan P et al 2022 Snowmass 2021 Computational Frontier CompF03 Topical Group Report: machine learning *2022 Snowmass Summer Study* (arXiv:2209.07559)
[78] Miao S, Liu M and Li P et al 2022 Interpretable and generalizable graph learning via stochastic attention mechanism *Proc. 39th Int. Conf. on Machine Learning* vol 162, ed K Chaudhuri p 15524
[79] Miao S, Luo Y, Liu M and Li P 2023 Interpretable geometric deep learning via learnable randomness injection *11th Int. Conf. on Learning Representations*
[80] Turvill D, Barnby L, Yuan B and Zahir A 2020 A survey of interpretability of machine learning in accelerator-based high energy physics *2020 IEEE/ACM Int. Conf. on Big Data Computing, Applications and Technologies (BDCAT)* p 77
[81] Lai Y S, Neill D, Płoskoń M and Ringer F 2022 Explainable machine learning of the underlying physics of high-energy particle collisions *Phys. Lett.* B **829** 137055
[82] Agarwal G, Hay L, Iashvili I, Mannix B, McLean C, Morris M, Rappoccio S and Schubert U 2021 Explainable AI for ML jet taggers using expert variables and layerwise relevance propagation *J. High Energy Phys.* JHEP05(2021)208
[83] Khot A, Neubauer M S and Roy A 2022 A detailed study of interpretability of deep neural network based top taggers (arXiv:2210.04371)
[84] Mokhtar F et al 2021 Explaining machine-learned particle-flow reconstruction *4th Machine Learning and the Physical Sciences Workshop at the 35th Conf. on Neural Information Processing Systems*
[85] Tang J, Alelyani S and Liu H 2014 Feature selection for classification: a review *Data Classification: Algorithms and Applications* ed C C Aggarwal (Chapman and Hall/CRC) p 37
[86] Ribeiro M T, Singh S and Guestrin C 2016 Why should I trust you? Explaining the predictions of any classifier *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* p 1135
[87] Chen X-W and Wasikowski M 2008 FAST: a ROC-based feature selection metric for small samples and imbalanced data classification problems *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* p 124
[88] Wang R and Tang K 2009 Feature selection for maximizing the area under the ROC curve *2009 IEEE Int. Conf. on Data Mining Workshops* (IEEE) p 400
[89] Serrano A J et al 2010 Feature selection using ROC curves on classification problems *The 2010 Int. Joint Conf. on Neural Networks (IJCNN)* (IEEE) p 1
[90] Pope P E et al 2019 Explainability methods for graph convolutional neural networks *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition* p 10772

[91] Binder A *et al* 2016 Layer-wise relevance propagation for deep neural network architectures *Information Science and Applications (ICISA) 2016* (Springer) p 913
[92] Montavon G *et al* 2019 Layer-wise relevance propagation: an overview *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (Springer) p 193
[93] Bach S, Binder A, Montavon G, Klauschen F, Müller K-R and Samek W 2015 On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation *PLoS One* **10** e0130140
[94] Schnake T *et al* 2021 Higher-order explanations of graph neural networks via relevant walks *IEEE Trans. Pattern Anal. Mach. Intell.* **44** 1
[95] Huerta E A, Blaiszik B and Brinson L C 2023 FAIR for AI: an interdisciplinary, international, inclusive, and diverse community building perspective *Sci. Data* **10** 487