

HPC Ontology: Towards a Unified Ontology for Managing Training Datasets and AI Models for High-Performance Computing

Chunhua Liao¹, Pei-Hung Lin¹, Gaurav Verma⁴, Tristan Vanderbruggen¹,
Murali Emani³, Zifan Nan², Xipeng Shen²

¹Lawrence Livermore National Laboratory, Livermore, CA 94550, USA

²North Carolina State University, Raleigh, NC 27695, USA

³Argonne National Laboratory, Lemont, IL 60439, USA

⁴Stony Brook University, Stony Brook, NY 11794, USA

Abstract—Machine learning (ML) techniques have been widely studied to address various challenges of productively and efficiently running large-scale scientific applications on heterogeneous supercomputers. However, it is extremely difficult to generate, access, and maintain training datasets and AI models to accelerate ML-based research. The Future of Research Communications and e-Scholarship has proposed the FAIR data principles describing Findability, Accessibility, Interoperability, and Reusability. In this paper, we present our ongoing work of designing an ontology for high-performance computing (named HPC ontology) in order to make training datasets and AI models FAIR. Our ontology provides controlled vocabularies, explicit semantics, and formal knowledge representations. Our design uses an extensible two-level pattern, capturing both high-level meta information and low-level data content for software, hardware, experiments, workflows, training datasets, AI models, and so on. Preliminary evaluation shows that HPC ontology is effective to annotate selected data and support a set of SPARQL queries.

Index Terms—Ontology, HPC, FAIR, datasets, AI models

I. INTRODUCTION

Due to the extreme heterogeneity and complexity of high-performance computing (HPC) node architectures, white-box analytical modeling techniques have become less tractable for analyzing and optimizing large-scale scientific applications. As an alternative, Artificial Intelligence (AI), especially machine learning (ML)-based techniques have been widely used to address various challenges in HPC, including those related to performance modeling and prediction [1]–[4], performance analysis [5]–[9], resilience [10], [11], data storage format selection [12], memory optimization [13], scheduling [14], and so on.

However, the HPC community’s consensus is that it is difficult to find, access, prepare, share, and reuse high-quality training datasets and AI models, as stated by a recent report of the Department of Energy’s Office of Science [15]. This is especially true when ML is applied to analyze and optimize large-scale HPC applications running on heterogeneous node architectures. Researchers spend a significant amount of effort using highly specialized tools (including compilers, performance tools, and runtime systems) to extract and process

training datasets from HPC systems. A range of machine learning frameworks is then used to generate AI models. Such datasets and AI models are stored in numerous formats, often without sufficient metadata to describe their semantics. Many datasets and models are underutilized. Part of the reason is that the community has not established standard processes to share the valuable datasets and the corresponding AI models. As a result, researchers and developers have to resort to costly repeated data collection processes. The HPC community cannot quickly build, evaluate, and reuse machine learning techniques to address pressing HPC challenges.

The problem the HPC community is facing is not unique. In many other research communities, researchers are establishing standard guidelines and recommending best practices to make scientific data Findable, Accessible, Interoperable, and Reusable (FAIR) [16]. Briefly, Findability means that data can be found online, typically through indexing in search engines. Accessibility indicates that data can be retrieved directly or via an approval process. Interoperability means that data follows standards. Finally, reusability denotes that the context of data generated (metadata) is documented so it can be compared to or integrated with other datasets.

This paper focuses on our preliminary work of developing a core component to enable FAIRness of training datasets and AI models for HPC: the HPC ontology, which is a collection of essential concepts and properties capturing information in the domain of HPC, using formal knowledge representation of Web Ontology Language (OWL) [17]. The HPC ontology provides a common vocabulary to describe various datasets and AI models. It also provides the semantics to enable the interoperability of heterogeneous data. In particular, we make the following contributions.

- We analyze the FAIR data principles and identify ontologies as essential enabling components.
- We create a set of guidelines for designing the HPC ontology, which takes into consideration the special requirements and constraints of making datasets and AI models FAIR in the HPC community.
- A high-level core ontology is designed to provide stan-

dard concepts and properties to annotate datasets, and AI models. This is required to facilitate data sharing and searching.

- A set of low-level supplemental components are presented to capture fine-grained information in various subdomains such as computers and performance datasets.
- Using a few use cases, we evaluate the benefits of the HPC ontology for annotating data and answer queries.

The remainder of this paper is organized as follows. In the next section, we will introduce the background information about the FAIR principles and the critical roles of ontologies. Section III presents the design philosophy of the HPC ontology. Section IV details the high-level concepts and properties of the HPC ontology, while Section V explains how more fine-grained information is organized in low-level components for various subdomains. We evaluate the current draft HPC ontology in Section VI. Related work is discussed in Section VII. Finally, Section VIII concludes the paper with a plan for future work.

II. BACKGROUND

We provide background information about the FAIR principles for sharing scientific data and how ontologies play an important role in achieving FAIRness.

A. The FAIR Data Principles

Accelerating scientific discoveries and innovations depend on a good ecosystem managing experiment data for various purposes such as validation and reusing. Unfortunately, existing scholar publication systems focus mostly on papers. The associated experiment data is either discarded or treated as a secondary citizen with lower standards for publishing. In response to the urgent need in the scientific community to improve the infrastructure supporting the reuse of scholarly data, the Future of Research Communications and e-Scholarship (FORCE11) proposed the FAIR data principles [16] describing four fundamental principles: Findability, Accessibility, Interoperability, and Reusability. These principles are further refined, as shown in Table I.

One of the main goals of the FAIR principles is to achieve machine-actionability in order to process the large amount of scholar data generated daily. This means that data management systems should provide rich and standard information such as the type of digital objects, their formats, licensing, and appropriate operations on them. To be practical, both contextual metadata surrounding a digital object and the content of the digital object should use controlled vocabularies, which in turn are associated with controlled semantics. As a result, several refined FAIR principles shown in Table I explicitly mention vocabularies (I2), knowledge representation (I1), and community standards (R1.3) to improve machine-actionability.

B. Ontologies

Ontologies can provide the much-needed controlled vocabularies, knowledge representation, and standards to implement

TABLE I
FAIR PRINCIPLES PROPOSED BY FORCE11 [16]

| | |
|-------------------------------|---|
| F: To be Findable | |
| F1 | (Meta)data are assigned a globally unique and persistent identifier |
| F2 | Data are described with rich metadata |
| F3 | Metadata clearly and explicitly include the identifier of the data they describe |
| F4 | (Meta)data are registered or indexed in a searchable resource |
| A: To be Accessible | |
| A1 | (Meta)data are retrievable by their identifier using a standardised communications protocol |
| A1.1 | The protocol is open, free, and universally implementable |
| A1.2 | The protocol allows for an authentication and authorisation procedure, where necessary |
| A2 | Metadata are accessible, even when the data are no longer available |
| I: To be Interoperable | |
| I1 | (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation |
| I2 | (Meta)data use vocabularies that follow FAIR principles |
| I3 | (Meta)data include qualified references to other (meta)data |
| R: To be Reusable | |
| R1 | (Meta)data are richly described with a plurality of accurate and relevant attributes |
| R1.1 | (Meta)data are released with a clear and accessible data usage license |
| R1.2 | (Meta)data are associated with detailed provenance |
| R1.3 | (Meta)data meet domain-relevant community standards |

the FAIR data principles. Ontology [18] is a concept originating in Philosophy, referring to the study of the nature of being, as well as the basic categories of them and their relations. In recent decades, it has become a formal way to explicitly represent knowledge in a domain. An ontology [18], [19] is a formal specification for explicitly representing knowledge about types, properties, and interrelationships of the entities in a domain. It provides a common vocabulary to represent and share domain concepts. Compared to tree-like taxonomy solely modeling the generalization-specialization relation, an ontology can form a much more complex graph with edges to model any kinds of relationships between entities (represented as graph nodes). Such graphs are often called knowledge graphs in many communities.

