A Comprehensive Cloud Architecture for Machine Learning-enabled Research

Joe Stubbs
Texas Advanced Computing Center
Austin, Texas, USA
jstubbs@tacc.utexas.edu

Nathan Freeman
Texas Advanced Computing Center
Austin, Texas, USA
nfreeman@tacc.utexas.edu

Dhanny Indrakusuma
Texas Advanced Computing Center
Austin, Texas, USA
dhannywi@utexas.edu

Christian Garcia
Texas Advanced Computing Center
Austin, Texas, USA
cgarcia@tacc.utexas.edu

François Halbach
Texas Advanced Computing Center
Austin, Texas, USA
francois@tacc.utexas.edu

Cody Hammock
Texas Advanced Computing Center
Austin, Texas, USA
hammock@tacc.utexas.edu

Gilbert Curbelo
Texas Advanced Computing Center
Austin, Texas, USA
gcurbelo@tacc.utexas.edu

Anagha Jamthe
Texas Advanced Computing Center
Austin, Texas, USA
ajamthe@tacc.utexas.edu

Mike Packard
Texas Advanced Computing Center
Austin, Texas, USA
mpackard@tacc.utexas.edu

Alex Fields
Texas Advanced Computing Center
Austin, Texas, USA
afields@tacc.utexas.edu

ABSTRACT

The success of machine learning (ML) algorithms, and deep learning in particular, is having a transformative impact on a wide range of research disciplines, from astronomy, materials science, and climate change to bioinformatics, computational health, and animal ecology. At the same time, these new techniques introduce computational modalities that create challenges for academic computing centers and resource providers that have historically focused on asynchronous, batch-computing paradigms. In particular, there is an emergent need for computing models that enable efficient use of specialized hardware such as graphical processing units (GPUs) in the presence of interactive workloads. In this paper, we present a comprehensive, cloud-based architecture comprised of open-source software layers to better meet the needs of modern ML processes and workloads. This framework, deployed at the Texas Advanced Computing Center and in use by various research teams, provides different interfaces at varying levels of abstraction to support and simplify the tasks of users with different backgrounds and expertise, and to efficiently leverage limited GPU resources for these tasks. We present techniques and implementation details for overcoming challenges related to developing and maintaining such an infrastructure which will be of interest to service providers and infrastructure developers alike.

CCS CONCEPTS

• **System and application engineering for GPUs**;

KEYWORDS

GPUs, Cloud Computing, Machine Learning

ACM Reference Format:
Joe Stubbs, Nathan Freeman, Dhanny Indrakusuma, Christian Garcia, François Halbach, Cody Hammock, Gilbert Curbelo, Anagha Jamthe, Mike Packard, and Alex Fields. 2024. A Comprehensive Cloud Architecture for Machine Learning-enabled Research. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3626203.3670525

1 INTRODUCTION

In the last decade, researchers have applied machine learning (ML) techniques to numerous domains of science and engineering with increasing success. For example, a recent Nature paper highlighted the power of deep-learning algorithms, including AlphaFold2 and RoseTTAFold, to predict the 3D shape of a protein from its genetic sequence [27]. As stated in that paper, a headline from the BBC called it "One of biology's biggest mysteries 'largely solved' by AI". Another Nature paper showed that the use of graph networks can improve the efficiency of discovery of materials by an order of magnitude [34]. The 2022 paper entitled "Tackling Climate Change with Machine Learning" describes a number of ways in which ML can help improve the efficiency of a wide range of systems, including electricity, transportation, buildings and cities, farms and forests, etc., thereby reducing greenhouse gas emissions [40]. Indeed, ML has led to major advances in virtually all fields of science – from Astronomy and the discovery of exoplanets [1] to Zoology and tasks such as species classification [43]. Three primary factors are driving these innovations: the availability of data at scales previously not seen, the increase in computing power, and the emergence of powerful opensource libraries that simplify the development of ML tools and applications.

At the same time, ML techniques require new computing paradigms and workload types. For example, most ML projects require exploratory data analysis where scientists perform data validation, cleaning, standardization, and other pre-processing techniques necessary to achieving good results with ML models. Interactive computing methods, such as Jupyter notebooks, enable researchers to incrementally execute code and immediately analyze the results before deciding how to proceed, a computing model that can simplify exploratory tasks. Moreover, once models have been trained — and researchers increasingly make use of pre-trained models — investigators deploy and query models to make predictions or other inference on new data. Deployed models can require large amounts of main memory and computing power, depending on their size. In particular, ML training and inference, particularly with deep learning, increasingly require specialized hardware, such as graphical processing units (GPUs), to complete in a timely manner. An ML *inference server*, which is a method of working with trained models that is gaining popularity, provides an HTTP interface to the inference task(s) provided by the model, allowing multiple users to interact with the deployed model concurrently over a network.

