HPCFAIR: Enabling FAIR AI for HPC Applications

Gaurav Verma* gaurav.verma@stonybrook.edu
Murali Emani† memani@anl.gov
Chunhua Liao‡ liao6@llnl.gov
Pei-Hung Lin‡ lin32@llnl.gov
Tristan Vanderbruggen‡ vanderbrugge1@llnl.gov
Xipeng Shen§ xshen5@ncsu.edu
Barbara Chapman* barbara.chapman@stonybrook.edu

*Stony Brook University, Stony Brook, NY 11794, USA
†Argonne National Laboratory, Lemont, IL 60439, USA
‡Lawrence Livermore National Laboratory, Livermore, CA 94550, USA
§North Carolina State University, Raleigh, NC 27695, USA

***Abstract*—Artificial Intelligence (AI) is being adopted in different domains at an unprecedented scale. A significant interest in the scientific community also involves leveraging machine learning (ML) to effectively run high performance computing applications at scale. Given multiple efforts in this arena, there are often duplicated efforts when existing rich data sets and ML models could be leveraged instead. The primary challenge is a lack of an ecosystem to reuse and reproduce the models and datasets. In this work, we propose HPCFAIR, a modular, extensible framework to enable AI models to be *Findable*, *Accessible*, *Interoperable* and *Reproducible* (FAIR). It enables users with a structured approach to search, load, save and reuse the models in their codes. We present the design, implementation of our framework and highlight how it can be seamlessly integrated to ML-driven applications for high performance computing applications and scientific machine learning workloads.**

***Index Terms*—HPC, FAIR, AI models, datasets, neural networks**

I. INTRODUCTION

While Machine Learning (ML) and Artificial Intelligence (AI) has disrupted every computing industry, the challenges in quickly accessing, reproducing the results, or reusing the research components have become overwhelming for researchers. The massive data produced by research communities such as experimental datasets, AI models constitute a rich repository of *artifacts*. Implementing sound data management principles is the need of the hour to leverage the rich repositories.

One formal proposal laid out by a consortium Future of Research Communications and e-Scholarship (FORCE11) define the four foundation pillars, namely **FAIR**, that stands for data artifacts to be *Findable*, *Accessible*, *Interoperable* and *Reproducible*. The FAIR principles are collectively proposed and adopted by the universities, industry, funding agencies, and scholarly publishers. It dictates the publication of datasets and AI/ML models and associated research components making them adhere to FAIR principles. The advantages are manifold in that it helps end-users such as domain scientists or application developers to adopt and easily integrate data artifacts into their applications for reuse. This significantly cuts down the application time development and support reproducing their experiments. The emergence of the frameworks' development [1]–[5] to address these challenges demonstrates a conspicuous necessity for applying FAIR [6] data guiding principles driving better data management and stewardship.

Here, we have concisely summed the FAIR data principles (shown in Table I) to recognize their potentials to drive innovations among the scientific computing community. Here onward, for brevity, we will express AI/ML models and related components as "data objects."

Adhering to the previously explained FAIR principles, we propose a framework, HPCFAIR, to assist the high performance computing and science communities comprehend the relationship between models, datasets, and data objects. The overarching goal of this framework is to implement FAIR principles for ML-driven HPC. With the APIs provisioned, users can query for datasets and models with metadata, deploy them in their application and run them without the need to worry about the software support and the need to build a model from scratch. It helps to explore models that are trained and tuned for specific tasks; if not available, the user can save them in a central repository for future reuse.

HPCFAIR empowers researchers to explore the research methodologies, metrics databases, varying datasets, and novel learning techniques. It enables researchers with a framework for pipeline development that

Principle	Description
Findable (F)	F1. Data objects (defined by R1 below) are described with rich metadata F2. Metadata clearly and explicitly include the identifier of the data objects it describes F3. Enable mechanism to find AI models by rich associated metadata F4. Data objects are served in a searchable resource
Accessible (A)	A1. Data objects stored are retrievable by their unique identifier. A2. Communication protocol to retrieve data objects is open, free, and universally implementable. A3. Access to data objects requires authentication and authorization, where necessary. A4. Metadata is accessible even when the data object is no longer available
Interoperable (I)	I1. Data objects use a formal, accessible, and shared language for information description. I2. Data objects are interoperable from one format to another. I3. Data objects include qualified references to other data objects.
Reproducible (R)	R1. Metadata (of the data object) is extensively described with high fidelity. R2. Data objects are served with a public and accessible data usage license. R3. metadata adheres to domain-relevant community requirements.

TABLE I: FAIR Principles

refers to codification and automation of stages to produce an AI model. It may consist of multiple sequential steps performing tasks like data loading, preprocessing, model training, including deployment. Also, it renders them a unified platform to optimize the pipeline using tools or compilers like TVM [7] and TensorRT [8]. Notwithstanding the proposed framework's capability to support the generic ML use cases, we primarily focus on tailoring it to suit the large-scale HPC workload.

The main contributions of this paper are summarized as follows:

- It presents a detailed analysis of existing efforts to help FAIRify AI models.
- The proposed work introduces our framework, HPCFAIR, to enable reusable and reproducible AI models.
- It highlights the capabilities of the proposed framework in pipeline development and design space exploration with a detailed design and API architecture.
- Finally, this work evaluates the functionality with standard AI models and use cases from both HPC and the scientific machine learning communities.

The remainder of the paper is organized as follows: Section II presents the related work in this area, comparing existing state-of-the-art frameworks. Section III describes the internal design and architecture of the proposed framework. Section IV and V discuss the evaluation and present use cases reinforcing FAIR principles to scientific applications. Section VI provides a discussion concluding our work and potential future steps.