Figure 1 illustrates an example ontology for Robotics. Each node in the graph represents a concept or instance, and each edge carries a property indicating the relations between the two nodes it connects. For example, there are two nodes named “Agent” and “Object” indicating two concepts (or classes). The “is-a” edge between them means that one is a subclass of the other. An edge labeled “instanceOf” denotes that a node is an instance of a node representing a class. Another edge (labeled “hasPart”) between “Robot” and “Arm” indicates that the latter is a part of the former.

A formal language for expressing ontologies is the Web Ontology Language (OWL) [17]. OWL is based on the description logic (DL) and is expressive enough for building sophisticated knowledge bases while still supporting efficient inference. Resource Description Framework (RDF) is a fundamental format used to store a wide range of information, including ontologies written in OWL. Each piece of knowledge

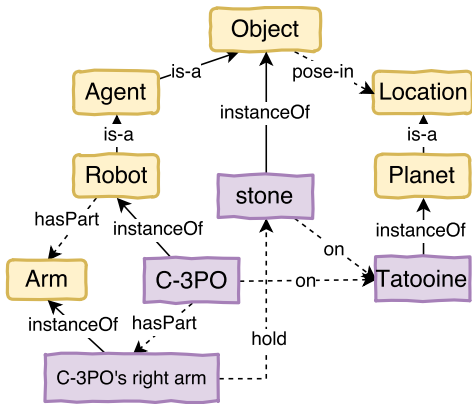


Fig. 1. An example ontology on robotics.

stored in RDF is represented as a triple, (*subject*, *property*, *object*). For instance, (“Tatooine”, *instanceOf*, “Planet”) states that *Tatooine* is an instance of *Planet*.

An ontology can be queried using a standard RDF query language named SPARQL. An example query in Listing 1 can be used to find all robots on the Tatooine planet using the Robotics ontology through the join of two properties. *rdf:type* is a standard RDF property to link a resource as an instance to its class. *:on* is the property to link a resource to its location.

Listing 1. Example SPARQL Query on the Robotics Ontology

```
1 SELECT ?s
2 WHERE { ?s rdf:type :Robot .
3         ?s :on :Tatooine }
```

Numerous ontologies or controlled vocabularies have been developed to enhance interoperability and reusability of data in different domains. For example, EDAM [20] is an ontology for bioinformatics operations and data types. Schema.org [21] is designed to improve the interoperability of web data. The Genomic Data Commons of National Cancer Institute [22] maintains standard terms and references to public ontologies and vocabularies in dictionary files to share linked clinical and genomic data. Brick [23] is an ontology to capture the common terms and relations in the domain of smart buildings.

In summary, the use of ontologies ensures “I”nteroperability and “R”eusability of the FAIR principles. In the context of making HPC training datasets and AI models FAIR, a supportive ontology is a natural prerequisite to provide a standard vocabulary of the HPC domain. However, there is a lack of effort of creating ontologies for the domain of machine learning-based program analysis and optimizations in high-performance computing (HPC). The focus of this paper is to contribute a design of such an ontology, namely HPC ontology.

III. DESIGN PHILOSOPHY

In order to support datasets and AI models used for HPC software analysis and optimizations, the HPC ontology has to cover a sufficiently large scope, including software, hardware, their interactions, as well as AI models. Developing a full-scale ontology to capture everything in HPC is impractical. It is also against the very idea of controlled vocabulary. There

is also a choice between developing an ontology from scratch and reusing existing ones. The key to the success of a domain-specific ontology is to have a carefully controlled scope so it is easy for users to learn while covering sufficient concepts and relationships of the selected domain. With these constraints in mind, we adopt a modular design with a two-level structure for the HPC ontology, with the following design principles.

a) Principle 1: We decide to have a set of high-level core components within the HPC ontology to capture high-level concepts and relationships used to describe entire datasets or AI models. This core ontology aims to provide metadata for users to annotate datasets and AI models as a whole. The metadata may include descriptive, administrative, and statistical information about datasets and models. We expect the high-level components are mostly stable for enhanced interoperability.

b) Principle 2: A set of low-level components are developed for representing fine-grain, internal information of various subdomains. These low-level components of the HPC ontology provide concepts and relationships details for smaller domains, such as program analyses, compilers, hardware features, performance tools, and their results, libraries, SDKs, AI models, and so on. The set of low-level components is extensible to accommodate the rapid evolvement of HPC software and hardware. They can be easily added, loaded, or phased out as supplemental components to the HPC ontology.

c) Principle 3: The reality is that most HPC researchers and developers are not familiar with ontology techniques. To make the HPC ontology easy to learn, we keep the core concepts and properties of the HPC Ontology into a single namespace (with a prefix named *hpc:*), without depending on names from other ontologies. This will relieve users from the burden of learning tens of other ontologies before they can use the HPC ontology to annotate basic data or form common queries.

d) Principle 4: For particular subdomains, if there are existing high-quality ontologies that are lightweight and easy to learn, we directly incorporate them as low-level, supplemental components. For instance, to encode content of scientific datasets, we have found that providing accurate unit information is indispensable. One example is execution duration. There are multiple units used in the field, such as seconds, milliseconds, microseconds and so on. After surveying the literature, we have found that the QUDT (Quantities, Units, Dimensions and Types) ontologies, originally developed by NASA researchers, already provide a good solution for encoding data with different units of measures. It also provides a mechanism to add extensions. Therefore, we directly use QUDT to encode units of scientific data.

e) Principle 5: To enable interoperability, we also link the content of HPC ontology with equivalent concepts and relationships from mainstream ontologies, such as Dublin Core metadata [24], Schema.org [21] and DBpedia [25]. Users who are already familiar with other ontologies can still use the same terms. SPARQL query engines can easily process queries using

popular vocabularies. This makes datasets and AI models annotated by HPC ontology part of the linked data ecosystem.

f) *Principle 6*: Although there is some progress for techniques used to automatically generate ontologies, they are still immature. We decided to create the content of the HPC ontology manually. This will give us more control over the set of concepts and relationships to be included or excluded. We can also curate the class hierarchies of different subdomains, leveraging the extensive human experiences of our team in HPC.

g) *Principle 7*: We use an incremental and use case driven approach to develop the HPC ontology. The initial version will be developed using a set of simple use cases. Later versions will be developed with increasingly more complex use cases combined with more strict evaluation criteria including size, completeness, accuracy, consistency, correctness, conformity to community standards, persistence, accessibility, and so on.

IV. HIGH-LEVEL CORE ONTOLOGY

In this section, we present representative high-level concepts and their corresponding properties of the HPC ontology.

A. Basic Scenario and Naming Convention

A basic scenario described by the HPC ontology is that *some people who are members of a project used some software and hardware to conduct some experiments, which in turn used some input data to generate training datasets or AI models*. The semantics can be mapped to three high-level concepts, including Agent, Activity, and Artifact. Essentially, *some Agent conducted some Activity which used some Artifact as input and generated some other Artifact*.

As shown in Figure 2, we follow a common naming convention when defining vocabularies in the HPC ontology: Singular nouns in CamelCase are used to indicate a Class. Multiword names are written without any spaces but with each word written in uppercase. Relationship (or Property) names start with lowercase letters. For example, *hpc:Project* means a class while *hpc:project* indicates a property that links some data with its associated project.

Dashed arrows in the figure indicate the *isA* relation between a subclass and its superclass. For instance, both training datasets and AI models are kind of data in this context. Solid line arrows indicate other relationships. Some arrows have only a single label to denote a single relationship. To simplify the diagram, inverse relationships are combined in a single arrow with a pair of relationship labels. For example, the label of the arrow between Person and Project includes both *hpc:memberOf* and *hpc:member*. Not all edges are shown in the figure to avoid a cluttered figure.

