Framework for tracking metadata, lineage and model provenance in hybrid simulation-AI HPC exascale workflows

Martin Foltin
martin.foltin@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Fort Collins, CO, USA

Andrew Shao
andrew.shao@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Victoria, BC, Canada

Rishabh Sharma
rishabh.sharma@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Milpitas, CA, USA

Shreyas Kulkarni
shreyas.kulkarni@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Fort Collins, CO, USA

Annmary Justine Koomthanam
annmary.roy@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Fort Collins, CO, USA

Aalap Tripathy
aalap.tripathy@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Spring, TX, USA

Wenqian Dong
wenqian.dong@oregonstate.edu
EECS Department, Oregon State University
Corvallis, OR, USA

Cong Xu
cong.xu@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Singapore, Singapore

Suparna Bhattacharya
suparna.bhattacharya@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Bangalore, Karnataka, India

Brian Sammuli
sammuli@fusion.gat.com
Advanced Computing Center, General Atomics
San Diego, CA, USA

Paolo Faraboschi
paolo.faraboschi@hpe.com
AI Research Lab, Hewlett Packard Labs, Hewlett Packard Enterprise
Milpitas, CA, USA

Abstract

The integration of AI in HPC workflows can have a profound impact on HPC scale and usability, for example, by accelerating simulations with surrogate models or intelligently steering simulations based on previous results. New workflows are explored in which AI models are iteratively improved by continual learning to better reflect input data distributions and avoid outliers and drifts. Tracking of model provenance in these workflows is important to understand how new data affect model performance, allow unwinding to previous iterations, and provide a better understanding of conditions where AI models perform well for future reuse. This is more challenging in hybrid HPC-AI workflows because the lineage and provenance must be tracked across multiple software components at different levels of scale. In this work, we extend HPE Common Metadata Framework to hybrid simulation – AI workflows. We demonstrate benefits of CMF tracking across simulation, AI training, and AI inference along HPE SmartSim system on a simple computational fluid dynamics problem with Eddy Kinetic Energy parameterized by AI. We track out-of-distribution data for continuous learning and employ adaptive switching between different models to improve the quality of results. We are working with fusion energy and materials science communities to enhance coupled simulation – AI workflows in a similar fashion in those domains.

CCS Concepts

• **Computing methodologies** → **Modeling and simulation**; **Online learning settings**; • **Applied computing** → *Earth and atmospheric sciences*; *Physics*.

Keywords

turbulence modelling, active-learning, metadata, data provenance, workflow monitoring

ACM Reference Format:
Martin Foltin, Andrew Shao, Rishabh Sharma, Shreyas Kulkarni, Annmary Justine Koomthanam, Aalap Tripathy, Wenqian Dong, Cong Xu, Suparna Bhattacharya, Brian Sammuli, and Paolo Faraboschi. 2025. Framework for tracking metadata, lineage and model provenance in hybrid simulation-AI HPC exascale workflows. In *Proceedings of Cray User Group (CUG).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

CUG, New York City, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

It has been demonstrated in many use cases that AI can significantly improve scientific simulations by the use of surrogate AI models for high quality physics parameterizations (AI in the loop) or by steering simulations based on intelligent analysis of previous results (AI on the loop and over the loop) [3]. AI models employed in these

cases often evolve over time to better reflect the input data distributions and avoid outliers and drifts. In practice, the performance of the AI model can be estimated by calculating a figure of merit within the simulation, monitoring simulation convergence (e.g., differential equation residuals), or by having the models estimate their own uncertainty through ensembles and other techniques. In this process, the models can be gradually improved in a continuous learning loop [2]. Input data for re-training need to be selected carefully to prevent catastrophic forgetting [14] and model bias.

Tracking model provenance allows a posteriori analysis of how new datasets improved model performance, supports unwinding to previous versions of the model when performance degrades, and provides better understanding of conditions (e.g., input data distributions) where these models performed well in previous use cases for future reuse. The provenance tracking in hybrid simulation-AI workflows is more challenging than in pure AI workflows because they involve multiple software components producing metadata at different volumes and velocities – the workflow manager, simulation code, an orchestrator serving AI model inference, and an AI model retraining system. Different contributions to AI model provenance need to be tracked across these components.