II. Related work

The advancement to promote FAIR principles in the AI universe has burgeoned in the recent past. We extensively studied the existing state-of-the-art frameworks and have identified and overcome any gaps in our proposed framework. We first briefly describe them , compare their features in Table II and provide our analysis in Section II-G.

A. MLCube

`MLCube` [9] by MLCommons™ is a containerized interface to machine learning models and datasets. It provides open-source runners capable of running on local machines, cloud servers, or Kubernetes clusters. Being in its infancy, `MLCube` supports tasks like dataset download and training. Users can also create containerized images, for their models, provided they have the dataset, code, and docker files. The generated MLCube can be configured using `mlcube_cookiecutter` APIs. It provides `YAML`-based configuration files that support defining tasks and additional hyperparameter options as runtime flags.

B. DLHub

Data and Learning Hub for Science (`DLHub`) [10] is a cloud-hosted learning system designed to enable the publication of models with descriptive metadata, persistent identifiers, and flexible access control. It packages models into portable containers, enabling low-latency, distributed serving of these models on various heterogeneous platforms. It implements command-line interface (CLI) and software development kit (SDK) support to store, discover, and publish models. `DLHub` provides optimizations as batching and memoization that enhance the inference performance. To assure that all operations are performed by authenticated and authorized users, `DLHub` utilizes Globus authentication mechanism.

C. Collective Knowledge

Collective Knowledge Framework (`CK` or `cKnowledge`) [1] provides unified APIs, command-line interfaces, meta-descriptions, and general automation actions to organize and manage research projects as a database of AI components. The customizable program pipeline with software detection plugins and the automatic installation of missing packages enables AI components like models, datasets, compilers, tools from varying vendors to build and run on unalike platforms. The modular `CK` approach has successfully automated benchmarking, auto-tuning, and co-designing

Category	Feature	Supported Data Object	MLCube	DLHub	CK	MLflow	HuggingFace	TFHub	TorchHub	HPCFAIR
Findable	Search Capability	Dataset	NA	NA	NA	NA	Yes	Yes	NA	Yes
		Model	NA	Yes	NA	Yes	Yes	Yes	Yes	Yes
	Metadata Information	All	NA	Yes	Yes	Only Models	Only Datasets and Models	NA	NA	Yes
Accessible	Accessibility Options	Dataset	API	API	API/CLI	NA	GUI/API	GUI/API	NA	API
		Model	API	GUI/API	API/CLI	API/CLI	GUI/API	GUI/API	GUI/API	API
	APIs Support	All	NA	Python	Python, JSON	Python, R, Java, REST	Python	Python	Python	Python
Interoperable	Format Conversion	Dataset	NA	NA	NA	NA	Yes	NA	NA	Ongoing
		Model	NA	NA	NA	NA	Yes	NA	NA	Yes
	Domain Support	All	Generic	Generic and Scientific	Generic	Generic	NLP	Generic	Generic	Generic, HPC and Scientific
Reproducible	Offering as Individual Component	All	NA	Limited	Limited	Limited	Only Datasets and Models	Only Datasets and Models	Only Models	Yes
	Train and Inference Support	Model	Train	Inference	Both	Both	Both	Both	Both	Both
	Pipeline Development Support	All	NA	NA	Yes	Limited	Yes	Yes	Limited	Yes

*NA: Not Applicable; GUI: Graphical User Interface; API: Application Program Interface; CLI: Command Line Interface; All: model, dataset, custom ML libraries; limited: not applicable to all the data objects; NLP: Natural Language Processing; Generic: industrial ML applications

TABLE II: Comparison of the existing state-of-the-art frameworks

software and hardware for AI. It is also being used to reproduce results from top-tier conferences such as ASPLOS, CGO and Supercomputing.

D. MLflow

`MLflow` [11] is an MLOps platform intended to streamline machine learning development, experimentation, and productization. Classified into several autonomous components such as tracking, projects, models, and registry, it can be collectively employed to log runtime information, package supporting tools, and provide a centralized store and APIs to manage the entire lifecycle of `MLflow` models. The lightweight APIs offered by `MLflow` can be utilized with any existing machine learning application or libraries like TensorFlow, PyTorch, and XGBoost. It presents support for notebooks, standalone applications, and cloud, along with Docker containers.

E. Hugging Face

`Hugging Face` [12] offers Transformer models and datasets as open-source libraries that equip developers with abstraction layers to store, load, and distribute pre-trained NLP models like BERT. The Transformer API is powered by transformer architecture under the hood that scales with training data and model size and facilitates efficient parallel training and captures long-range sequence features. It contains over 2000 pre-trained and fine-tuned models. The hub offers user interface and command line features to load and upload models with associated metadata.

F. Tensorflow and PyTorch Hub

Tensorflow Hub (`TFHub`) [13] is a platform for distributing, discovering, and reusing machine learning components as self-contained Python modules in TensorFlow (TF). A module and pre-trained weights can be reused to retrain across other related and similar tasks assisting transfer learning. `TFHub` provides modules in various domains like text, video, image, etc., in formats like saved models, TF.js, TFLite, and Coral. Once exported to the disk, the modules are self-contained and can be used as an interface to preprocess the user input. The modules are applied to build the part of the TF Graph.

PyTorch Hub [14] is an API and workflow employed to publish pre-trained models to a GitHub repository by adding a Python script that contains functions to load a pre-trained model. These functions, alias "entry-points," define the input and the output of a model. Like other AI model hubs, PyTorch advances research within machine learning community by allowing researchers and developers to leverage plug-and-play models. It provides an interface to load models and pre-trained weights. As of today, according to the PyTorch Hub's official GitHub repository, they don't support hosting pre-trained weights. The users with pre-trained weights need to host them correctly themselves.