B. Top level Concept: Thing

To provide provenance of all information, the very top level concept in the HPC ontology, *hpc:Thing*, is associated with a set of fundamental properties (listed in Table II) about its unified resource identifier, the type of the ID (such as Open

Researcher and Contributor ID and Digital Object Identifier), name, URL, etc. Any other concepts (from both high and low levels) are direct or indirect subclasses of *hpc:Thing*. They naturally inherit all the fundamental properties of *hpc:Thing*. The *hpc:Thing* node and its edges are not shown in Figure 2 to simplify the figure.

TABLE II
PROPERTIES OF THE THING CLASS

| Property | Data-type | Description |
|--------------------------|--------------|--|
| <i>hpc:id</i> | xsd:anyURI | URI of the thing |
| <i>hpc:idType</i> | xsd:string | Type of the ID, such as ORCID, DOI, etc. |
| <i>hpc:name</i> | xsd:string | Name of the thing |
| <i>hpc:alternateName</i> | xsd:string | An alias for this item |
| <i>hpc:description</i> | xsd:string | Short description |
| <i>hpc:url</i> | xsd:anyURI | URL of the official website of a thing |
| <i>hpc:submitter</i> | xsd:anyURI | Who submits this piece of info. |
| <i>hpc:submitDate</i> | xsd:dateTime | Date of submission |

C. Activity and Experiment

As shown in Figure 2, the centerpiece of this design is the concept of Activity, which connects to many other concepts through properties such as *hpc:used*, *hpc:generated*, *hpc:wasAssociatedWithSoftware*, *hpc:usedWorkflow*, *hpc:wasConductedBy*, and so on. An activity is something that occurs over a period of time. An activity could happen after another one, linked using *hpc:wasPrecededBy*. Experiment is a subclass of Activity to represent HPC experiments.

Table III lists major properties of the Experiment class. For convenience, we also define equivalent properties as needed. For example, *hpc:used* is the same as *hpc:hadInput*. *hpc:wasAWS* is a short name for *hpc:wasAssociatedWithSoftware*. *hpc:wasAWS* is equivalent to *hpc:wasAssociatedWithHardware*. There is also the *wasPrecededBy* property to link a sequence of experiments.

TABLE III
MAJOR PROPERTIES OF THE EXPERIMENT CLASS

| HPC Ontology Property | Data-type | Description |
|---------------------------|--------------|---|
| <i>hpc:used</i> | xsd:anyURI | Input data, == <i>hpc:hadInput</i> |
| <i>hpc:generated</i> | xsd:anyURI | Generated data, == <i>hpc:hadOutput</i> |
| <i>hpc:usedWorkflow</i> | xsd:anyURI | Workflow used |
| <i>hpc:wasPrecededBy</i> | xsd:anyURI | Experiments before this one |
| <i>hpc:wasConductedBy</i> | xsd:anyURI | Link to persons |
| <i>hpc:wasAWS</i> | xsd:anyURI | Software used |
| <i>hpc:wasAWH</i> | xsd:anyURI | Hardware used |
| <i>hpc:startDate</i> | xsd:dateTime | Start time |
| <i>hpc:endDate</i> | xsd:dateTime | End time |
| <i>hpc:project</i> | xsd:anyURI | Associated projects |

D. Artifact and Data

We define Data as a subclass of Artifact and further categorize it into Workflow, Training Dataset, and AI model. We have found that in practice, any combinations of mixed scripts, datasets, and AI model files are shared and reused. They are just vaguely categorized as Data currently. In the HPC software analysis and optimization domain, software itself is also a kind of artifact that can be used as input or output

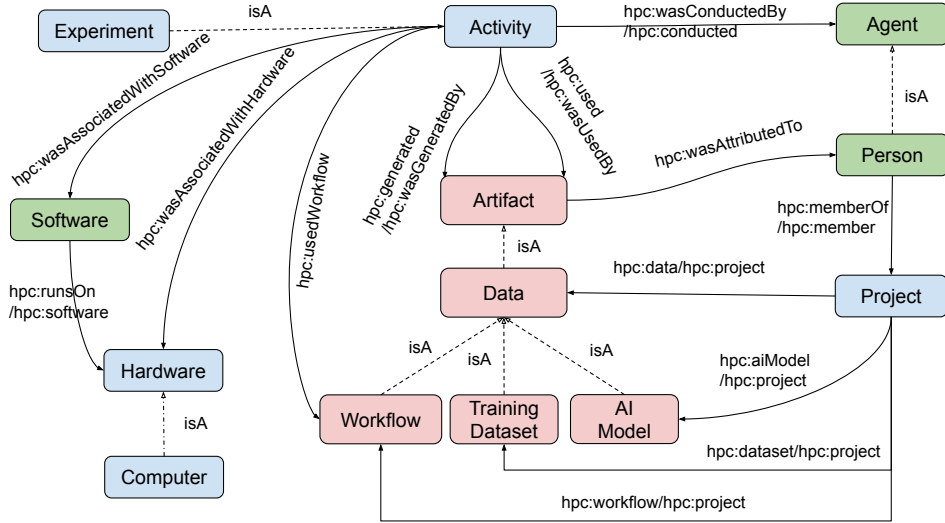


Fig. 2. Major High-level Concepts and Relationships of the HPC Ontology

data. For example, many compilers, tools and workflows process software as input data to generate program analysis information. So there is a property link (*isA*) between Software and Data also. Again, this edge is not shown in Figure 2 to avoid cluttering.

Figure 3 shows the partial class hierarchy rooted at Artifact. *AIModel* is equivalent to *MachineLearningModel* in the figure.

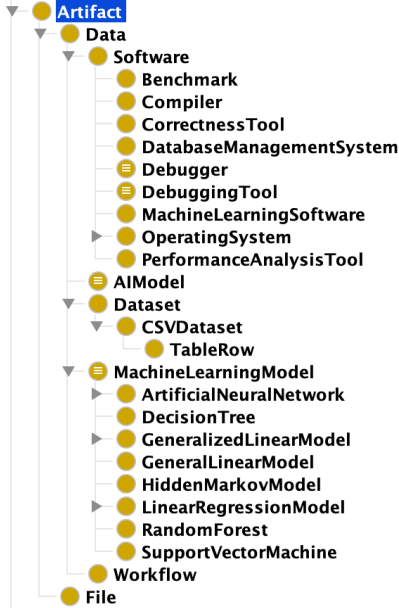


Fig. 3. Partial class hierarchy of artifact

Table IV shows essential properties for the Data class. There are properties about license, version, associated projects, as well as experiments generating or using the data. Note that Data can be associated with one or more files through the *hpc:file* property. Each file in turn has its properties

such as name, size, format, MD5, URL, etc. Data can be derived from other data in a sequence of experiments. So we have *hpc:wasDerivedFrom* to indicate such a property. In the domain of program analysis and optimization, Data generated often has one or more focus applications or machines. We introduce *targetApplication* and *targetMachine* to directly support such links.

TABLE IV
MAJOR PROPERTIES OF THE DATA CLASS

| Property | Datatype | Description |
|------------------------------|------------|---------------------------------------|
| <i>hpc:license</i> | xsd:string | License of the data |
| <i>hpc:version</i> | xsd:string | Version number |
| <i>hpc:subject</i> | xsd:string | Performance modeling, optimization... |
| <i>hpc:file</i> | xsd:anyURI | Associated files |
| <i>hpc:project</i> | xsd:anyURI | Link to the associated projects |
| <i>hpc:wasGeneratedBy</i> | xsd:anyURI | output of experiments |
| <i>hpc:wasUsedBy</i> | xsd:anyURI | Input to experiments |
| <i>hpc:wasDerivedFrom</i> | xsd:anyURI | Some data was derived from other |
| <i>hpc:targetApplication</i> | xsd:anyURI | Applications being targeted |
| <i>hpc:targetMachine</i> | xsd:anyURI | Computers being targeted |

For Dataset and AI Model classes, they inherit all properties of their superclass Data. They also have additional properties as needed. For example, the AI Model class shown in Table V has extra properties such as source artifacts (*hpc:wasDerivedFromDataset*) used to generate the model, machine learning framework used (*hpc:machineLearningFramework*), configurable parameters (*hpc:params*), accuracy (*hpc:accuracy*), and so on. Note that *hpc:wasDerivedFromDataset* (shown in its abbreviation form *hpc:wasDFD*) is a subclass (or a subproperty in ontology's term) of *hpc:wasDerivedFrom*.