These new workload types and computing methods present a challenge to existing academic computing centers and service providers that have traditionally specialized in batch computing models. Most HPC clusters utilize a scheduling system, such as Slurm or PBS, where users submit jobs to a queue, and the scheduler starts them some time in the future. The amount of time the user has to wait for their job to start could be seconds, minutes, hours or days, depending on demand and the available resources in the cluster. Moreover, creating a secure network connection (e.g., HTTPS) to the running application imposes additional challenges that usually must be solved on a case-by-case basis by each researcher, as HPC clusters rarely provide facilities for making secure inbound connections to compute nodes. Finally, even if the researcher manages to overcome these challenges, the interactive nature of these workloads typically implies that the underlying hardware resources remain idle for significant periods of time. The relative scarcity of specialized hardware suggests that high resource utilization is an imperative. The National Artificial Intelligence Research Resource (NAIRR) pilot, initiated by the National Science Foundation in response to a US presidential executive order, has recognized the significant gap between the supply and demand of computing power for AI and the critical need to address it. [18].

In this paper, we present a comprehensive architecture comprised of a suite of opensource software to better meet the demands of modern ML workloads and, in particular, to simplify and increase the efficient use of GPUs for research computing. This architecture complements and integrates with traditional batch-scheduled systems deployed at HPC centers. The high-level architecture, depicted in Figure 1, consists of the following primary components: 1) *OpenStack Cloud*: for managing virtual servers with attached GPUs, networking, storage volumes and associated objects (IP addresses, security groups, etc.); 2) *Kubernetes*: for container orchestration, including jobs, and persistent services, with associated objects (pods, services, PVCs); 3) *Scinco JupyterHub*: providing authenticated, per-user Jupyter notebook servers that can be scheduled onto and utilize GPUs with a high degree of configurability; 4) *Tapis Pods API*: providing an authenticated HTTP service capable of deploying persistent networked services and automatically managing DNS, TLS certificates, and related matters; 5) *Tapis Jobs API*: for submitting and managing long-running training jobs to traditional HPC clusters; 6) *Tapis ML Hub*: an HTTP service enabling researchers to discover, persist and manage existing model artifacts and deploy model inference servers; and 7) *Tapis Workflows*: for orchestrating multi-step applications making use of one or more of the previous services/layers. Also depicted are traditional HPC workload and provisioning systems, such as Slurm and the TACC-developed LOSF [33]. Tapis services mentioned in items 4, 5, 6 and 7 are developed as a part of the Tapis framework [42], which is a NSF funded, web-based framework to enable scientific computing in any domain. It is a collaborative project between TACC and University of Hawaii.

The individual software layers provide APIs at different abstraction levels to appropriately address the needs of users with different goals and technical backgrounds. The architecture supports two broad class of users, including ML and infrastructure developers as well as domain researchers wishing to make use of ML in their work.

In the following sections, we discuss the target users and use cases enabled by each layer in the architecture, as well as a number of techniques for overcoming some of the challenges associated with developing, configuring and deploying the associated technology. We provide examples of capabilities that are currently being incorporated into real research projects, and we identify future work that is needed to further improve the experience of developing and utilizing ML in research computing environments.

2 OPENSTACK

2.1 Background and Supported Use Cases

Software development efforts to create, enhance and maintain libraries and tools that simplify the use of Machine Learning algorithms and techniques constitute an increasingly important set of tasks that must be supported. Cloud infrastructure provides a computing model that can more readily support this use case through development environments on virtual machines (VMs) that attach persistent volumes and GPUs as well as networking (e.g., IP addresses) for accessibility. VM-based development environments can be spun up, paused, resumed and shut down, on demand, can easily run for months on end, and can be arbitrarily sized with resources (e.g., memory, cores, disk space) to meet the needs of the develop.

TACC operates a general purpose OpenStack cluster which was chosen to host this effort as it provides flexibility for provisioning dynamic and experimental workloads. A developer using virtual machines can deploy and re-deploy an environment changing the CPU count, memory amount, guest operating system, networking, storage, or other attributes as needed. This OpenStack cluster has been used to provide development environments for more than a dozen undergraduate and graduate students as part of efforts