In this work, we demonstrate a hybrid simulation-AI workflow enabled by extensions to HPE's Common Metadata Framework (CMF) library and the SmartSim AI/HPC workflow manager [6, 12]. CMF [4] is an open source infrastructure previously developed to track metadata, lineage and model provenance across federated AI workflows.

The goal of this workflow is to retrain AI models (distinguished by their architectures) used for turbulence parameterizations when new, out-of-distribution data is generated by a high-resolution simulation. The best-in-breed trained model is then embedded in a coarse-resolution simulation, called every timestep to parameterize the effects of turbulence. The motivation of this application is to demonstrate how the tracking of model provenance and metadata from both workflow and simulation can help gain insight into these loosely-coupled workflows. We note that the primary purpose is to provide a concrete example with components that are actually used in scientific studies, but make no claim on its domain science results.

The novelty of this work is in the:

- development of a framework enabling metadata and model provenance tracking across different components of hybrid simulation-AI workflow that impose different throughput, latency, and programming environment requirements, including the workflow manager, the simulation code, and the AI model inference orchestrator
- demonstration of benefits of this framework in enabling continual learning of turbulence parameterization model

Section 1 first highlights the features of two software libraries CMF and SmartSim. Then, Section 2 describes a continual learning workflow from a conceptual level before introducing an implementation of this workflow specifically for training a neural network model of turbulence in Section 3. We then discuss the benefits of developing metadata-driven workflows at scale in Section 4 before summarizing the main results and describing future work in Section 5.

1 Software used in this framework

This framework is supported by two open-source projects, the Common Metadata Framework (CMF) and SmartSim both developed by HPE. This section highlights the features of each library that are used for the continual learning example described later in Section 2.

1.1 Common Metadata Framework

Figure 1 shows an AI in the loop surrogate modeling workflow with the CMF tracking library included in different workflow components. Notably, the library can log metadata and data at multiple levels of granularity depending on the needs of the workflow. All metadata and execution information of the workflow are captured in a SQLite database.

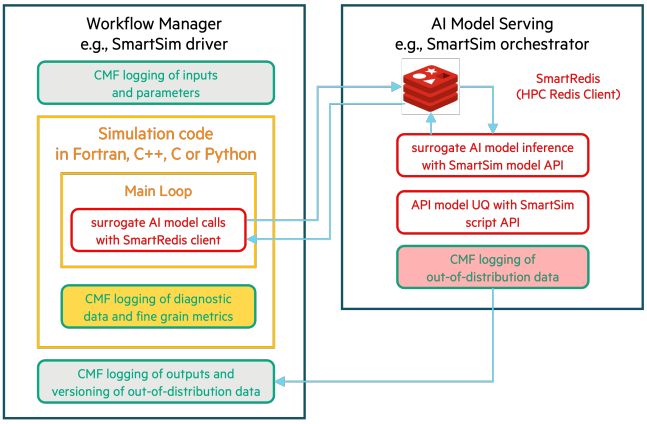

Figure 1: Hybrid simulation/AI workflow showcasing specific integrations of the Common Metadata Framework (CMF) library with the SmartSim workflow manager. Yellow and pink boxes are new lightweight integrations into the simulation and the AI orchestrator, respectively. The main CMF library (gray box) collects and versions outputs from these integrations to include as part of the experiment lineage.

CMF defines three levels of workflow abstractions that map to corresponding portions of a workflow managed by an arbitrary workflow manager, ordered from highest to lowest:

- Pipeline: the overarching workflow itself
- Context: a particular stage of the workflow
- Execution: a specific invocation of a stage.

These are created by instantiating the CMF object and calling `create_context` and `create_execution` methods in corresponding parts of user workflow. Notably, metadata can be attached to each of these as well, for example execution time, iteration number, etc.

Metadata and data can be attached to every unique combination of Pipeline, Context, and Execution. `log_dataset` and `log_model` record both hashes and the actual datasets generated by the applications and machine-learning models, respectively. Hashes of the artifacts allow for quick detection of whether these datasets have changed since the last execution. This is particularly important for applications which must guarantee data lineage throughout the

entirety of the workflow. The hashes are used to reference different versions of an artifact in a Git-like version control system employed by the CMF backend. The lineage is built by connecting the corresponding output and input artifact references from successive workflow stages. `log_execution_metric` captures metrics associated with the execution of a stage, e.g. the final validation loss for a training script. `log_metric` logs user-defined metrics at the finest level of granularity, often embedded within the inner loops of the application (e.g. per epoch training loss, simulation convergence by iteration, etc.).