E. Software

A piece of software (or a program) can be associated with lots of details. We have to narrow down the scope to

TABLE V
MAJOR PROPERTIES OF THE AI MODEL CLASS

| Property | Datatype | Description |
|-------------------------|-------------|--|
| hpc:modelFormat | xsd:string | Such as protobuf, onnx, h5 |
| hpc:isTunable | xsd:boolean | Tunable during the runtime or not |
| hpc:params | xsd:string | Configurable parameters |
| hpc:framework | xsd:anyURI | Such as TF, PyTorch, MXNet, etc. |
| hpc:modelType | xsd:string | Relevant problem domains |
| hpc:supportsAccelerator | xsd:boolean | Run on accelerators or not |
| hpc:accuracy | xsd:double | Accuracy of the model |
| hpc:overhead | xsd:double | Overhead of the model |
| hpc:wasDFD | xsd:string | Datasets used to generate this model |
| hpc:learningType | xsd:string | Supervised, reinforcement learning, etc. |
| hpc:learningAlgorithm | xsd:string | Decision tree, random forest, etc. |
| hpc:hyperParameter | xsd:dict | Hyperparameters such as batch size |
| hpc:modelProperty | xsd:dict | Such as number and types of layers |
| hpc:inputShape | xsd:string | Input shape to be fed to the model |
| hpc:inputDatasetFormat | xsd:string | Input format of the dataset |

support generic program analyses and optimizations. In the HPC ontology class hierarchy, Software is a subclass of Data so it has all of Data’s properties. We have added a few popular subclasses of Software, including Compiler, OperatingSystem, Benchmark, and so on as shown in Figure 3. In addition, we add the properties shown in Table VI to support brief information about a piece of software or benchmark being analyzed or optimized.

TABLE VI
MAJOR PROPERTIES OF A SOFTWARE PROGRAM/BENCHMARK

| Property | Datatype | Description |
|-----------------------|------------|----------------------------|
| hpc:programLanguage | xsd:string | Programming languages used |
| hpc:operatingSystem | xsd:string | OS classes supported |
| hpc:firstReleaseDate | xsd:date | First release date |
| hpc:latestReleaseDate | xsd:date | Latest release date |
| hpc:vendor | xsd:string | Link to vendors |
| hpc:runsOn | xsd:anyURI | Machines runs the software |
| hpc:input | xsd:anyURI | Input data of the software |
| hpc:output | xsd:anyURI | Generated output data |

F. Hardware and Computer

The HPC ontology must capture sufficient hardware and computer information to be useful. An HPC computing system can be a single node computer (including a workstation or server) or a cluster with a set of computers. The computer class in the HPC ontology collects the properties describing hardware, performance, and manufacturing related details, as shown in Table VII. Note that some properties should link to structured QUDT data types to encode both values and accurate units. For example, the *hpc:harddriveSize* property has *qudt:QuantityValue* to capture sizes in different units such as GigaBytes or TeraBytes.

We also model CPUs and coprocessors used by a computer. They are linked from a computer object through *hpc:cpu* and *hpc:coprocessor* respectively to provide more information. Table VIII shows some basic CPU class’s properties. Given the fast changing nature of CPUs and coprocessors (such as Nvidia GPUs), we model their details in low-level supplemental components with vendor-specific terms and properties. We don’t put them into the core, high-level ontology.

TABLE VII
MAJOR PROPERTIES OF THE COMPUTER CLASS

| Property | Datatype | Description |
|-----------------------------|--------------------|------------------------------|
| hpc:vendor | xsd:string | Vendor of the machine |
| hpc:cpu | xsd:anyURI | Associated CPUs |
| hpc:cpuCoresPerNode | xsd:integer | CPU core count |
| hpc:threadsPerNode | xsd:integer | Threads count |
| hpc:coprocessor | xsd:anyURI | Associated coprocessors |
| hpc:coprocessorCoresPerNode | xsd:integer | Coprocessor core count |
| hpc:memorySize | qudt:QuantityValue | Total memory size |
| hpc:harddriveSize | qudt:QuantityValue | Total hard drive size |
| hpc:totalPeakPerformance | qudt:QuantityValue | The peak performance |
| hpc:hasOperatingSystem | xsd:string | The operating system |
| hpc:hasCompiler | xsd:string | The compilers available |
| hpc:power | qudt:QuantityValue | Power consumption |
| hpc:powerEfficiency | qudt:QuantityValue | Such as GFlops/Watts |
| hpc:hasRmax | qudt:QuantityValue | Obtained peak perf. |
| hpc:hasRpeak | qudt:QuantityValue | Theoretical peak |
| hpc:dateCommissioned | xsd:date | Commission date |
| hpc:site | xsd:string | Hosting facility/institution |
| hpc:country | xsd:string | Located country |

TABLE VIII
MAJOR PROPERTIES OF THE CPU CLASS

| Property | Datatype | Description |
|------------------------------|--------------------|-----------------------|
| hpc:processorTech | xsd:string | Codename/model |
| hpc:processorGeneration | xsd:string | Processor generation |
| hpc:processorCorePerSocket | xsd:integer | Cores in a socket |
| hpc:cpuFrequency | qudt:QuantityValue | Frequency |
| hpc:memoryBandwidth | qudt:QuantityValue | Memory bandwidth |
| hpc:processorPeakPerformance | qudt:QuantityValue | Processor performance |
| hpc:vendor | xsd:string | Link to vendors |

Cluster is another class in the HPC ontology, inheriting properties from the Computer subclass, to cover additional properties needed for cluster systems. Each cluster object can be linked to one or more computer objects through its *hasNode* property. Table VII shows major properties of the Cluster class. Given the importance of the Top500 project, a cluster may have related Top500 ranking information (e.g. *hpc:top500Rank*).

TABLE IX
MAJOR PROPERTIES OF THE CLUSTER CLASS

| Property | Datatype | Description |
|---------------------------------|--------------------|------------------------|
| hpc:totalClusterCPUCoreCount | xsd:integer | Total CPU cores number |
| hpc:systemArchitecture | xsd:string | MPP, cluster, etc.) |
| hpc:computeNodeCount | xsd:integer | Computer node count |
| hpc:gpuNodeCount | xsd:integer | Number of GPU nodes |
| hpc:top500Rank | xsd:integer | Top500 ranking |
| hpc:top500nmax | xsd:integer | Problem size used |
| hpc:totalClusterMemorySize | qudt:QuantityValue | Total memory |
| hpc:totalClusterPeakPerformance | qudt:QuantityValue | Total peak performance |
| hpc:hasNode | xsd:anyURI | Login or compute node |

There are a few other high-level concepts such as Person and Project to help describe the semantics of HPC datasets and AI models. We omit their details since their properties are trivial.

V. LOW-LEVEL COMPONENTS OF HPC ONTOLOGY

Low-level components of the HPC Ontology provide fine-granularity concepts and properties to describe subdomains,

including hardware details, contents of profiling datasets, and internal details of AI models. They are provided as needed to achieve maximal FAIRness by providing rich attributes to describe data elements. In this paper, we focus on several example subdomains relevant to the scope of our work.