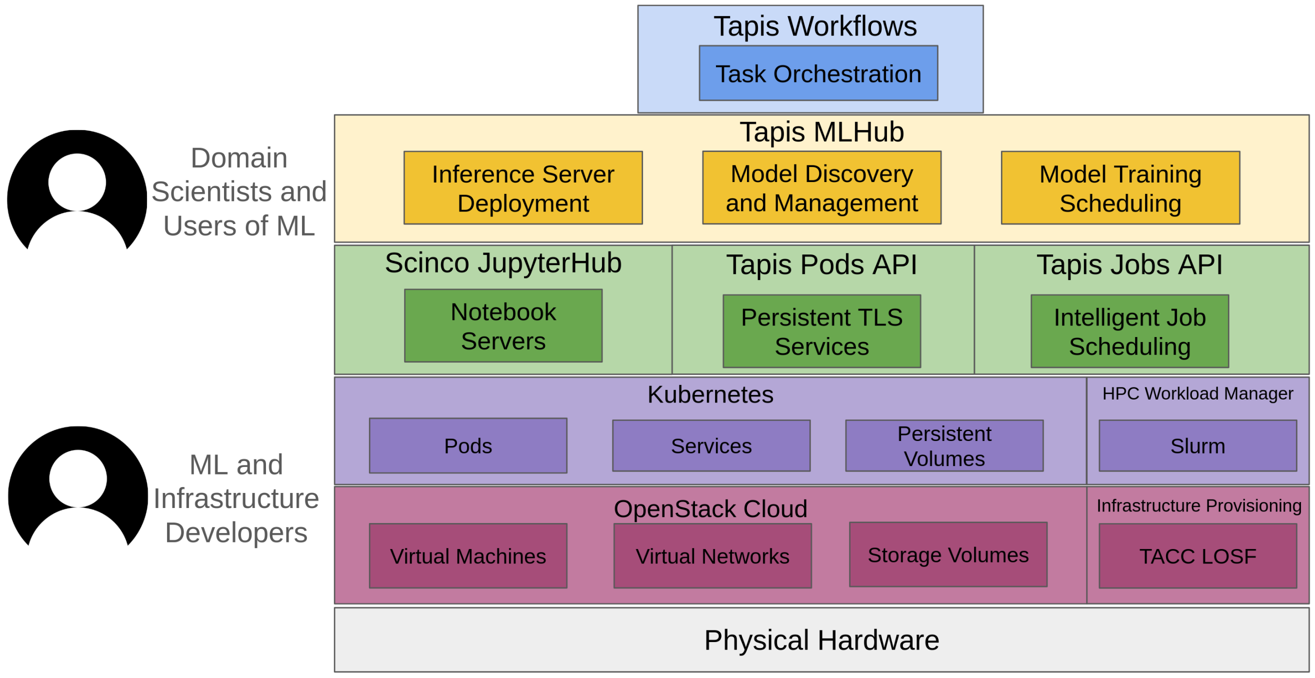

Figure 1: Layered Architecture for Modern ML Research Computing

including the NSF-funded ICICLE AI Institute [7] as well as the SGX3 Fellows program [8].

Our GPU is an NVIDIA V100, mounted in one of several Dell PowerEdge R740 servers running Ubuntu 20.04.2. Each of these servers have two V100 GPUs installed. We chose to make our GPUs available to virtual machines using PCI pass-through, wherein a virtual machine (VM) accesses the device directly via its PCI address on the hardware node hosting the VM. To pass this address to the VM, we must collect some details about the GPU, which will be used to configure a allowable list of PCI devices identified by their PCI bus addresses or by vendor and product identifiers of the device. We chose to use vendor and product details, as it makes no assumptions about which PCI-e address assignments may be used by our GPUs. To obtain the required information we use the `lspci` command:

```
$ lspci -nn | grep NVIDIA
af:00.0 3D controller [0302]: NVIDIA Corporation GV100GL
                    [Tesla V100 PCIe 16GB] [10de:1db4] (rev a1)
d8:00.0 3D controller [0302]: NVIDIA Corporation GV100GL
                    [Tesla V100 PCIe 16GB] [10de:1db4] (rev a1)
```

Here we see the two GPUs present including their vendor ID and product IDs in the bracketed string: `[10de:1db4]`, where the vendor ID is `10de` and the product ID is `1db4`. This information was included into Kolla Ansible by variables to the main configuration file, `globals.yml`.

2.2 OpenStack Deployment at TACC

TACC's OpenStack cluster is managed using Kolla Ansible [13], a component project of OpenStack used to deploy and configure services across the cluster. Kolla Ansible enables an operator to make configuration changes on a single deployment host and quickly distribute those changes to the appropriate hosts. Making configuration changes is done using a combination of variable definitions in the main configuration file [17] and in Jinja2 templates [14] located in the `kolla/config` directory.

In `globals.yml`, the variable `nova_pci_passthrough_allowlist` adds a configuration that permits matching devices to be passed through to a VM. A custom variable `nova_pci_passthrough_alias` allows us to assign a friendly name to matching devices, which is parsed into the deployed configuration by the Jinja2 template engine that is added to `nova.conf` file (see [19] for an example). It is necessary to configure the OpenStack scheduler to use additional metadata regarding GPUs that will be used. The `AggregateInstanceExtraSpecsFilter` and `PciPassthroughFilter` must be added to the list of enabled schedule filters in `kolla/config/nova.conf` as well. When invoked, Kolla Ansible adds the necessary configuration to the OpenStack configuration on the hypervisors and to the scheduling server and restarts the service daemons for the operator.

2.3 Metadata Assignments

When OpenStack launches an VM, a scheduling agent determines which hypervisor is appropriate to host it. We must provide the scheduler with data about which hosts have GPUs installed and a mechanism to use that data when starting a VM. A host aggregate allows a subset of hypervisors to be assigned additional metadata that is used by the system to assign appropriate hardware to a VM. In this case, a host aggregate was added and configured with a metadata parameter `gpu=true`.

When launching a VM, a user selects a *flavor* that defines properties for the VM, such as the number of CPUs and the amount of RAM to assign to the VM. A flavor is created that additionally included parameters `gpu=true` and `pci_passthrough:alias=v100:1`. The `gpu=true` parameter ensures that instance flavors including that value are launched on GPU-enabled nodes that have been assigned to the host aggregate previously mentioned. The `pci_passthrough:alias=v100:1` parameter instructs OpenStack to assign only one of the 2 available GPUs matching the alias `v100` in a node to the VM. This alias is defined in the `nova_pci_passthrough_alias` above. By assigning only one GPU to an instanace, the remaining GPU to be used by another VM on the same hypervisor, permitting more users to have access to the available GPUs.