For this work, we created Fortran and C-based interfaces to CMF to allow the simulation to post fine-grained metrics during its internal timestepping. This integration relies on well-established Python-C interoperability with Fortran bindings to interact with the C-layer. Broadly speaking, the primary `Cmf` object which has both initialization and the `log_metric` method is first initialized with a given pipeline name, context, and execution. When initializing this library, the user calls the C-layer which will create the object in memory as a singleton. Subsequent methods written in C are simply interfaces to this Python-object. All the supporting libraries and the Python interpreter itself need to be loaded for every rank that attempts to initialize the client which results in a memory footprint of about 160MB. Because these find-grained metrics are intended to track the overall progress of the simulation, we only instantiate the CMF client on the primary rank.

1.2 SmartSim

The SmartSim library has two purposes in this workflow: 1) Workflow management: referring to the configuration and launching applications 2) In-memory data movement: deploying an in-memory database and providing communication clients that enable compiled applications to communicate with the database. The overall architecture has been previously described in the literature [11, 12], so here we only briefly highlight the key components germane to this workflow.

The workflow management component allows users to define and configure applications, specify resources, interact with the workflow manager for resource allocations, launch applications, and define the overall workflow via a python-based driver script. This abstracts many of the details of launching applications (e.g. SLURM or PBS) from the user in addition to tracking workflow component status and execution states. In this study, the workflow management component configures applications, sets up the run directory, and launches the application by interacting with the workload manager.. The data sampler and model trainers are also executed at the appropriate times. This driver script was instrumented with CMF to track workflow-level metrics and inputs/output datasets to the simulation components.

SmartSim also deploys a Redis database that can be used as an in-memory datastore accessible to every workflow component. The SmartRedis sister library provides clients that can interact with the database, to send/receive data from simulations and applications. The datastore here is used in two ways: 1) as a way to cache information streamed from the simulation until it is consumed by a separate application and 2) an inference service that the simulation can call to enable online neural network prediction.

Two keys features of the SmartRedis library are exercised here to help move data from the data-parallel simulation to other parts of the workflow: datasets and dataset lists. SmartRedis Datasets are containers for metadata and array/tensor-like information, reducing the number of interactions with the database. Dataset lists allow users to append individual datasets into a retrievable object. Notably, for long lists, the SmartRedis client can pull multiple datasets in parallel. These features are used to transfer data from the simulation to the data sampler and the data sampler to the model trainer.

Tracking data important for model retraining poses a special challenge for CMF. Unlike aggregated simulation diagnostics that are typically logged from rank 0, the out-of-distribution data can occur in any rank. For workflows managed by SmartSim, we take advantage of SmartSim client-server architecture where simulation clients pass data to SmartRedis server(s) running in AI orchestrator(s) to perform inference. We track out-of-distribution data in CMF as output artifacts from the data sampler. The tracked data can be saved in a file or pushed to SmartRedis list, whereby a separate client drains the list to a file, moving the file access away from latency critical inference work. After the simulation, CMF library integrated to the workflow manager versions this data for the given simulation experiment, and ensures inclusion in the workflow lineage.

2 A continual learning workflow driven by metadata

Continual learning is a machine-learning framework where models are continually updated as new data become available. This is crucial in computational simulations where the modelled state cannot be constrained to fall within a space spanned by a previously curated training dataset. Particularly for neural network surrogates of sub grid-scale (SGS) parameterizations, these models may be forced to extrapolate and create predictions which are unphysical. Detecting these situations and retraining the neural network to accommodate these new portions of the input space is thus crucial.

The architecture of a neural network often determines how efficiently it can adapt to new data, however determining which of these architectures are best suited for a specific problem a priori is challenging. Training ensembles of AI models allows for a number of architectures to be evaluated at once. In a continual learning context, the best of breed architecture may change as more data is ingested.