A. A Coprocessor: NVIDIA GPU

For heterogeneous architectures, Nvidia’s GPUs are popular components of many supercomputers. Six out of the top ten most powerful computers use Nvidia GPUs, as shown in June 2021’s Top500 list [26]. We have added a low-level GPU component into HPC ontology to model the properties of GPUs. Similar low-level components can be added in the future to support other types of heterogeneous processors such as TPU, neuromorphic processors, FPGAs and so on.

Table X shows major properties of a NVIDIA GPU. The properties can be divided into two sets: one is the set of fixed properties of the GPU such as *hpc:theoreticalGPUOccupancy*. The other set includes configurable properties such as *hpc:gpuThreadBlockSize* used to indicate the thread block size configured during a kernel launch.

B. A GPU Performance Dataset: XPlacer

When building AI models for program analyses and optimizations, performance profiling information is often a critical part of the corresponding training datasets. We select a dataset from XPlacer [13] as an example. The dataset contains GPU performance profiling data used to build models guiding GPU memory placement choices for arrays.

A key observation here is that program optimizations often happen at the kernel (or function) level. So we provide properties associated with kernel performance. Table XI lists properties to capture Nvidia GPU’s performance profiling information. The properties can be grouped into several categories, including basic information about a kernel and its major data objects (arrays), execution time, and hardware counter based information such as cache utilization rate, page faults, data transfer sizes. Finally, for optimization problems selecting optimal code variants, we provide *hpc:codeVariant* and *hpc:bestCodeVariant* to annotate labels.

VI. PRELIMINARY RESULTS

This section presents some preliminary use cases using the current draft HPC Ontology, including providing standard metadata for annotating various data and answering questions posed as SPARQL queries.

An AWS machine is used to run experiments. It has 4 cores of Intel Xeon CPU E5-2676 v3 running at 2.40GHz and 16 GB main memory. The evaluation uses a set of tools. Protege v5.5.0 is used for ontology development. Typically raw data is stored in CSV files. Tarql (git hash #b06b4dd) [27] is used to automatically convert CSV files into RDF triples saved into N-Triples (.NT) files. Tarql’s conversion rules are manually specified using SPARQL 1.1 syntax. Blazegraph v2.1.6 is used as the graph database supporting standard RDF/SPARQL

APIs. All N-Triples files are loaded into a namespace of Blazegraph in a triples mode and inference turned on.

JSON-LD, a JSON-based format storing Linked Data, is used as the main format to present annotated data. JSON-LD can be viewed as JSON plus Context and ID information. The context is used to define the short-hand names. IDs are unique name identifiers to describe keys in the document. These IDs are used as the keys of JSON’s key-value pairs to unambiguously encoding information linked to formally defined entities in an ontology. We use a Python script calling RDFLib APIs to generate the JSON-LD output from .NT files.

A. Providing Metadata for Top500 Supercomputers

Machine properties are fundamental information in HPC. We use the concepts and properties of the HPC ontology to annotate the tabular information of the fastest machine, Supercomputer Fugaku, published in June 2021’s Top500 list [26].

The HPC Ontology has all the concepts and properties needed to fully annotate the content of the entire top 500 spreadsheet. Listing 2 shows various example properties of Fugaku in JSON-LD format. All properties are stored in key-value pairs. The machine is uniquely identified by its system ID (179807) assigned by the Top500 website. The HPC ontology provides vocabularies of property names, such as *hpc:name*, *hpc:Cluster*, and *hpc:power*. The QUDT ontology provides structured values with detailed information for the units used, such as MegaHZ and KiloWatt. Note that QUDT originally did not have the TeraFLOPS unit; we have extended it to support this.

Listing 2. Example Fugaku’s Information Annotated Using HPC Ontology

```

1 {
2   "@id": "https://www.top500.org/system/179807",
3   "@type": "hpc:Cluster",
4   "hpc:name": "Supercomputer Fugaku",
5   "hpc:top500Rank": 1,
6   "hpc:vendor": "Fujitsu",
7   "hpc:country": "Japan",
8   "hpc:cpuArchitecture": "Fujitsu ARM",
9   "hpc:cpuFrequency": { "@id": "._:B9e8a3" },
10  "hpc:hasOperatingSystem": "Red Hat Enterprise Linux",
11  "hpc:hasRmax": { "@id": "._:B17f29" },
12  "hpc:power": { "@id": "._:N15c5a" },
13  "hpc:processorCorePerSocket": 48,
14  "hpc:processorGeneration": "Fujitsu A64FX",
15  "hpc:site": "RIKEN Center for Computational Science",
16  "hpc:systemArchitecture": "MPP",
17  "hpc:systemModel": "Supercomputer Fugaku",
18  "hpc:totalClusterCPUCoreCount": 7630848
19 },
20
21 { "@id": "._:B17f29",
22   "@type": "qudt:QuantityValue",
23   "qudt:unit": { "@id": "http://qudt.org/vocab/unit/
24     TeraFLOPS" },
25   "qudt:value": "442010.00"
26 },
27 { "@id": "._:B9e8a3",
28   "@type": "qudt:QuantityValue",
29   "qudt:unit": { "@id": "http://qudt.org/vocab/unit/MegaHZ"
30     },
31   "qudt:value": "2200"
32 },
33 { "@id": "._:N15c5a",
34   "@type": "qudt:QuantityValue",

```

TABLE X
MAJOR PROPERTIES OF A NVIDIA GPU

| Property | Datatype | Description |
|--|--------------------|---|
| hpc:dramFrequency | qudt:QuantityValue | Frequency of DRAM |
| hpc:streamingMultiprocessorFrequency | qudt:QuantityValue | Frequency of Streaming multiprocessor |
| hpc:activeCyclesOfStreamingMultiprocessor | xsd:integer | Active cycle counts from SM |
| hpc:theoreticalActiveWarpsPerSM | xsd:integer | Theoretical Active Warps per SM |
| hpc:theoreticalGPUOccupancy | xsd:double | Theoretical Occupancy |
| hpc:maxGPUThreadBlockSizeLimitedBySM | xsd:integer | Max block limited by SM |
| hpc:maxGPUThreadBlockSizeLimitedByRegister | xsd:integer | Max block limited by registers |
| hpc:maxGPUThreadBlockSizeLimitedBySharedMemory | xsd:integer | Max block limited by shared memory |
| hpc:maxGPUThreadBlockSizeLimitedByWarps | xsd:integer | Max block limited by warps |
| hpc:gpuThreadBlockSize | xsd:integer | Launch block size |
| hpc:gpuThreadGridSize | xsd:integer | Launch grid size |
| hpc:registersPerThread | xsd:integer | Launch register//thread |
| hpc:gpuSharedMemoryConfigurationSize | qudt:QuantityValue | Launch shared memory configuration size |
| hpc:gpuStaticSharedMemorySizePerBlock | qudt:QuantityValue | launch static shared memory |
| hpc:gpuThreadCount | xsd:integer | GPU thread count |
| hpc:gpuWavesPerSM | xsd:integer | Launch wave per SM |
| hpc:gpuUnifiedMemoryRemoteMapSize | qudt:QuantityValue | Unified memory remote map |