3 KUBERNETES

Building on the OpenStack layer that provides virtual servers, the next layer in the stack makes use of the open-source Kubernetes platform [11] to support containerized use cases. Kubernetes is a container orchestration platform for deploying and managing containerized applications. Built-in container scheduling, auto-scaling, resource guarantees, and GPU support makes Kubernetes a powerful platform upon which to develop and run highly-available and decoupled scientific applications.

3.1 Kubernetes deployment

TACC runs several internal and external Kubernetes clusters, all deployed using Ansible. Our most recent deployments were completed using Kubespray [16], a tool which uses Ansible to deploy and configure Kubernetes on clouds or baremetal with minimal configuration.

3.2 NVIDIA driver and toolkit installation

In order to enroll a GPU VM as a node into Kubernetes, we configure the node and the containers running in it to be able to communicate with its GPU. The node should be equipped with NVIDIA drivers compatible with the underlying GPU. Since the Openstack configuration uses PCI passthrough to enable VMs to communicate directly with the GPUs, no NVIDIA drivers are required on the hypervisors. If the hypervisor does not use PCI passthrough but has NVIDIA drivers installed on it, matching NVIDIA drivers are required on the VM.

We also install the NVIDIA container toolkit on the GPU node, which contains a container runtime library, and allows for the deployment of containers which are able to make use of the GPUs. We specify the use of the NVIDIA container runtime in the containerd configuration file instead of the standard runc runtime.

3.3 NVIDIA device plugin installation

We then deploy the NVIDIA device plugin daemonset across our Kubernetes cluster. This daemonset runs a pod on each node of the cluster, including non-GPU nodes, and reports the number and status of GPUs on each node.

```
$ kubectl create -f \
https://raw.githubusercontent.com/NVIDIA/k8s-device-
               plugin/v0.14.5/nvidia-device-plugin.yml
```

The NVIDIA device plugin provided by Nvidia is deployed via kubectl, but it can also be deployed via Helm, permitting more feature customization such as allowing for Multi-instance GPU (when supported by the GPU).

3.4 Avoiding and specifying GPU nodes

Once a GPU node is successfully provisioned, we must prevent non-GPU pods from being deployed on it, and make it possible for the node to be specifically selected when needed. Tainting the node allows us to prevent any workloads from being deployed on it:

```
$ kubectl taint nodes <node_name> nvidia.com/gpu=
                                  Exists:NoSchedule
```

Pods with GPU workloads can bypass the taint by including a matching toleration in the .yml file used to deploy them:

```
      tolerations:
      - key: "nvidia.com/gpu"
        operator: "Exists"
        effect: "NoSchedule"
```

To be able to specifically select a node, we label it:

```
$ kubectl label nodes cyclone-cickube-gpu01 gpu=v100
```

Just like with our toleration, we modify our deployment file to include a node selector matching the created label, in order to select a node matching it.

```
      nodeSelector:
        gpu: "v100"
```

3.5 GPU sharing

A current limitation of our configuration is the lack of support for GPU sharing. If a pod is deployed on a node with 1 GPU and reserves this GPU, no more GPU pods can be deployed on the node, even if the pod is not making use of the entire GPU, or running workloads at all times. Multi-Process Service (MPS), an alternative to the CUDA API, is a feature of Nvidia GPUs which addresses this issue, allowing multiple processes to share GPU resources. Its implementation is in progress and will allow for more efficient use of our GPU computing resources.

4 JUPYTERHUB AND PODS API

In this section, we describe two solutions that build on top of Kuberenetes to provide higher-level functionality for users and use cases where direct access to the lower-level infrastucture is not needed.

4.1 JupyterHub

As mentioned previously, Jupyter notebooks provide an interactive computing modality that simplifies some ML tasks, such as exploratory data analysis and validation. The Scinco project [41] offers a customized JupyterHub instance hosted by TACC, allowing users to access notebook servers running on remote machines. Scinco incorporates *multitenancy*, which allows different project teams to configure these notebook servers based on their individual project needs. Tens of thousands of Python notebooks have been authored by thousands of researchers on Scinco as part of research efforts in various domains of science and engineering. Machine learning packages and use cases are an indispensable part of the Scinco offering.

Each user's notebook server is launched as a pod on a Kubernetes [11] cluster. To accomplish this, Scinco builds upon the open-source Kubespawner [3] project with additional features by implementing two *hooks*. One hook retrieves data stored on remote hosts and mounts it in the notebook server for the user, while the second hook retrieves the user's configurations from the Tapis [42] Metadata API, along with other spawner details. Each Scinco project (or "tenant") maintains its own configuration within this API, which can be updated at any time by a tenant administrator.