This study describes an archetypal continual learning workflow applied to the problem of training a neural network for SGS physics. Like many of the emerging hybrid AI/simulation workflows, it has loops and branching logic where the AI and simulation components sometimes interact or execute independently of each other. Four types of entities are considered in this workflow with four separate types of entities

- Data artifacts: input and output datasets of the workflow
- Dynamic workflow data: ephemeral data which are produced but not retained as part of the workflow
- AI artifact: a trained AI model that is continually updated as part of the workflow

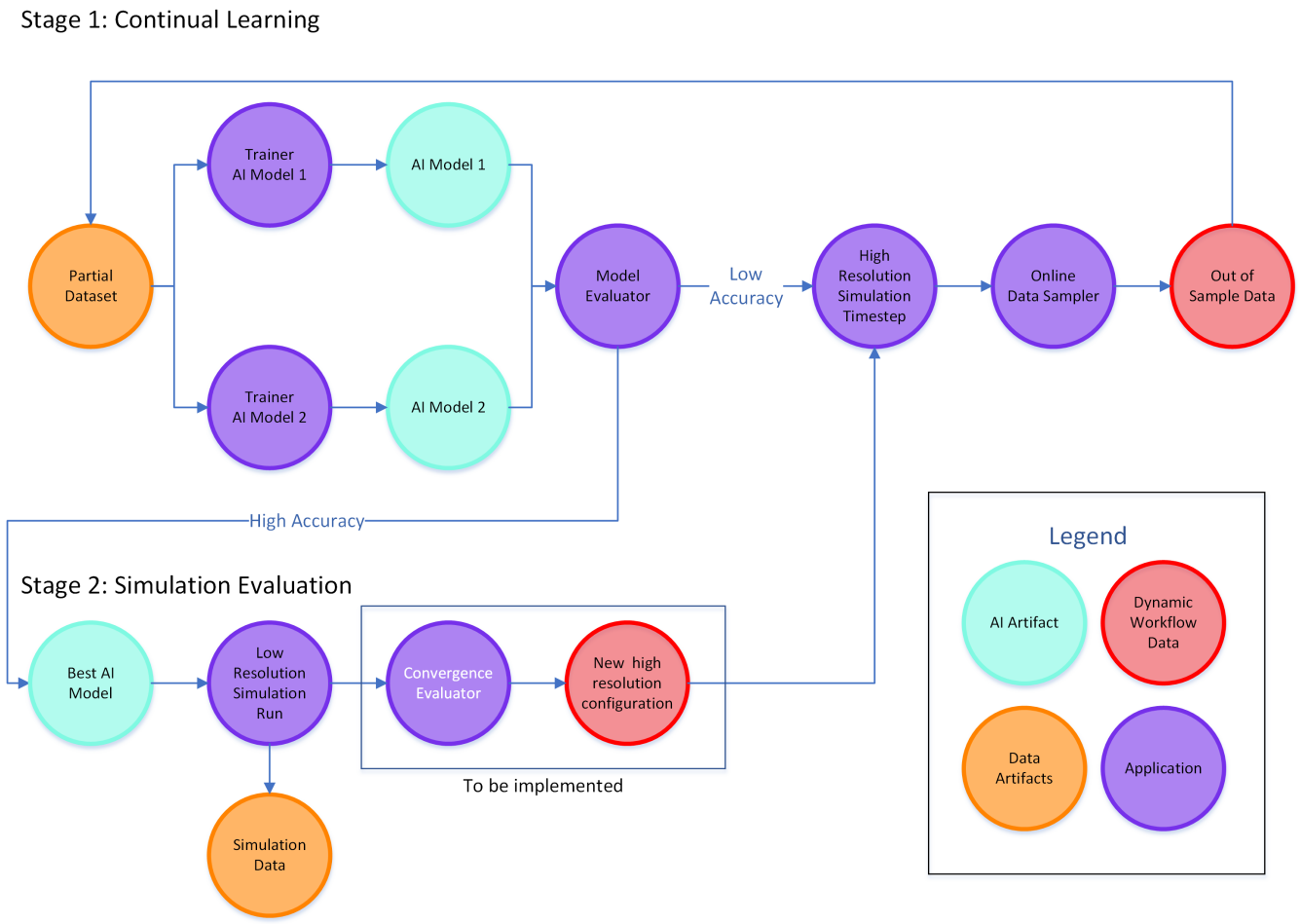

Figure 2: Overview of the continual learning workflow implemented in this paper. The first stage trains a neural network to learn the turbulence parameterization and then queries the execution metrics of the trainer to determine whether either model is ready to be deployed. If not, a high-resolution simulation is integrated to generate new data. If yes, the second stage of the workflow begins where the AI model is deployed into a low-resolution simulation. Artifacts (both data and trained neural networks) are tracked via CMF.