G. Comparing State-of-the-art Frameworks

Table II compares different state-of-the-art frameworks proposed recently to implement a resolution to facilitate FAIR AI, as briefly described above. We analyzed them to understand support for multiple features and have attempted to distinguish the uniqueness amongst each.

While frameworks like TFHub and Hugging Face offer search capabilities for datasets and models, none other offer such capabilities for datasets. Except for Hugging Face, other frameworks currently do not support model interoperability. This is a barrier when researchers want to compare the performance of various frameworks on heterogeneous backends. Also, offering each AI component as a data object is critical. While CK does offer support for packaging and reproducing the entire result, individual component reuse in pipeline support is limited. TFHub and Hugging Face offer this support only for the datasets and the models, not custom libraries. Whereas, PyTorchHub supports only models. The innate nature of Hugging Face means that it is extensively built for models used in appliations from natural language processing (NLP) domain. This prevents it from being portable across other domains such as HPC and scientific applications. Among all the existing frameworks, DLHub alone supports scientific workloads explicitly. Similarly, TFHub or PyTorchHub are specific to the TensorFlow-based or Torch-based models. In addition to the discussed frameworks, we also studied various publicly available model zoos from GluonCV [15], Caffe [16], and ONNX [17]. However, most of them lacked the support for datasets and data objects, maintaining only models. Such constraints propose a need for a more generic platform assisting researchers, especially in the HPC and machine learning communities.

Our framework HPCFAIR aims to address the aforementioned limitations: provide support for model interoperatbility, search capabilities for datasets and models, packaged to run seamlessly and integrate into any application while catering for HPC and science domains.

III. Proposed Framework: HPCFAIR

In this section, we will be discussing our solution to help enable FAIRify AI models. Henceforth, we will address our proposed approach as "HPCFAIR". We first detail how our solution addresses the FAIR principles. Next, we present design architecture and the implementation details of HPCFAIR. Designing as a three-tier architecture will enable us to implement each component as an independent module with minimal dependencies and is easily extensible to the other language APIs.

We modularize the storage of data objects' metadata enabling efficient findability. The indexed metadata allows users to search for the required data objects based on tags or keywords. We store the metadata in the JSON-LD format to ensure that it can be accessed via open and standard communication protocols like API calls. In addition to models and datasets, we also provide support to store user implemented custom modules in a format that can easily be discovered, loaded, and employed in a pipeline. At the moment, we provide ONNX support to convert the models. We are working towards implementing checkpoint conversions for the selected models. Also, while saving any new object, we check for duplicate insertions based on primary keys, eliminating any redundant information. We currently support access to public data objects and present steps to access any behind the login data objects. Similarly, while loading any data object, we check for its existence in the cache. In such scenarios, we provide users with either a reuse option or newly force-load the data object. We aim to incorporate authentication checks to ensure that access is granted to only authorized users.

A. Design Overview

As shown in Figure 1, HPCFAIR has a front-end connected to several components implementing tags-based search, user notification, load and store of models and datasets. It also contains a supportive component (HPC Ontology) to provide metadata and an advanced component to automatically synthesize workflows. These two components are still under development and are not within the scope of this paper.

From our detailed analysis of existing state-of-the-art frameworks, we observed that MLCube, in its pre-alpha stage of development, offers a perspective that is easily extensible and obeys the "plug-and-play" philosophy. Considering MLCube as a basis, we have further implemented enhancements to bridge gaps in achieving FAIR AI for HPC.

We developed our framework as a Python library for a lightweight implementation. We will extend it to support other languages such as C++, Java, etc. in the future. Developers are provided with CLI support to discover and load the required components. We store detailed