TABLE XI
MAJOR PROPERTIES OF KERNEL PERFORMANCE PROFILING DATA

| Property | Datatype | Description |
|--------------------------------|--------------------|--------------------------|
| Kernel Information | | |
| hpc:kernelName | xsd:string | Kernel/function name |
| hpc:benchmark | xsd:anyURI | Associated benchmark |
| hpc:commandLineOption | xsd:string | Command line Options |
| hpc:arrayName | xsd:string | Name of array |
| hpc:allocatedDataSize | qudt:QuantityValue | Memory allocation size |
| hpc:beginMemoryAddress | xsd:string | Beginning array address |
| hpc:endMemoryAddress | xsd:string | Ending array address |
| Performance Information | | |
| hpc:cycle | xsd:integer | Profiled cycle count |
| hpc:executionTime | qudt:QuantityValue | Execution time |
| hpc:numberOfCalls | xsd:integer | Number of calls |
| hpc:averageExecutionTime | qudt:QuantityValue | Average execution time |
| hpc:minExecutionTime | qudt:QuantityValue | Min. execution time |
| hpc:maxExecutionTime | qudt:QuantityValue | Max. execution time |
| hpc:executionTimePercentage | xsd:double | Percentage of time spent |
| hpc:memoryThroughputRate | xsd:double | Memory Throughput |
| hpc:dramUtilizationRate | xsd:double | DRAM utilization rate |
| hpc:l1CacheUtilizationRate | xsd:double | L1/Tex utilization rate |
| hpc:l2CacheUtilizationRate | xsd:double | L2 utilization rate |
| hpc:achievedGPUOccupancy | xsd:double | GPU occupancy |
| hpc:achievedActiveWarpsPerSM | xsd:integer | Active warps per SM |
| hpc:cpuPageFault | xsd:integer | CPU page fault count |
| hpc:gpuPageFault | xsd:integer | GPU page fault count |
| hpc:hostToDeviceTransferSize | qudt:QuantityValue | HostToDevice transfer |
| hpc:deviceToHostTransferSize | qudt:QuantityValue | DeviceToHost transfer |
| Labels | | |
| hpc:codeVariant | xsd:integer | Code variant ID |
| hpc:labeledCodeVariant | xsd:integer | Best variant ID |

Listing 3 shows a benchmark software package [28] which was used as an input dataset for a code similarity analysis experiment. It is easy to identify metadata for its name, license, subject, related project, and so on.

Listing 3. Example Dataset Annotated Using HPC Ontology

```

1 {
2   "@id": "http://example.org/dataset/DA000005",
3   "@type": "hpc:Dataset",
4   "hpc:description": "microbenchmark kernels in C/C++ and
      Fortran",
5   "hpc:hasIDType": "System Generated",
6   "hpc:license": "BSD",
7   "hpc:name": "DataRaceBench micro-benchmarks",
8   "hpc:project": {
9     "@id": "http://example.org/project/PR000002"
10  },
11  "hpc:subject": [
12    "OpenMP",
13    "Data Race Detection",
14    "Computer Science"
15  ],
16  "hpc:url": "https://github.com/LLNL/dataracebench/tree/
      master/micro-benchmarks",
17  "hpc:version": "1.3.2"
18 }

```

Listing 4 shows metadata for an AI model released by a study [13]. Besides common metadata for its name and type, it additionally has metadata specific to this AI model, such as *hpc:wasDerivedFromDataset*, *hpc:learningAlgorithm*, and *hpc:targetMachine*.

Listing 4. Example AI model annotated using HPC ontology

```

1 {
2   "@id": "http://example.org/AIModel/MO000001",
3   "@type": "hpc:AIModel",
4   "hpc:contributor":
5     { "@id": "http://example.org/person/PI000013" },
6   "hpc:description": "Onnx version of the decision tree
      model for XPlacer",
7   "hpc:wasGeneratedBy":
8     { "@id": "http://example.org/experiment/EX000002" },
9   "hpc:hasIDType": "System Generated",
10  "hpc:wasDerivedFromDataset": [
11    { "@id": "http://example.org/dataset/DA000001" },
12    { "@id": "http://example.org/dataset/DA000002" },
13    { "@id": "http://example.org/dataset/DA000003" },
14    { "@id": "http://example.org/dataset/DA000004" }
15  ],
16  "hpc:isTunable": false,
17  "hpc:learningType": "Supervised ",
18  "hpc:modelFormat": "ONNX",

```

B. Providing Metadata for Datasets and AI Models

We are developing a web portal to allow users to submit information about training datasets and AI models in HPC. To make the submitted data compliant with the FAIR principals, standard metadata must be used to annotate the datasets and AI models. The HPC ontology can serve as the metadata for this purpose.

```

35   "qudt:unit": { "@id": "http://qudt.org/vocab/unit/KiloW" },
36   "qudt:value": "29899.23"
37 }

```



```

19 "hpc:modelType": "Prediction",
20 "hpc:learningAlgorithm": "Decision Tree",
21 "hpc:name": "decisionTree.onnx",
22 "hpc:project":
23   {"@id": "http://example.org/project/PR000001"},
24 "hpc:subject": [
25   "GGPU",
26   "Heterogeneous Systems",
27   "Data Placement",
28   "Decision Tree"
29 ],
30 "hpc:supportsAccelerator": true,
31 "hpc:targetMachine":
32   {"@id": "http://example.org/cluster/lassen"},
33 "hpc:url": "https://github.com/xyz/decisionTree.onnx"
34 },
35 "hpc:version": "1.2.1"
36 }

```

C. Encoding GPU Profiling Dataset Stored in a CSV File

Low-level components of the HPC ontology can be used to encode the internal content of datasets or AI models. Listing 5 shows the annotation of a row of a CSV file representing a dataset released by a study [13]. Each row represents information about an array accessed within a kernel code variant and the kernel’s associated profiling metrics such as GPU data transferring data sizes and page faults. The dataset was used to train a model deciding which code variant delivers the best performance. Using properties defined in Table XI, the annotated json-ld file contains accurate, machine readable information for each cell, which facilitates interoperability and reusability of the dataset.

Listing 5. Example profiling dataset annotated using HPC ontology

```

1 {
2   "@id": "https://github.com/xyz/rodinia_3.1
3     _cuda_bfs_datalevel-lassen.csv#L11",
4   "@type": "hpc:TableRow",
5   "hpc:BeginMemoryAddress": "0x200060080000",
6   "hpc:EndMemoryAddress": "0x200060090000",
7   "hpc:allocatedDataSize": 65536,
8   "hpc:arrayID": "1",
9   "hpc:arrayName": "h_graph_mask",
10  "hpc:codeVariant": "111100",
11  "hpc:commandLineOption": "graph65536",
12  "hpc:cpuPageFault": 2,
13  "hpc:deviceToHostTransferSize": {
14    "@id": "._:Nca485"
15  },
16  "hpc:hostToDeviceTransferSize": {
17    "@id": "._:Na7702" }
18 },
19 {
20   "@id": "._:Na7702",
21   "@type": "qudt:QuantityValue",
22   "qudt:unit": {
23     "@id": "http://qudt.org/vocab/unit/KiloBYTE" },
24   "qudt:value": {
25     "@type": "http://www.w3.org/2001/XMLSchema#decimal",
26     "@value": "192.0" }
27 },
28 {
29   "@id": "._:Nca485",
30   "@type": "qudt:QuantityValue",
31   "qudt:unit": {
32     "@id": "http://qudt.org/vocab/unit/KiloBYTE"
33   },
34   "qudt:value": {
35     "@type": "http://www.w3.org/2001/XMLSchema#decimal",
36     "@value": "0.0"
37   }
38 }
39 }

```

D. Enabling Various Queries

A standard approach to evaluating an ontology is to check if the ontology can be used to formulate questions asked by intended users. Such questions often are called competency questions. We anticipate some typical questions from a HPC user may include the following:

- Q1: What are the ids of datasets from a research project named “Xplacer”?
- Q2: What are the names of AI models available for a supercomputer named “lassen”?
- Q3: What AI models are available for machines with GPUs named “Nvidia V100”?
- Q4: What datasets of projects funded by NSF are available for building Performance Prediction models?
- Q5: What workflows are available for generating AI models guiding “Heterogeneous Mapping” of a benchmark (e.g. the NAS Parallel Benchmark or NPB) running on a machine using AMD GPUs?

We populated a Blazegraph RDF database with information encoding a few example datasets and AI models found in the literature, assuring diversified datasets and models across widely researched domains. We gathered data from projects like XPlacer [13] that come with the workflows defining dataset collection and offer models to perform experiments. Further, there were scenarios when researchers publish their results as a package or complete tool. The ProGraML [29] and MLGO [30] are such considered examples. Additionally, we included literature from HPC Energy Research [31] providing datasets or models alone. We also collected some datasets and refined models hosted at Kaggle. They are related to the GPU Kernel Performance [32]–[34].