To add support for GPUs in Scinco notebook servers, the hook updates the Kubespawner with the `extra_pod_config`, `extra_resource_guarantees` and `extra_resource_limits` attributes from the metadata pertaining to the user. An example configuration is provided in the repository for this paper [21]. For more details on these configurations, we refer the reader to the Kubespawner documentation, [15]. Note that Scinco supports creating these configurations on a per-tenant, per-group or per-user basis. Groups allow tenants to create logical partitions of users that should be configured differently than other users. This approach enables tenant administrators to ensure that limited GPUs are available to the teams and individuals that need them most.

4.2 Tapis Pods Service

A number of ML use cases require the deployment of persistent services with transport-level encryption, including inference servers and knowledge graphs. To further simplify the use of Kubernetes for these use cases we have developed and deployed the Tapis Pods service. The Tapis Pods service is a hosted API within the Tapis platform that provides users a feature-rich API to deploy long-lived Kubernetes containers with manageable storage, shareable resources and transport-level encryption at the Texas Advanced Computing Center (TACC). For example, a user can deploy a Neo4j [12] graph database, a custom Flask [10] server, or any containerized workloads using a single API call to the Pods service. The Pods service is primarily used for secure cloud database access, application deployments, and multi-container server deployments.

Following the addition of GPU support in Kubernetes, the Pods service's API had an expansion in the form of GPU requisitions. This feature allows users to run long-lived workloads which make use of GPU performance gains. Complex simulations or web-accessible large language models (LLM) are able to be easily deployed resulting from this new feature. This is made possible by having the Pods service manage node scheduling, resource requests, and health of user containers with the Kubernetes API. This functionality is made available with an easy-to-use interface that allows any level of researcher to quickly get their development on the internet for sharing or full-time deployments.

The Pods service is used to host the ML Hub's Model Hub and Inference Server with appropriate access to GPU resources (see the next section). The ML Hub utilizes the Pods service's storage sharing to cache models from outside repositories such as Hugging Face for utilization at deployment time. Once deployed, the Inference server runs Python code which makes use of the GPU resources for compute.

The Pods service is isolated from most issues due to making use of the Kubernetes layer. Apart from Kubernetes configuration, the Pods service does need to manage GPU allocations and state. Going forward, development will be focused on partitioning GPU resources for more discrete units of compute rather than the current standard of appointing one full GPU to each pod which requests one. This development will allow us to scale further. Additionally, metrics on GPU utilization will be a future point of work. These will allow us to ensure minimum percentages of compute usage, ensuring efficient compute-use independent of workload and providing valuable stats to users using the service.

5 MODEL TRAINING AND THE MACHINE LEARNING HUB API

Unlike other aspects of ML workflows that are naturally interactive, model training can often efficiently make use of traditional HPC systems that utilize batch scheduling because of the long, uninterrupted executions involved. The Tapis Jobs service provides a general facility for executing user-defined workloads asynchronously on remote systems, including traditional HPC clusters. Using the Tapis Jobs service to execute ML model training jobs on batch systems is one approach to integrating with existing HPC resources.

The Machine Learning Hub (ML Hub) [31] is a set of REST APIs developed as part of the Tapis Framework specifically for ML workflows. It is designed to simplify the researcher's machine learning workflows by abstracting the complexities of fetching and running inference on open-source, pre-trained models hosted on Hugging Face [2]. By leveraging over 500,000 models available on Hugging Face, ML Hub users can experiment with state-of-the-art AI applications suitable for a specific task, without the technical complexities of building models from scratch.

The ML hub consists of three main services - the Models and Datasets Hub, the Inference Server, and the Training Engine. We have containerized the Model and Datasets Hub and Inference Server using Docker [35] and deployed it to the Pods Service [25]. We are collaborating with Computational Learning in the Environment (ICICLE) AI Institute to develop a federated model repository that will list the models created by researchers on the ML Commons platform in the ML Hub platform.

The Training Engine and ML Hub's integration with Tapis UI [32] is under active development. Projects such as ClaimBuster [30] are actively collaborating with the Machine Learning Hub for model hosting and ongoing research efforts. For the training engine, we are designing it to submit the model training requests to the HPC cluster using the Tapis Jobs API, which in turn, is submitted to the slurm HPC workload manager. Finally, an infrastructure is provisioned by the TACC LOSF to run the training jobs and update the user when the job is done.

5.1 Model and Datasets Hub

The Models and Datasets Hub (Hub) provides a central access point for users to discover pre-trained ML models and Datasets. The API utilizes Flask, a lightweight web framework developed in Python. Flask is used as it seamlessly integrates with Hugging Face Hub's Python Library [6] to fetch information and serve download links to the user. The Hub consists of the public endpoints for authenticated Tapis users as shown in table 1 below, and the endpoints starts with /v3/ml-hub.