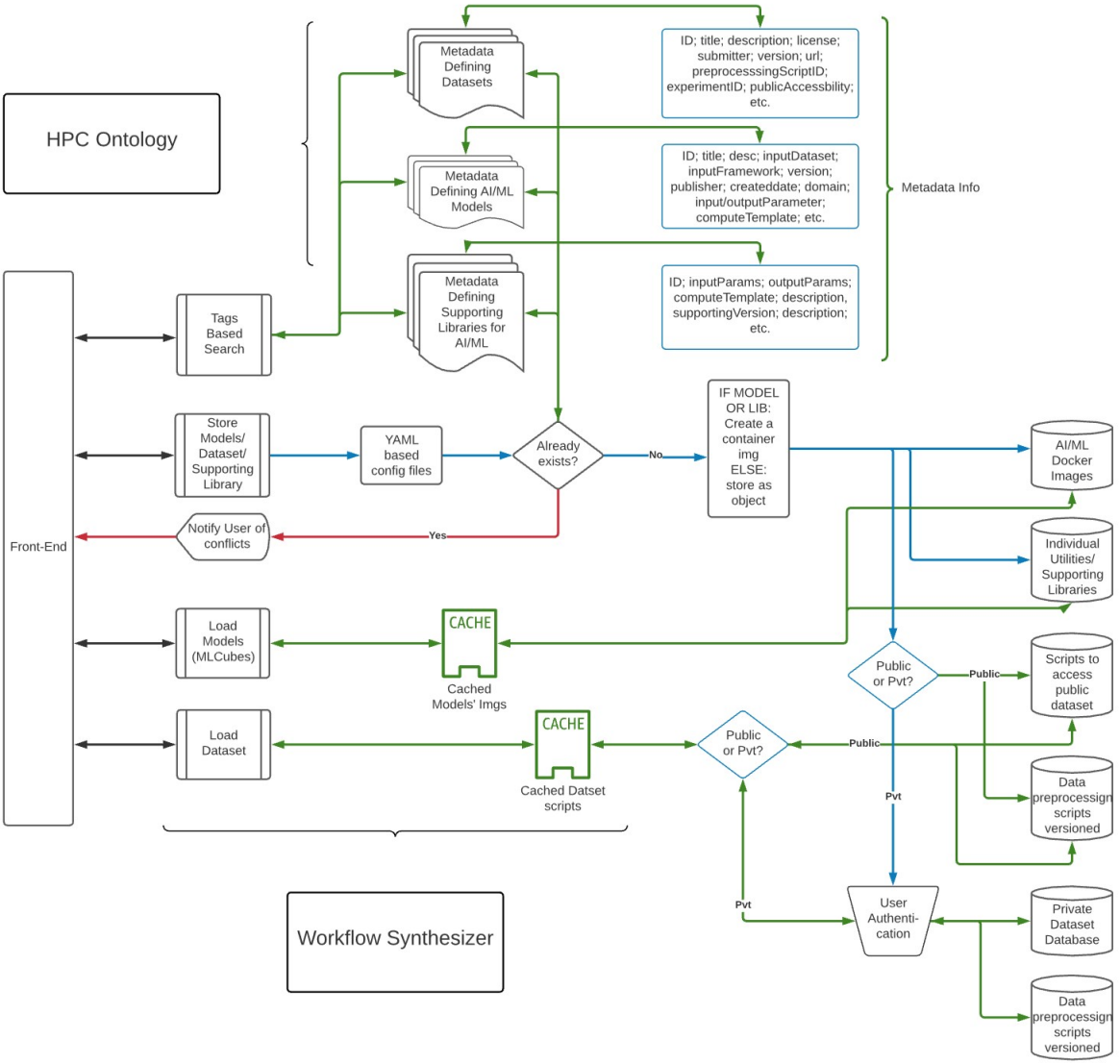

Fig. 1: HPCFAIR: The Proposed Architecture

information about each component in Github repositories in the JSON-LD format. It helps us store a data object and associated files, thus keeping the relationship among them intact. Also, the JSON-LD format allows avenues to be converted into a more efficient search data structure which is our future direction. The requested information is provided to the developers in the dictionary format (key: value) that is easy to comprehend.

1) Serving models, datasets, and data objects: Discussing storage of a model, a dataset, or an individual component, we consider storing the AI components and the experiment's metadata and runtime system configuration information. This will help the users to reproduce the results with added correctness. We employ `MLCube` to store the neural network-based models as containerized images. To store the ML models, we *pickle* them, i.e serialize to the disk as an `MLCube`. It offers `mlcube_ccokiecutter` support to readily generate a containerized image given code, data, and docker file. Instead of tightly coupling the dataset and the model, we create a containerized image of the model alone, facilitating the pipeline development. A containerized image consists of YAML-based configuration files with information about the model and any associated components such as runtime libraries. A uniqueness check is performed to ensure that there is no duplicate submission. Understanding that HPC and scientific workloads might be public or restricted in nature, we have authentication

mechanism in our plan. The future plan is to authenticate users for access to particular data objects, thus serving private data objects along with public data objects. The authentication and authorization should be performed for all the actions on any restricted data object.

2) Tags-based search: As shown in Figure 1, a user can perform "tags-based search" that inherently retrieves information from the enriched metadata. The metadata information is categorized into incorporating components to support efficient and low-latency fetch. Based on the information presented by the search action, a user can further load discovered datasets, models, or any other components. When a load request is placed, an inspection is performed to verify if that component exists in the cache directory. In that case, a duplicate fetch is avoided by reusing the existing component. A user can still "force" load, deleting the current component and downloading it anew. This is more to provide the caching behavior to reduce the time-to-respond. Further, we store metadata broadly classified as models, datasets, and data objects distinctly from each other. The modularity allows an efficient search and facilitates user queries, create, update, read, and delete (CRUD) operations.

3) Pipeline development support: Researchers often want to compare their results against various hand-tuned libraries or custom modules like cost functions. Also, it is not profitable to re-implement the same algorithm from scratch for the same experiment or create a pipeline. In such scenarios, reusability of data objects to achieve an experiment pipeline is highly critical. We enable developers to serve and load data objects on demand. The versioning of data objects allows having multiple versions of the same data object. A ranking system based on usage makes it possible to rank them in a longer run. The detailed metadata of these data objects permits usability specific to applications. In the subsequent sections, we would discuss the encoding of datasets, models, and individual data objects like ML libraries, workflows (experimentations, scripts, etc.) and the granularity of the metadata information stored.

B. Metadata

An AI project consists of many components. We have classified these components into the following categories: dataset, models, individual libraries, supporting scripts like pre and post-processing, associated experiments or workflows, and runtime system configuration. As shown in listing 1, all information is stored into the JSON-LD format using hierarchical key-value pairs. The keys include standard metadata keywords such as `@id` and `@title`. We also provide additional keys with a prefix of `hpc:`.

For example, every dataset is uniquely identified by its value of the `@id` key. The `hpc:tags` are used to manage information like version, license, and tasks. It is a dynamic field permitting users to add user-defined properties. We further store metadata describing the associated files and workflows or experiments to reproduce the research submissions. Additionally, to support citations and locate relevant publications, we save citations as a linked data field.

```
{
    "@id": "http://example.org/DA000001",
    "@title" : "MNIST",
    "@description" : "The MNIST dataset",
    "hpc:submitter" : "admin",
    "hpc:tags" : ["Version" : 0.1, "License" : "MIT",
     "tasks" : "img classification"],
    "hpc:associatedFiles" : "pre_process_mnist",
    "hpc:associatedExpt" : "expt_img_classification",
    "hpc:citation" : "@article{lecun2010mnist}",
}
```