Listing 6 shows the SPARQL queries sent to the Blazegraph database and the corresponding answers obtained, for Q1 through Q3. Listing 7 shows queries and results for Q3 and Q4. The results show that HPC Ontology is complete to support the concepts and properties expressed in these queries which in turn obtain desired results.

Listing 6. SPARQL queries to answer competency questions 1-3

```

1 PREFIX hpc: <https://example.org/HPC-Ontology#>
2 # Query for Q1: dataset ids of a project
3 #-----
4 SELECT ?ds
5 WHERE { ?pid rdf:type hpc:Project .
6         ?pid hpc:name "Xplacer" .
7         ?pid hpc:dataset ?ds }
8
9 # Results: IDs of datasets
10 # <http://example.org/dataset/DA000001>
11 # <http://example.org/dataset/DA000002>
12 # <http://example.org/dataset/DA000003>
13 # <http://example.org/dataset/DA000004>
14
15 # Query for Q2: AI models' names of a supercomputer
16 #-----
17 SELECT ?model_name
18 WHERE { ?model_id rdf:type hpc:AIModel .
19         ?model_id hpc:targetMachine
20           <http://example.org/cluster/lassen> .
21         ?model_id hpc:name ?model_name }
22
23 # Results : names of AI models
24 # decisionTree.onnx
25 # randomForest.onnx

```

```

26
27 # Query for Q3: machines with NVidia V100 and their models
28 #-----
29 SELECT ?machine ?model_id
30 WHERE { ?gpu rdf:type hpc:GPU .
31         ?gpu hpc:name "Nvidia V100".
32         ?machine hpc:coprocessorModel ?gpu .
33         ?model_id hpc:targetMachine ?machine .
34         ?model_id rdf:type hpc:AIModel .
35         ?model_id hpc:name ?model_name }
36
37 # Results: machine names and AI model IDs
38 <http://example.org/cluster/lassen> <http://example.org/
39 AIModel/MO000001>
40 <http://example.org/cluster/lassen> <http://example.org/
41 AIModel/MO000002>

```

Listing 7. SPARQL queries to answer competency questions 4-5

```

1 PREFIX hpc: <https://example.org/HPC-Ontology#>
2
3 # Query for Q4: NSF project's datasets for building
4 performance prediction models
5 #-----
6 SELECT ?project_id ?ds_id
7 WHERE {
8     ?project_id rdf:type hpc:Project .
9     ?project_id hpc:fundedBy "the National Science
10 Foundation" .
11     ?ds_id hpc:project ?project_id .
12     ?ds_id rdf:type hpc:Dataset .
13     ?ds_id hpc:subject "Performance Prediction" }
14
15 # Results: project IDs and dataset IDs
16 <http://example.org/project/PR000003> <http://example.org/
17 dataset/DA000007>
18 <http://example.org/project/PR000003> <http://example.org/
19 dataset/DA000008>
20 <http://example.org/project/PR000003> <http://example.org/
21 dataset/DA000009>
22
23 # Query of Q5: workflow generating AI models for NPB's
24 heterogeneous mapping on AMD GPU
25 SELECT ?pid ?pname
26 WHERE {
27     ?pid rdf:type hpc:Workflow .
28     ?pid hpc:subject "Heterogeneous Mapping" .
29     ?pid hpc:name ?pname .
30
31     ?pid hpc:targetMachine ?machine_id .
32     ?machine_id hpc:coProcessor ?gpu_id .
33     ?gpu_id hpc:vendor "AMD" .
34
35     ?pid hpc:targetApplication ?app_id .
36     ?app_id hpc:name "NPB" .
37 }
38
39 # Results
40 <http://example.org/workflow/WF000020> OpenCL
41 Heterogeneous Mapping

```

VII. RELATED WORK

Due to their consistency and expressivity, numerous ontologies [35]–[38] have been developed to effectively combine data or information from multiple heterogeneous sources. For example, DBpedia [25], [39] extracts entities and relations from Wikipedia to create an ontology which becomes a natural hub for connecting datasets. Sun et al. [40] developed Geospatial data ontology as the semantic foundation of geospatial data integration and sharing. Their ontology is a framework with three dimensions. The ontological category dimension has general, domain, and application-level ontologies. The content dimension has essential, morphology, and provenance ontologies. Finally, the logic dimension has classes, relations, properties, and instances. Google, Bing and Yahoo rely on

schema.org [21], essentially a public shared ontology, to maintain a list of vocabularies that developers can use to publish structured data about web pages. This ontology also drives features like Google’s Knowledge Graph. Brick [23], [41] is a domain-specific ontology to enable portable applications managing commercial buildings.

Ontologies have been studied to improve machine learning in various aspects, including sentiment classification, answer evaluation, personalized recommendation and so on [42]. OntoDM [43] is an ontology of data mining. Sudathip et al. [44] presented a machine learning ontology covering concepts from learning, applications, techniques, and evaluation. Xu et al. proposed an OWL ontology to make machine-learned functions findable on the web [45]. VIS4ML [46] is an ontology for visual analytics assisted machine learning. It can be used to describe ML workflows. Hwang et al. proposed a task ontology to enable autonomous machine learning modeling [47].

Some studies leveraged ontologies For program analyses and optimizations. Yue et al. developed ontology-based program analysis [48]. Liao et al. explored ontologies to enhance domain-specific language implementations [49]. Pattipati et al. [50] built an extensible framework for ontology-based advanced program analysis.

VIII. CONCLUSION

This paper presents our ongoing efforts to build an ontology to enable FAIR datasets and AI models in the HPC program analysis and optimization domain. We adopt a two-level design to be lightweight and extensible. The resulting HPC Ontology has modeled properties of essential concepts in the domain, including computers, experiments, software, hardware, datasets, AI models, and so on. Preliminary results show that our design is effective to annotate both high-level metadata and low-level details of selected sample data. The ontology is also able to support typical competency questions formed as SPARQL queries.

Future work will include more low-level supplement components. Examples include more comprehensive AI model categories and properties describing internal details of different machine learning models. We will also add a workflow component in order to support workflow synthesizing in HPC. Automated techniques will be explored for generating RDF triples describing more software, hardware, HPC datasets and AI models. The current draft HPC Ontology is available online¹ under a Creative Commons license to collect community input. We welcome community comments and suggestions to enhance the ontology.

ACKNOWLEDGMENT

Prepared by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-826494), the Argonne Leadership Computing Facility under Contract DE-AC02-06CH11357, and the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Program under Award Number DE-SC0021293.