Table 1: Model Hub endpoints

Route	Method	Usage
/models	GET	Provides an overview of the top 100 models
/models/<path:model_id>	GET	Returns detailed information on the requested model id
/models/<path:model_id>/inference	GET	Allows user to check availability of inference server for the specified model id
/models/authors/<author_id>	GET	Filters available models by author
/models/search/<path:query>	GET	Filters available models by user query
/models/tasks/<task_type>	GET	Filters available models by task type
/models/trained_datasets/<path:dataset>	GET	Filters available models by dataset(s) it was trained on
/models/libraries/<library_name>	GET	Filters available models by the foundational libraries the models were originally trained from
/models/languages/<language_name>	GET	Filters available models by language
/download_model/<path:model_id>	GET	Retrieves download links for requested model id
/model_card/<path:model_id>	GET	Fetch model's card data
/datasets	GET	Provides an overview of the top 100 datasets
/datasets/<path:dataset_id>	GET	Returns detailed information on the requested dataset id
/datasets/authors/<author_id>	GET	Filters available datasets by author
/datasets/search/<path:query>	GET	Filters available datasets by user query
/datasets/tasks/<task_type>	GET	Filters available datasets by task type
/datasets/languages/<language_name>	GET	Filters available datasets by language
/datasets/size_categories/<category>	GET	Filters available datasets by its size category
/download_dataset/<path:dataset_id>	GET	Retrieves download links for requested dataset id
/dataset_card/<path:dataset_id>	GET	Fetch dataset's card data

Table 2: Inference Server endpoints

Route	Method	Usage
/inference	GET	Returns available models for inference
/inference/<path:model_id>	POST	Returns inference results from a model as a JSON Response

5.2 Inference Server

We developed the Inference server to allow users to experiment with various pre-trained machine learning and deep learning models. By sending requests to a specific model's server and evaluating the results, users could discover the right models for a specific task in their domain expertise.

The server is implemented using FastAPI, an open-source, high-performance Python web framework [5]. FastAPI is used as it has built-in data validation for incoming requests and most of the popular deep learning libraries such as Transformers [4] and PyTorch [36] are developed in Python. Two endpoints are available for the Inference Server as shown in table 2, and the inference endpoints starts with /v3/ml-hub.

For our initial development of the inference server, we focused on the text-to-text generation task. It currently supports inference for three fine-tuned models based on the Text-To-Text Transfer Transformer (T5) encoder-decoder architecture [38]:

- **Voicelab's vlT5-base-keywords**: A fine-tuned T5-base model that generates keywords from short texts [37].
- **Google's FLAN-T5 large**: An improved T5 model that has been pre-trained on various NLP tasks [26].
- **Grammarly's CoEdIT-Large**: A fine-tuned FLAN-T5 large model that edits texts based on user's instructions [39].

5.3 Training Engine

The Training Engine, currently under active development, will provide first-class support for executing training and fine-tuning jobs on HPC and cloud resources via HTTP requests. The Model Hub Training Engine wraps the lower level Tapis Jobs service and adds additional capabilities for ML models, including: 1) a simplified fine-tuning process for existing models available via the Hub; 2) automatic checkpointing and job re-submission, to allow for training runs that exceed the maximum allowable run time of an HPC cluster; and 3) automatic persistence of learned model weights and integration with the inference server. s

6 CONFIGURABLE MACHINE LEARNING WORKFLOWS WITH TAPIS WORKFLOWS

In this section, we present an inference pipeline developed in Tapis Workflows [28] which enables users to choose which type of accelerator that their inference server will use. This pipeline is a reusable template and can be found in the Tapis Workflows Task Template Github repository [22].

6.1 Overview of Tapis Workflows

Tapis Workflows is an API and Workflow Engine in a Tapis deployment that are designed to facilitate the construction and execution of complex, distributed, and reproducible workflows. Tapis Workflows powers CI/CD pipelines for ML-enhanced HPC codes for the Tuitus project [9] as well as the up-coming ETL (Extract, Transform, Load) pipelines for processing satellite image data in the NASA JPL SPHEREx project [20].

Tapis Workflows comprises two main services. The first is the API which is responsible for validating, persisting, and submitting **pipelines** (**workflows**). The second is the Workflow Engine which is responsible for coordinating the execution of individual **tasks** which comprise a workflow, as well as preparing the infrastructural components upon which that work will be performed. In the section below, we will briefly cover the terminology employed in Tapis Workflows followed by a description of the pipeline.

- **Tasks**: Instruction-sets that describe a discrete unit of work to be performed in a workflow, as well as the dependencies on and relationships to other units of work
- **Pipelines**: Collections of tasks modeled as a Directed Acyclic Graph (DAG). In a addition to tasks, a pipeline describes an execution profile to control how and if a pipeline will run at submission time, an environment which serves as a static data source for tasks, and a parameter set that - in addition to being a variable source of data for tasks - describes the interface that needs to be satisfied in order for a pipeline to run

6.2 Inference Pipeline Composition

This pipeline is made up of three sequential tasks:

(1) The first is the data ingestion task. This task takes a URL provided to the pipeline via an argument in the pipeline submission request and performs an HTTP request to fetch the HTML contents of the web page. The text in the HTML is then parsed out and returned as the output of the task.
(2) The second task takes the text from the first task and processes it so that keyword analysis can be run on it. This processed text is then returned as output to be used by the inference task.
(3) The inference task takes the processed text and submits it to the Tapis ML Hub Inference API for keyword analysis. Users can provide a flag via pipeline arguments to indicate to this task which accelerators to use for inference. Once inference is performed, the results of that inference are returned as output and made available to the user.