Listing 1: Selected fields from the dataset metadata

Similarly, for the models as presented in Listing 2, we collect the model's submitted format which may be be a `saved_model, onnx,` or `h5` formats, and so on. The `hpc:isTunable` flag informs the user whether the model is tunable during the runtime or not. Furthermore, `hpc:hyperParams` consists of the parameters that can be passed as arguments to the model during runtime. Metadata for machine learning frameworks and model types enables an efficient search and enhances the framework's usefulness. We also present acceleration support and available metrics to the user. It lets the user choose metrics of interest instead of evaluating all.

```
{
    "@id": "http://example.org/MD000001",
    "@title" : "SSD_MobileNet_v2",
    "hpc:modelFormat" : "pb",
    "hpc:isTunable" : "false",
    "hpc:hyperParams" : "",
    "hpc:machineLearningFramework" : "tensorflow",
    "hpc:modelType" : "SSD",
    "hpc:acceleratorSupport" : "true",
    "hpc:metrics" : "throughput,latency",
    "hpc:tags" : {"category":"object detection", "
     dataset":"COCO", "License" : "MIT"}
}
```

Listing 2: Selected fields from the model metadata

Another significant component is experiment or workflow-related metadata as depicted in Listing 3. It describes the files or prerequisite actions required as part of the experiment. The linked workflow files are the command files needed to set up the environment or perform an action. In cases where a whole project package is submitted, a workflow can be leveraged to reproduce the results. To standalone execution, we have fields to record the required software and hardware, in addition to the dataset and model used as part of the experiment. From experience, we have seen that it is critical to have system configuration details to reproduce the expected results. Hence, we provide the functionality to store them.

```
{
    "@id" : "http://example.org/EX000001",
    "@url" : "https://github.com/userX/repoY/XPlacer-
     Adapter.md"
    "hpc:associatedWorkflow" : "workflow_file_ex000001
     ",
    "hpc:reqSoftwares" : ["scikit-learn","pandas","skl
     2onnx","onnxruntime","hyperopt"],
    "hpc:reqHardware" : "nvidia-GPU",
    "hpc:sysConfig" : ["OMP_NUM_THREADS":"4","
     data_format":"NHWC","kmp_affinity":["granularity":
     "fine,compact,1,0"]]
    "hpc:associatedDataset" : ["AWS_data.csv","
     IBM_data.csv","Merged_data.csv"],
    "hpc:associatedModel" : ["modelLearnerGUI.py","
     offline_trainer.py"],
}
```

Listing 3: Selected fields from the experiment metadata

C. Enabling FAIR Principles for AI Models

We have adopted Python API-based methodology to serve, discover, and load the AI models. To incorporate nearly all the available AI frameworks, we serve machine learning models encoded as `Pickle` and deep neural networks based models in various formats including `saved_format (Protobuf, .pb), ONNX, .h5, .pth and .pkl`. Following are a few primary model APIs that are offered to apply FAIR guidelines to the models:

- **`model.list_models()`** returns a list of available models.
- **`model.get_metadata(model_name)`** returns the metadata as a dictionary for a given model or list of models.
- **`model.search(keyword)`** returns the list of models matching the regex.
- **`model.run("train", model_name, dataset_path, **kwargs)`** trains the passed model on the given dataset with the parameters set passed as args. It saves the trained model locally in the provided path.
- **`model.run("inference", model_name, dataset_path, **kwargs)`** runs the inference using the provided model and the dataset with passed hyperparameters.
- **`model.convert(from_format, to_format, src_path, dest_path)`** supports converting model from one format to another; saves it locally.

To serve the model, the user needs to create a containerized image and push it into the Github repository as per the guidelines provided by `MLCube`[1]. This approach to serve models allows the users to package models in any available format. To facilitate interoperability among DNN models, we provide API support to convert models from one format to another. Currently, we support only ONNX-TensorFlow inter-conversion and PyTorch to ONNX conversion. In the near future, we would like to support checkpoint conversion from Tensorflow to PyTorch and vice-versa.

Additionally, we offer YAML-based config files in the containerized images to support training schedules and hyperparameters portability. It is a challenge to scale the training automatically, and we are exploring how an automated hyperparameter optimization (HPO) framework can address this.

[1]https://mlcommons.github.io/mlcube/tutorials/create-mlcube.html

D. Enabling FAIR Principles for Datasets

Alike enabling FAIR principles for the AI models, HPCFAIR API design also provides basic functionality to serve, discover, or load datasets. Below are few APIs that make the pipeline development experience seamless:

- **`dataset.list_datasets()`** returns the list of datasets available publicly or behind the login.
- **`dataset.get_metadata(dataset_name)`** returns the metadata as a dictionary for a given dataset or list of datasets.
- **`dataset.search(keyword)`** returns the list of datasets matching the regex.
- **`dataset.list_supporting_files(dataset_name)`** returns a list of pre-processing or any script associated with the dataset with its description and input arguments.
- **`dataset.load_dataset(dataset_name, src_path, dest_path, **kwargs)`** returns the path of fetched dataset.
- **`dataset.apply(dataset_name, data_path, script_name, **kwargs)`** applies the script from `list_supporting_files` above in-place to the dataset.

While serving the dataset, we maintain the association and versioning to distinguish between various experiments. There are three possibilities while fetching the dataset. The dataset download script is executed to download the dataset in the destination folder passed as an argument if it is public. If any particular dataset requires user login (e.g., the Imagenet dataset), we present the user with a set of steps needed to acquire that. Lastly, if a dataset is private, we are working on provisioning an authentication mechanism where a user can log in and access the private dataset hosted on our framework.