- Application: Executable components of the workflow which includes various training/processing scripts and the simulation components.

These entities are composed into a workflow with two stages (Figure 2).

Stage 1 represents the continual learning portion beginning with a pre-selected dataset that encompasses only a portion of the entire input space. This dataset is used to train an ensemble of neural networks which differ in their architectures, resulting in preliminarily trained models. These are then sent to an evaluator that determines whether new data need to be generated or if the models are ready to be deployed. In the case where new data need to be generated, the high-fidelity simulation is executed to generate more data. As the simulation executes, the output is sampled to detect which portions lie in the out-of-distribution space. These are then added to the original dataset to retrain the ensemble of neural networks which are then sent to the evaluator again.

Stage 2 is the deployment stage of the workflow and begins when the model evaluator determines that at least one of neural networks has achieved a desired level of accuracy. The best of breed is selected and deployed inside of the lower fidelity simulation as part of the timestepping. Generally, this is where the workflow would end. In cases where the improvement is measurable only through the emergent behavior of the simulation, a new configuration of the high fidelity simulation may be needed to explore a new portion of the simulation space. Stage 2 would then return to Stage 1 and result in an outer loop of the workflow.

3 Applying continual learning to fluid dynamics

This section applies the conceptual framework to an increasingly common simulation/AI task in computational fluid dynamics (CFD)- the training of AI surrogate models of turbulence from high-resolution simulations for deployment in low-resolution simulations. Sections 3.1 and 3.2 provide a brief overview of the theory behind turbulence modelling and the specific CFD simulation used in this study. Section 3.3 discusse the implementation and results of the workflow.

3.1 Modelling Turbulence

The Reynolds-Averaged Navier-Stokes (RANS) equations are often the starting point for many CFD applications due to the relatively low compute requirements. These can be written as:

$$\frac{\partial \overline{u_i}}{\partial t} + \overline{u_j}\frac{\partial \overline{u_i}}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} + \nu\frac{\partial^2 \overline{u_i}}{\partial x_j^2} - \frac{\partial \overline{u_i' u_j'}}{\partial x_j} \tag{1}$$

where u_i is the velocity in the i direction, the $\overline{u}$ and $\overline{p}$ are the spatiotemporally-averaged velocity and pressure respectively (also referred to as the mean quantities), ρ is the density of the fluid, and u' refers to the eddy velocity. The last term is generally referred to as the Reynolds stress term. In computational fluid dynamics, the mean quantities represent the actual resolved features of the flow, whereas the eddy quantities represent sub grid-scale features. Arguably, one of the most common ways to parameterize turbulence is by invoking the Boussinesq hypothesis and representing the eddy term as a function of the resolved fields

$$\overline{-\rho u_i' u_j'} = 2\mu S_{ij} - \frac{2}{3}\rho\delta_{ij}k \tag{2}$$

where $S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$ is referred to as the mean strain tensor, δ_{ij} is Kronecker delta function, and $k = \frac{1}{2}\overline{u_i' u_j'}$. Turbulence models attempt to compute the two unknown quantities μ and k from the resolved quantities. k is often interpreted as eddy kinetic energy (EKE) or sometimes turbulent kinetic energy.

In [9], a prognostic, PDE-based equation for k attempts to capture the flow of energy between the coarse and fine-scales. Scaling arguments are then used to derive a spatiotemporal value for μ. As noted in the [9], this turbulence model has a number of free parameters and source/sink terms which must be estimated. As an alternative to this PDE-based approach [12], used a neural network trained on data from a large-eddy simulation (LES) simulation of the global ocean to predict the EKE from spatially-filtered fields representative of the spatial resolution from the equivalent RANS solution. They showed that a neural network which predicts EKE from four features (mean kinetic energy, density surface slope, relative vorticity (RV), and the non-dimensional ratio between grid-spacing and scale of turbulence) resulted in significantly more realistic EKE fields than the MEKE parameterization.