¹<https://hpc-fair.github.io/ontology/>

REFERENCES

- [1] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran, "Benchmarking machine learning methods for performance modeling of scientific applications," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2018, pp. 33–44.
- [2] J. Sun, S. Zhan, G. Sun, and Y. Chen, "Automated performance modeling based on runtime feature detection and machine learning," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. IEEE, 2017, pp. 744–751.
- [3] K. Singh, E. İpek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana, "Predicting parallel application performance via machine learning approaches," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 17, pp. 2219–2235, 2007.
- [4] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso, "A survey of performance modeling and simulation techniques for accelerator-based computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 272–281, 2014.
- [5] T. Islam, A. Ayala, Q. Jensen, and K. Ibrahim, "Toward a programmable analysis and visualization framework for interactive performance analytics," in *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools)*. IEEE, 2019, pp. 70–77.
- [6] J. J. Thiagarajan, R. Anirudh, B. Kaikhura, N. Jain, T. Islam, A. Bhatlele, J.-S. Yeom, and T. Gamblin, "Paddle: Performance analysis using a data-driven learning environment," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 784–793.
- [7] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Diagnosing performance variations in hpc applications using machine learning," in *International Supercomputing Conference*. Springer, 2017, pp. 355–373.
- [8] S. Jayasena, S. Amarasinghe, A. Abeyweera, G. Amarasinghe, H. De Silva, S. Rathnayake, X. Meng, and Y. Liu, "Detection of false sharing using machine learning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–9.
- [9] J. Vetter, "Performance analysis of distributed applications using automatic classification of communication inefficiencies," in *Proceedings of the 14th international conference on Supercomputing*, 2000, pp. 245–254.
- [10] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9428–9433.
- [11] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden, "Doomsday: predicting which node will fail when on supercomputers," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 108–121.
- [12] Y. Zhao, J. Li, C. Liao, and X. Shen, "Bridging the gap between deep learning and sparse matrix format selection," in *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, 2018, pp. 94–108.
- [13] 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.
- [14] N. Sukhija, B. Malone, S. Srivastava, I. Banicescu, and F. M. Ciorba, "A learning-based selection for portfolio scheduling of scientific applications on heterogeneous computing systems," *Parallel and Cloud Computing*, vol. 3, no. 4, pp. 66–81, 2014.
- [15] K. Fagnan, Y. Nashed, G. Perdue, D. Ratner, A. Shankar, and S. Yoo, "Data and models: a framework for advancing ai in science," USDOE Office of Science (SC)(United States), Tech. Rep., 2019.
- [16] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.
- [17] B. Motik, P. F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Rutenber, U. Sattler *et al.*, "Owl 2 web ontology language: Structural specification and functional-style syntax," *W3C recommendation*, vol. 27, no. 65, p. 159, 2009.
- [18] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The knowledge engineering review*, vol. 11, no. 2, pp. 93–136, 1996.
- [19] S. Staab and R. Studer, *Handbook on ontologies*. Springer Science & Business Media, 2013.
- [20] J. Ison, M. Kalaš, I. Jonassen, D. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer, and P. Rice, "Edam: an ontology of bioinformatics operations, types of data and identifiers, topics and formats," *Bioinformatics*, vol. 29, no. 10, pp. 1325–1332, 2013.
- [21] R. V. Guha, D. Brickley, and S. Macbeth, "Schema.org: evolution of structured data on the web," *Communications of the ACM*, vol. 59, no. 2, pp. 44–51, 2016.
- [22] M. A. Jensen, V. Ferretti, R. L. Grossman, and L. M. Staudt, "The nci genomic data commons as an engine for precision medicine," *Blood*, vol. 130, no. 4, pp. 453–459, 2017.
- [23] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal *et al.*, "Brick: Metadata schema for portable smart building applications," *Applied energy*, vol. 226, pp. 1273–1292, 2018.
- [24] S. L. Weibel and T. Koch, "The dublin core metadata initiative," *D-lib magazine*, vol. 6, no. 12, pp. 1082–9873, 2000.
- [25] P. N. Mendes, M. Jakob, and C. Bizer, "Dbpedia: A multilingual cross-domain knowledge base," in *LREC*. Citeseer, 2012, pp. 1813–1817.
- [26] "Top500 List, June 2021," <https://www.top500.org/lists/top500/2021/06/June>, 2021.
- [27] "Sparql for tables: Turn csv into rdf using sparql syntax," <https://tarql.github.io/>, accessed: 2021-06-09.
- [28] C. Liao, P.-H. Lin, J. Asplund, M. Schordan, and I. Karlin, "Dataracebench: a benchmark suite for systematic evaluation of data race detection tools," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–14.
- [29] C. Cummins, Z. Fisches, T. Ben-Nun, T. Hoefler, M. O'Boyle, and H. Leather, "ProGraML: A Graph-based Program Representation for Data Flow Analysis and Compiler Optimizations," in *Thirty-eighth International Conference on Machine Learning (ICML)*, 2021.
- [30] M. Trofin, Y. Qian, E. Brevdo, Z. Lin, K. Choromanski, and D. Li, "Mlgo: a machine learning guided compiler optimizations framework," *arXiv preprint arXiv:2101.04808*, 2021.
- [31] C. Phillips, "Hpc energy research," 2016. [Online]. Available: <https://cscdata.nrel.gov/#/datasets/d332818f-ef57-4189-ba1d-beea291886eb>
- [32] R. Ballester-Ripoll, E. G. Paredes, and R. Pajarola, "Sobol tensor trains for global sensitivity analysis," *Reliability Engineering & System Safety*, vol. 183, pp. 311–322, 2019.
- [33] C. Nugteren and V. Codreanu, "Clune: A generic auto-tuner for opencl kernels," in *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. IEEE, 2015, pp. 195–202.
- [34] R. Shrivastava, "Gpu kernel performance dataset," <https://www.kaggle.com>, 2020.
- [35] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A large ontology from wikipedia and wordnet," *Journal of Web Semantics*, vol. 6, no. 3, pp. 203–217, 2008.
- [36] C. Matuszek, J. Cabral, M. J. Witbrock, and J. DeOliveira, "An introduction to the syntax and content of cyc," in *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*. Citeseer, 2006, pp. 44–49.
- [37] M. A. Sicilia, E. Garcia, S. Sanchez, and E. Rodriguez, "On integrating learning object metadata inside the opencyc knowledge base," in *IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings*. IEEE, 2004, pp. 900–901.
- [38] A. Pease, I. Niles, and J. Li, "The suggested upper merged ontology: A large ontology for the semantic web and its applications," in *Working notes of the AAAI-2002 workshop on ontologies and the semantic web*, vol. 28, 2002.
- [39] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*. Springer, 2007, pp. 722–735.
- [40] K. Sun, Y. Zhu, P. Pan, Z. Hou, D. Wang, W. Li, and J. Song, "Geospatial data ontology: the semantic foundation of geospatial data integration and sharing," *Big Earth Data*, vol. 3, no. 3, pp. 269–296, 2019.
- [41] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal *et al.*, "Brick: Towards a unified metadata schema for buildings," in *Proceedings of the 3rd*

ACM International Conference on Systems for Energy-Efficient Built Environments, 2016, pp. 41–50.

- [42] L. Rachana and S. Shridevi, “A literature survey: semantic technology approach in machine learning,” *Advances in Smart Grid Technology*, pp. 467–477, 2021.
- [43] P. Panov, S. Džeroski, and L. Soldatova, “Ontodm: An ontology of data mining,” in *2008 IEEE International Conference on Data Mining Workshops*. IEEE, 2008, pp. 752–760.
- [44] K. Sudathip and M. Sodanil, “Ontology knowledge-based framework for machine learning concept,” in *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, 2016, pp. 50–53.
- [45] J. Xu, H. Wang, and H. Trimbach, “An owl ontology representation for machine-learned functions using linked data,” in *2016 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 2016, pp. 319–322.
- [46] D. Sacha, M. Kraus, D. A. Keim, and M. Chen, “Vis4ml: An ontology for visual analytics assisted machine learning,” *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 385–395, 2018.
- [47] K. S. Hwang, K. S. Park, S. H. Lee, K. I. Kim, and K. M. Lee, “Autonomous machine learning modeling using a task ontology,” in *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE, 2018, pp. 244–248.
- [48] Y. Zhao, G. Chen, C. Liao, and X. Shen, “Towards ontology-based program analysis,” in *30th European Conference on Object-Oriented Programming (ECOOP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- [49] C. Liao, P.-H. Lin, D. J. Quinlan, Y. Zhao, and X. Shen, “Enhancing domain specific language implementations through ontology,” in *Proceedings of the 5th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, 2015, pp. 1–9.
- [50] D. K. Pattipati, R. Nasre, and S. K. Puligundla, “Opal: An extensible framework for ontology-based program analysis,” *Software: Practice and Experience*, vol. 50, no. 8, pp. 1425–1462, 2020.