Furthermore, before fetching the dataset afresh, HPCFAIR checks for the cached image of the same dataset to avoid duplicate requests. Finally, the support to apply pre-processing scripts or any other related scripts to the dataset in-place reduces memory consumption. The passed arguments to the pre-processing scripts like `num_threads` to parallelize the execution makes the whole process efficient. As a future direction, we are working on representing the datasets using a standard

in-memory representation framework like Apache Arrow [18]. It combines the benefits of columnar data structures with in-memory computing along with the performance benefits like maximized cache locality, pipelining, and support to the SIMD instructions. Currently, we are replicating these performance gains using APIs.

E. Managing Experiment or Workflows

In addition to the models and datasets, HPCFAIR also supports to enable FAIR principles for experiments or workflows. Often, researchers submit repositories to reproduce results. We decompose that data into data objects if feasible, else contain the information as such and serve the scripts to execute those experiments. We also provide the environment setup files enabling users to recreate the execution environment. In this manner, a developer can use the experiments as a sovereign module in pipeline development.

IV. EVALUATION

To evaluate HPCFAIR for its compliance with FAIR principles, we have conducted the following diversified experiments. We have compared the results for correctness in regards to the original experiments. Additionally, we have demonstrated how the data objects' reusability and interoperability can further tune a model and achieve an efficient software-hardware co-design system.

A. Evaluating Support for DNN models

To assess the HPCFAIR framework upon a fundamental use case, we consider experimenting with the MNIST dataset [19]. The MNIST dataset consists of 60,000 examples of handwritten digits forming a test set. The intention here is to replicate the experimentation by MLCube [2], where they employ a simple neural network to classify the earlier mentioned training set into ten classes. In the original approach, the download and train task is unified. In contrast, as shown in Listing 4, we have decoupled them into two independent tasks to 1) reuse the dataset, line 18 and 2) train the model, line 23. In the background, we use runners provided by MLCube to run cubes on different platforms, including docker and singularity. The parameter.yaml file contains the hyperparameters that can be set during the task execution. If not provided during the runtime, the default hyperparameter values are used. As shown in lines 1-2 in Listing 4, users can load the `hpcfair_dataset` and `hpcfair_model` as modules. They can search for the MNIST dataset and related models, as shown in lines 5 and 9. Further, the dataset and model can be loaded, and the training task is run with the custom hyperparameters, as shown in lines 18 and 7 respectively.

[2]https://github.com/mlcommons/mlcube_examples/tree/master/mnist

```
1  >>> from hpcfair import hpcfair_model as model
2  >>> from hpcfair import hpcfair_dataset as dataset
3  >>>
4  >>>
5  >>> model.search("mnist")
6  ['mnist_model_us1']
7  >>> model_mnist = model.load("mnist_model_us1")
8  >>>
9  >>> dataset.get_metadata("mnist")
10 [hpcfair_dataset.DatasetMetadata(
11         title='mnist',
12         description='The MNIST dataset consists of
       70,000 28x28 black-and-white images in 10 classes
       (one for each digits), with 7,000images per class.
        There are 60,000 training images and 10,000 test
       images.',
13         files=None,
14         supporting_files=pre_processing_mnist_0.py,
15         isTunable='true'
16 )]
17 >>>
18 >>> dataset.load_dataset("mnist")
19 "D:\\hpcfair\\data\\mnist.npz"
20 >>> dataset.apply("mnist","D:\\hpcfair\\data\\","
       pre_processing_mnist_0",num_threads="4")
21 Successfully processed the dataset:  python3
       pre_processing_mnist_0.py --data=mnist.npz
       num_threads=4
22 >>>
23 >>> model.run("train", model_mnist, "D:\\hpcfair\\data
       \\mnist.npz", log_dir="D:\\logs")
24 Model successfully trained. Check logs directory for
       more details.
```

Listing 4: Support for generic DNN-models

B. Evaluating Support for ML libraries

In another evaluation, we applied some machine learning algorithms, such as linear regression and logistic regression, on the GPU runtime dataset [20] as independent data objects. We performed a gradient-descent with batch updates to predict GPU computation time. The dataset includes 14 independent features and 241,600 rows. In this evaluation, we applied data processing files on the dataset stored as dataset metadata. The hyperparameters experimented with are learning rates and convergence threshold.

- Search for the dataset
- Load the dataset
- Apply preprocessing steps as a script
- Search the ML libraries stored as data objects in the pickle format
- Deserialize the ML libraries
- Provide hyperparameter values and train the pipeline; experiment with multiple values
- Use the above model to predict GPU runtime based on varying selected features.

Consequently, the resultant pipeline can be saved as a readily reproducible pipeline, a data object, complying with the FAIR principles.

C. Evaluating Reproducibility of Published Research

With proliferating research involving ML and DL, it is essential to reproduce presented results with the least effort. Effectuating these needs, we sought to produce

the results from the Best Paper Award winner submitted in PACT'17 by Chris Cummins et al. [21]. This work introduces a novel framework DeepTune, proposing heuristics predicting optimal mapping for heterogeneous parallelism and GPU thread coarsening factors using LSTM - a deep neural network.

The artifact submitted tightly depends on the CLgen [22] version and has recommended Ubuntu and Python versions. The containerized packaging of the HPCFAIR facilitates the reproduction of the results with efficiency, managing the installation of required dependencies. Also, results achieved from the DeepTune have been confronted with results from the state-of-the-art frameworks from Grewe et al. [23] and Magni et al. [24].