3.2 Simplified fluid dynamics problem

This study adopts the AI-based approach introduced in [12], but applies it to a significantly simpler simulation: the Phillips 2-layer case. This is motivated by the desire to make this paper easier to reproduce because no data files need to be shared and requires significantly fewer computational resources. We emphasize that this simplification does not affect the generality of the workflow as it still retains essential characteristics (high and low areas of turbulence, a broad dynamic range of inputs, chaotic solutions, etc.) that are representative of many CFD cases.

The Phillips 2-layer case is a classic problem in fluid dynamics due to its ability to model some of the largest scales of turbulence common in geophysical fluid dynamics [13]. Briefly, the fluid comprises two-layers with different densities, but a non-zero vertical eddy viscosity that serves to transfer momentum between the two layers. The fluid is driven at the surface by a prescribed momentum flux with momentum dissipation occurring through viscous interactions at the side and bottom boundaries. Periodic boundary conditions are specified in the x-direction. When run at sufficient resolution, this simple setup is capable of generating the type of turbulence that dominates large-scale geophysical flows.

This study numerically solves this problem using the MOM6 ocean model [1] with three different configurations. The first (referred to as Phillips-High hereafter) is an LES-scale simulation with a domain size of 240×320 (5km grid spacing) that utilizes only a Smagorinsky viscosity to control grid-scale noise. This case is run on 256 cores for the spin-up (though has also been scaled up to 20,000 cores previously). The second and third are a lower-resolution configuration (referred to as Phillips-Low) with a domain size of 60×80 (20km grid spacing) which is a turbulence-permitting regime, i.e. the grid is sufficient to resolve only a portion of the largest eddy scales. These differ by how the EKE is calculated. Phillips-Low-ML-EKE uses a trained AI model to predict EKE which then uses the MEKE parameterization to calculate the coefficients; Phillips-Low-MEKE uses the previously described MEKE parameterization.

These two simulations serve separate purposes in this workflow. Phillips-High serves as the ground-truth simulation used to generate the training data for the neural network model for turbulence. Phillips-Low-MLEKE and Phillips-Low-Constant represent the lower-resolution models used for weather and climate-scale modelling.

3.3 Implementing the continual learning workflow for turbulence modelling

We use the simulation components described above to implement the workflow described in Section 2. The source code is included in a supporting repository which can be used to recreate the workflow here and as a template for similar workflows. The main workflow definition is the SmartSim driver which coordinates the configuration and execution of each component in step.

The main input to this workflow is the partial dataset used to initially train the neural networks. To create this dataset we first integrate the Phillips-High case for 5 simulated years and store daily, instantaneous data from the sixth year. Features are pre-processed in a similar way as described in Partee et al. [12]. This is demonstrated for one of the features in Figure 3. The raw data from the simulation (Figure 3a) is transformed using a Gaussian filter with a length scale of 80km [7, 10]. This serves to remove the fine-scale features and mimics the smallest-scale that Phillips-Low might be aable to resolve. To demonstrate the out-of-distribution component

of this workflow, we omit a specific portion of the dataset via K-Means clustering. The normalized input features are clustered into 6 clusters; the cluster with the highest, positive relative vorticity is then excluded. These cluster definitions are also saved to be reused by the sampling application.

This partial dataset is trained using two separate AI models, both based on ResNet [8] where one network has simple skip-connections (referred to as EKE-ResNet) and the other has bottleneck layers (referred to as EKE-Bottleneck). These are each trained for a maximum of 800 epochs, though may stop early if the validation loss is less than 0.4. This value is somewhat arbitrary however it results in a dimensional average error of 1.5 $m^3 s^{-2}$ which is an error of around 5% for the areas where the parameterization is most critical. CMF is used here to capture dataset used to train the model and to log the training and validation losses each epoch. These can be used for a posteriori analysis to investigate how the new samples in the training dataset affect the convergence rate of the models. At the end of the training script, the loss for test dataset (the excluded cluster) is logged.

The SmartSim driver script itself plays the role of the Model Evaluator by simply querying the test loss value of each of the trained models. This is crucially important for HPC environments where the driver scripts are often executed on computational nodes which may not have access to GPU resources and/or even the software environments to run AI models. By simply querying a previously stored piece of a metadata from a component of the process, a minimal amount of information can take the place of a more complex component.