This pipeline must be submitted to the Tapis Workflows API with two arguments (provided in the request body). The first is "URL", which is the web URL that contains the text upon which the user wants to run keyword analysis. The second is "ACCELERATOR". This argument controls whether inference will be run on GPUs or CPUs.

7 SCALABILITY

Scaling the described architecture as implemented is currently limited primarily by the resources available, as GPU accelerators are relatively scarce. Generally speaking, the technologies in use have demonstrated the ability to scale well. At TACC the Jetstream and Chameleon clusters operate OpenStack installations of several hundred nodes in a combination of virtualized infrastructure and bare metal as a service since 2015. Furthermore, CERN is well-known for operating OpenStack at the scale of over 9000 servers, providing more than 300,000 cores [24]. Several of the Tapis services mentioned have also undergone rigorous scalability studies. For example, the Tapis Jobs service was shown to scale to 10,00 concurrent jobs [23], while the Tapis functions service was shown to correctly scale to 100 JetStream m1 medium instances [29].

8 CONCLUSION

In this paper we presented a cloud-based architecture and integrated, open-source software stack, that provides capabilities that simplify the use of modern machine learning in research computing and increases the utilization of GPU hardware currently in high-demand. This software suite is deployed at the Texas Advanced Computing Center and utilized by a number of research teams. We provide techniques for overcoming challenges related to developing, configuring and administering the associated technologies.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Office of Advanced CyberInfrastructure, the Tapis Framework:[1931439 and 1931575], the Tuitus project [2229702], the ICICLE AI Insitute [2112606], and SGX3 [2231406].

REFERENCES

[1] 2017. *Artificial Intelligence, NASA Data Used to Discover Eighth Planet Circling Distant Star.* https://www.jpl.nasa.gov/news/artificial-intelligence-nasa-data-used-to-discover-eighth-planet-circling-distant-star Last access: 2024-02-06.
[2] 2017. *Hugging Face.* https://huggingface.co Last access: 2024-03-08.
[3] 2017. *kubespawner.* https://github.com/jupyterhub/kubespawner Last access: 2024-02-06.
[4] 2017. *Transformers.* https://github.com/huggingface/transformers Last access: 2024-03-07.
[5] 2018. *FastAPI.* https://fastapi.tiangolo.com Last access: 2024-03-08.
[6] 2020. *Hugging Face Hub.* https://github.com/huggingface/huggingface_hub Last access: 2024-03-08.
[7] 2021. *ICICLE AI InstituteL Intelligent CI with Computational Learning in the Environment.* https://icicle.osu.edu/ Last access: 2024-03-07.
[8] 2021. *SGX3 Fellows Journey: From Research to Software Engineering the AI-Driven HPC Resource Prediction and PEARC23 Experiences.* https://sciencegateways.org/networking-community/blogs/b/s/sgx3-fellowship-journey-swathi-vallabhajosyula Last access: 2024-03-07.
[9] 2022. *Tuitus: Award Abstract.* https://www.nsf.gov/awardsearch/showAward?AWD_ID=2229702&HistoricalAwards=false Last access: 2023-02-13.
[10] 2023. *Flask.* flask.palletsprojects.com/en/2.3.x/ Last access: 2023-4-19.
[11] 2023. *Kubernetes.* kubernetes.io Last access: 2023-4-19.
[12] 2023. *Neo4j.* neo4j.com/ Last access: 2023-4-19.
[13] 2024. *Kolla Ansible's Documentation.* https://docs.openstack.org/kolla-ansible/latest/ Last access: 2024-03-07.
[14] 2024. *Kolla Jinja Template Example.* https://palletsprojects.com/p/jinja/ Last access: 2024-03-07.
[15] 2024. *Kubespawner documentation.* https://jupyterhub-kubespawner.readthedocs.io/en/latest/spawner.html Last access: 2024-03-07.
[16] 2024. *Kubespray: deploy a production-ready Kubernetes cluster.* https://github.com/kubernetes-sigs/kubespray Last access: 2024-03-07.
[17] 2024. *Main Kolla Ansible Configuration File Example.* https://github.com/tapis-project/tapis-gpu-paper/blob/main/openstack/globals.yml Last access: 2024-03-07.
[18] 2024. *National Artificial Intelligence Research Resource Pilot: NSF.* https://new.nsf.gov/focus-areas/artificial-intelligence/nairr Last access: 2024-02-06.
[19] 2024. *nova.conf Configuration File Example.* https://github.com/tapis-project/tapis-gpu-paper/blob/main/openstack/nova.conf Last access: 2024-03-07.

[20] 2024. *SPHEREx*. https://www.jpl.nasa.gov/missions/spherex Last access: 2024-3-07.

[21] 2024. *tapis-gpu-paper/jupyterhub*. https://github.com/tapis-project/tapis-gpu-paper/blob/main/jupyterhub/configs.json Last access: 2024-03-07.

[22] 2024. *Tapis Workflows Task Templates*. https://github.com/tapis-project/tapis-workflows-task-templates Last access: 2024-3-07.