D. Evaluating Support For Workflows

Ensuing is the evaluation revealing the support for the workflows. Often, there are research submissions in the form of packages where a sequence of steps needs to be performed to set up the prerequisites for reproducibility. One such example is where we evaluated HPCFAIR for its efficacy to reproduce the results from Xplacer by Xu et al. [25].

```
>>> from hpcfair import hpcfair_model as model
>>> from hpcfair import hpcfair_dataset as dataset
>>>
>>> model.search_workflows("xplacer")
['XPlacer']
>>> model.get_metadata("xplacer")
[hpcfair_model.ModelMetadata(
        title='XPlacer: A framework for Guiding
    Optimal Use of GPU Unified Memory',
        description='The goal is to decide which
    memory placement policy is best for a given data
    object...',
        files=None,
        supporting_files=['xplacer_wf1.sh','
    xplacer_wf2.sh','xplacer_wf3.sh'],
        isTunable='true',
        stepsToRun='xplacerRunFile.txt'
)]
```

Listing 5: Support for workflow

As prerequisites, the experiment includes steps like building the adapter, collecting the data-level and kernel-level baseline data, merging two levels of baseline data, labeling the merged data, and executing all variants to decide the best-performing variants. The authors do provide a sequence of steps and scripts to be executed to achieve the above. As a solution, we transform them into workflow scripts and serve the related details as metadata in our database. These data objects are bundled together as part of the containerized image. As shown in the Listing 5, line 4-6, a user can search for an experiment and associated workflow scripts. Additionally, a user can view the brief description explaining the steps to execute when projects are submitted as packages using the corresponding metadata key.

E. Evaluating Support For Design Space Exploration

Lately, edge computing has received considerable attention addressing the demand for faster Deep Learning applications at edge devices. This has led to the development of custom accelerators (TPU, GPU, FPGA), DNN-compilers (TVM, TF-Lite), and frameworks (MxNet, Pytorch, TF). However, recognizing a set of components briefed above best suited for a DL task is strenuous. Hence, a comprehensive framework capable of tackling this issue is of paramount relevance for the researchers. In a similar attempt, we have evaluated HPCFAIR to reproduce the results [26] explaining its capabilities for an efficient Design Space Exploration. A code snippet for the same is presented in the Listing 6.

```
>>> from hpcfair import hpcfair_model as model
>>> from hpcfair import hpcfair_dataset as dataset
>>>
>>> model.search("resnet")
['ResNet50_v2']
>>> model_resnet50 = model.load("ResNet50_v2")
>>> data = dataset.load_dataset("img0.jpg", fromPath="
    D:\\data")
>>>
>>> res_trt = model.run("inference", model_resnet50,
    data, config="trt", log_dir="D:\\logs")
>>> res_tflite = model.run("inference", model_resnet50
    , data, config="tflite", log_dir="D:\\logs")
```

Listing 6: Support for design space exploration

The evaluation includes inference using the Resnet50 v2 model trained on the ImageNet dataset [27]. We make the inference pipeline efficient using optimizations proposed by TFLite [28] and TF-TensorRT integrated solutions [29]. The intention is to select the optimal set of components as a pipeline, essentially improving the inference efficiency based on metrics like power consumption, throughput, and reduction in the model size. HPCFAIR offers models, datasets, and frameworks as data object readily integrable into the pipeline rather than building from scratch. Therefore, instead of repeating from scratch, we adopted the plug-and-play methodology, comparing contrasting design space alternatives fairly for each combination of the framework, accelerator, and optimization engine.

F. Enabling FAIR Principles by HPCFAIR

We achieve FAIR guidelines to the AI applications and data objects as listed below.

- **Findable:** The metadata associated with data is registered and indexed in a searchable resource. The metadata is assigned a globally unique and persistent identifier. This enriched metadata enhances searchability. A user will be able to search for the components corresponding to the application's requirements.
- **Accessible:** The metadata is retrieved using a standardized communication protocol. It enables users

to publish or discover their AI components efficiently. Further, access to the metadata is authorized and authenticated wherever necessary.
- **Interoperable:** To manage interoperability, we have represented the metadata using a formal language, JSON-LD [30]. We support qualified references among the stored metadata and data objects. Additionally, to maintain interoperability at the application level, we serve metadata information concerning individual supporting files associated with the data object. We also support ONNX [31], equipping application users to transform models from one format to another.
- **Reusable:** The scientific community oftentimes interacts among researchers to share and reuse crucial components. We provide metadata with detailed provenance to reuse the components to build an AI pipeline by plugging the data objects. The loosely coupled nature of the stored data enables efficient development.

V. Use Cases Reinforcing Scientific Machine Learning Applications

The purpose of HPCFAIR is not limited to the assessed experiments and features' support. Subsequent use cases show how it can be used to apply FAIR principles to the scientific community applications.

A. Democratizing Datasets and Models in Medical Research

Part of the Cancer Distributed Learning Environment (CANDLE) project, Uno [32] is a cancer deep learning benchmark to predict drug response based on molecular features of tumor cells and drug descriptors. The training task on all data sources is a slow process. But there are hand-tuned configurations that can speed up the process for a single data source. The training and inference can further be optimized using a pre-staged dataset. This requires regeneration of the dataset for varying configurations followed by data processing and many other significant hyperparameters like `batch_size`, `cache`, `use_landmark_genes`, `preprocess_rnaseq`, `no_feature_source`, `shuffle`, etc. The HPCFAIR enables democratization by offering datasets and preprocessing scripts as data objects. The access to the data objects is authenticated to allow for only approved users to access them.

Further, the hyperparameters can be set using a YAML-based configuration file information stored as metadata. A trained model can be served to be reused later for inference. This also would help with easy and efficient porting of the models to diverse HPC systems such as ThetaGPU, Polaris at ALCF, Summit, Frontier at OLCF and Cori, PerlMutter at NERSC.

B. Predicting Cosmological Parameters Efficiently

CosmoFlow [33] is a scalable TensorFlow-based deep learning framework to process sizeable 3D cosmology datasets on a modern HPC platform. The model aims to predict a couple of parameters from the distribution of matter. The dataset consists of simulation boxes of dark matter distribution. It is further augmented and pre-processed. HPCFAIR stores the script that converts the original input data from `.npy` to `.tfrecord`. Additionally, it stores hyperparameters in a script. The processed data can be stored as a versioned data object to account for the efforts required to regenerate the dataset.

VI. Discussion

With the proliferating AI research and development among the HPC and scientific community, the need for a platform to contain data objects in an easily findable, accessible format, enabling interconvertibility and reusability among various frameworks, is indispensable. HPCFAIR is an attempt in that direction. We have endeavored to abridge gaps in different state-of-the-art frameworks. We exhibited how HPCFAIR can assist development from a baseline DNN experiment to individual ML Libraries as reusable components, extending to reproduce research results and pipeline development. We also demonstrated how it could be leveraged to expedite and ease the design-space exploration.

HPCFAIR is our very first step in applying FAIR principles to HPC applications. Where we support only TF-ONNX interconversion, PyTorch-ONNX conversion at present, the eventual intention is to implement checkpointing conversion-like methodology. It will equip the user with the ability to make TF-PyTorch models interoperable. We also aim to have GUI providing better search capabilities. Along with these, we are constantly refining and implementing all the proposed features to proffer the scientific community a unified platform that will make the emerging workloads efficiently reusable and portable. The current implementation of HPCFAIR will be released as open source software on github once we get necessary approvals from our organizations.

Acknowledgment

This research was funded by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. It was performed under the auspices of the Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-826435). This work is also supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Program under Award Number DE-SC0021293.

REFERENCES

[1] "Collective knowledge framework," June 2021. [Online]. Available: http://cknowledge.org/

[2] T. be Findable, T. be Accessible, T. be Interoperable, and T. be Reusable, "Making nano data fair enough."

[3] W. Wang, B. Bleakley, C. Ju, V. Kyi, P. Tan, H. Choi, X. Huang, Y. Zhou, J. Wood, D. Wang *et al.*, "Aztec: A platform to render biomedical software findable, accessible, interoperable, and reusable," *arXiv preprint arXiv:1706.06087*, 2017.

[4] T. Weigel, U. Schwardmann, J. Klump, S. Bendoukha, and R. Quick, "Making data and workflows findable for machines," *Data Intelligence*, vol. 2, no. 1-2, pp. 40–46, 2020.

[5] J. Wise, A. G. de Barron, A. Splendiani, B. Balali-Mood, D. Vasant, E. Little, G. Mellino, I. Harrow, I. Smith, J. Taubert *et al.*, "Implementation and relevance of fair data principles in biopharmaceutical r&d," *Drug discovery today*, vol. 24, no. 4, pp. 933–938, 2019.

[6] M. D. Wilkinson, S.-A. Sansone, E. Schultes, P. Doorn, L. O. B. da Silva Santos, and M. Dumontier, "A design framework and exemplar metrics for fairness," *Scientific data*, vol. 5, no. 1, pp. 1–4, 2018.

[7] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 578–594.

[8] H. Vanholder, "Efficient inference with tensorrt," 2016.

[9] "Mlcube," June 2021. [Online]. Available: https://mlcommons.org/en/mlcube/

[10] R. Chard, Z. Li, K. Chard, L. Ward, Y. Babuji, A. Woodard, S. Tuecke, B. Blaiszik, M. J. Franklin, and I. Foster, "Dlhub: Model and data serving for science," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2019, pp. 283–292. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPS.2019.00038

[11] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[12] "Hugging face," June 2021. [Online]. Available: https://huggingface.co/

[13] "Tensorflow hub," June 2021. [Online]. Available: https://www.tensorflow.org/hub

[14] "Pytorch hub," June 2021. [Online]. Available: https://pytorch.org/hub/

[15] Gluoncv model zoo. [Online]. Available: https://cv.gluon.ai/model_zoo/index.htmlx

[16] Caffe model zoo. [Online]. Available: https://caffe.berkeleyvision.org/model_zoo.html

[17] Onnx model zoo. [Online]. Available: https://github.com/onnx/models

[18] "Apache arrow," June 2021. [Online]. Available: https://arrow.apache.org/

[19] "Mnist dataset," June 2021. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[20] "Sgemm gpu kernel performance data set," June 2021. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance

[21] C. Cummins, P. Petoumenos, Z. Wang, and H. Leather, "End-to-end deep learning of optimization heuristics," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 219–232.

[22] "Clgen," June 2021. [Online]. Available: https://chriscummins.cc/clgen/

[23] D. Grewe, Z. Wang, and M. F. O'Boyle, "Portable mapping of data parallel programs to opencl for heterogeneous systems," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2013, pp. 1–10.

[24] A. Magni, C. Dubach, and M. O'Boyle, "Automatic optimization of thread-coarsening for graphics processors," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, 2014, pp. 455–466.

[25] H. Xu, M. Emani, P.-H. Lin, L. Hu, and C. Liao, "Machine learning guided optimal use of gpu unified memory," in *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 2019, pp. 64–70.

[26] G. Verma, Y. Gupta, A. M. Malik, and B. Chapman, "Performance evaluation of deep learning compilers for edge inference," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 858–865.

[27] Imagenet. [Online]. Available: http://image-net.org/index

[28] Google. Tensorflow lite. [Online]. Available: https://www.tensorflow.org/lite

[29] NVIDIA. Tensorrt. [Online]. Available: https://developer.nvidia.com/tensorrt

[30] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström, "Json-ld 1.0," *W3C recommendation*, vol. 16, p. 41, 2014.

[31] "Onnx," June 2021. [Online]. Available: https://onnx.ai/

[32] Uno. [Online]. Available: https://github.com/ECP-CANDLE/Benchmarks/tree/develop/Pilot1/Uno

[33] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook *et al.*, "Cosmoflow: Using deep learning to learn the universe at scale," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 819–829.