[23] Richard Cardone, Joe Stubbs, Steve Black, Christian Garcia, Anagha Jamthe, Mike Packard, and Smruti Padhy. 2022. A Design Pattern for Recoverable Job Management. In *Practice and Experience in Advanced Research Computing (PEARC '22)*. Association for Computing Machinery, New York, NY, USA, Article 29, 4 pages. https://doi.org/10.1145/3491418.3535136

[24] Castro León, José. 2019. Advanced features of the CERN OpenStack Cloud. *EPJ Web Conf.* 214 (2019), 07026. https://doi.org/10.1051/epjconf/201921407026

[25] Richard Cardone Christian Garcia, Joe Stubbs and Nathan Freeman. 2023. Tapis Pods Service Exploration and Initial Performance Analysis. In *Science Gateways 2023 Annual Conference*. Zenodo. https://doi.org/10.5281/zenodo.10034631

[26] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. https://doi.org/10.48550/ARXIV.2210.11416

[27] Michael Eisenstein. 2021. Artificial intelligence powers protein-folding predictions. *Nature* 599 (2021), 706–708. https://doi.org/10.1038/d41586-021-03499-y

[28] Freeman et al. 2023. *Detailed Functional Overview of an API and Workflow Engine for Scientific Researach Computing*. https://dl.acm.org/doi/fullHtml/10.1145/3569951.3593609 Last access: 2023-07-20.

[29] Christian Garcia et al. 2020. The Abaco Platform: A Performance and Scalability Study on the Jetstream Cloud. In *The 16th International Conference on Grid, Cloud, and Cluster Computing (GCC'20)*. Springer Nature.

[30] Naeemul Hassan, Fatma Arslan, Chengkai Li, and Mark Tremayne. 2017. Toward Automated Fact-Checking: Detecting Check-worthy Factual Claims by ClaimBuster. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1803–1812.

[31] Dhanny Indrakusuma, Nathan Freeman, and Joe Stubbs. 2023. Machine Learning Hub for Tapis Poster Presentation. In *Science Gateways 2023 Annual Conference*. Zenodo. https://doi.org/10.5281/zenodo.10055681

[32] Keith Strmiska Joe Stubbs Sean Cleaveland Joon Yee Chuah, Jake Rosenberg and Jared McLean. 2023. Tapis UI - A Rapid Deployment Serverless Science Gateway Built on the Tapis API. Zenodo, Science Gateways 2021. https://doi.org/10.5281/zenodo.5570569

[33] Robert McLay, Karl W. Schulz, William L. Barth, and Tommy Minyard. 2011. Best practices for the deployment and management of production HPC clusters. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11. https://doi.org/10.1145/2063348.2063360

[34] A. Merchant, S. Batzner, and S.S. Schoenholz. 2023. Scaling deep learning for materials discovery. *Nature* 624 (2023), 80–85. https://doi.org/10.1038/s41586-023-06735-9

[35] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* 2014, 239 (2014), 2.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[37] Adam Wawrzyński Filip Żarnecki Bartłomiej Nitoń Piotr Pęzik, Agnieszka Mikołajczyk and Maciej Ogrodniczuk. 2022. Transferable Keyword Extraction and Generation with Text-to-Text Language Models. In *Lecture Notes in Computer Science, vol 14074*, Bob Johnson (Ed.). Computational Science – ICCS 2023. ICCS 2023, Springer, Cham, Los Angeles, CA, 398–405. https://doi.org/10.1007/978-3-031-36021-3_42 This is a sample entry for a paper in conference proceedings.

[38] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[39] Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. 2023. CoEdIT: Text Editing by Task-Specific Instruction Tuning. (2023). arXiv:cs.CL/2305.09857

[40] David Rolnick, Priya L. Donti, Lynn H. Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, Alexandra Sasha Luccioni, Tegan Maharaj, Evan D. Sherwin, S. Karthik Mukkavilli, Konrad P. Kording, Carla P. Gomes, Andrew Y. Ng, Demis Hassabis, John C. Platt, Felix Creutzig, Jennifer Chayes, and Yoshua Bengio. 2022. Tackling Climate Change with Machine Learning. *ACM Comput. Surv.* 55, 2, Article 42 (feb 2022), 96 pages. https://doi.org/10.1145/3485128

[41] Joe Stubbs et al. 2020. Integrating Jupyter into Research Computing Ecosystems. Proceedings of the Practice and Experience on Advanced Research Computing, PEARC 2020.

[42] Joe Stubbs, Richard Cardone, Mike Packard, Anagha Jamthe, Smruti Padhy, Steve Terry, Julia Looney, Joseph Meiring, Steve Black, Maytal Dahan, Sean Cleveland, and Gwen Jacobs. 2021. Tapis: An API Platform for Reproducible, Distributed Computational Research. In *Advances in Information and Communication*, Kohei Arai (Ed.). Springer International Publishing, Cham, 878–900.

[43] Cheng-Hong Yang, Kuo-Chuan Wu, Li-Yeh Chuang, and Hsueh-Wei Chang. 2022. DeepBarcoding: Deep Learning for Species Classification Using DNA Barcoding. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 19, 4 (2022), 2158–2165. https://doi.org/10.1109/TCBB.2021.3056570