The data generation portion of the workflow is done in a streaming fashion using both a data sampling worker and the Phillips-High simulation. Phillips-High is run for 30 days with each rank sending its own portion of the dataset sent via the SmartRedis client to the database every simulated day (in the same manner as the original data were generated). Asynchronously, the data sampler polls the Redis database for available datasets, reconstructs the global array from the rank-by-rank-data, and processes it into features. The previously-saved cluster definitions are then used to select only the portions of the data that fall into the excluded cluster. These datasets are logged using CMF to facilitate the ability to later discover which datasets led to better improvements in training.

The workflow continues to iterate until at least one of the neural networks converges. Again by querying CMF, the SmartSim driver chooses the latest version of this model and loads it into the database. The Phillips-Low-ML-EKE case which incorporates EKE-Bottleneck into its solver is then configured and launched. At every timestep, each rank of the simulation calculates the input features from its portion of the subdomain, sends the input array to the database, requests an inference, and then ingests the result as part of the MEKE parameterization.

At each point of this workflow, the inputs and outputs of each component are tracked by CMF and thus the entire lineage of any artifact and its relationship to other artifacts, can be reconstructed. This capability is built into CMF and the graph can be explored by the user. Figure 4 demonstrates this reconstruction for the last iteration of the continual learning workflow, beginning from the partial dataset and the out-of-distribution samples taken from the previous integration of the Phillips-High case. The retrained EKE-ResNet (leftmost leaf) remains an artifact of the workflow, whereas the re-trained EKE-Bottleneck serves as an input to the Phillips-Low-ML-EKE (rightmost leaf). This graph serves as a visual verification that the workflow executed as intended, e.g. both EKE-ResNet and EKE-Bottleneck consumed the same data and that EKE-Bottleneck was the one deployed in Phillips-Low.

As a demonstration that this continual training workflow behaved as expected, the zonally-averaged EKE from the Phillips-High, Phillips-Low-MEKE, and Phillips-Low-ML-EKE are compared for one snapshot (Figure 5). While it is difficult to directly compare these snapshots because the system is chaotic, the two Phillips-Low simulations generate significantly different profiles of EKE. Similar to the 'true' Phillips-High case, the y-profile of EKE is not symmetric, has grid-scale structure, and has a similar range. In contrast, the Phillips-MEKE case is fairly symmetric around the center of the domain, has a relatively smooth profile, and overestimates the maximum EKE by almost 50%. This would result in a simulation that is likely to smooth and diffuse, suppressing even the small amounts of turbulence that the coarse grid would permit.

4 Discussion

This study focuses on the novel workflows that can be created by simply tracking small, but discrete parts of metadata. The individual components are vastly simplified (e.g. relatively simple neural network architectures, small simulations) in comparison to research-level production. In this discussion, we note how this workflow can be scaled to significantly more complex components and scales that are representative of the types of applications planned for exascale machines. Specifically, we emphasize how tracking metadata for the workflow artifacts vastly simplifies the recordkeeping and historical backtracking that is often crucial to scientific research and eliminates traditional barriers to scale.

Horizontal scaling of this workflow represents more challenges than simple vertical scaling (e.g. increasing the number of compute resources per component due to larger simulations or using more advanced neural network architectures neural networks). Increasing the number of neural network architectures m or using ensembles of high-resolution simulations to generate the data n scales the complexity of the data sharing graph as $m \times n$ if every application can communicate point-to-point. By using the database as a central data-store, the data communication paradigm aligns more closely with a a much simpler, spoke-and-hub type topology instead of a fully-connected network. The traditional bottleneck that arises from having a single hub can be alleviated by sharding the database over multiple nodes. SmartSim and SmartRedis provide support for scaling to large ensembles by monitoring each job individually and providing unique keyspaces for every ensemble member.

For iterative workflows like the ones described here, it can be impractical to store the entirety of the input datasets at every timestep. For large CFD problems these data per integration can be on the order of terabytes to petabytes. Moving traditional simulation/training pipelines to a more producer/consumer paradigm mitigates this problem since intermediate artifacts can be treated as ephemeral. Especially during development, however, the lack of insight into the data itself impedes the ability to develop and iterate.

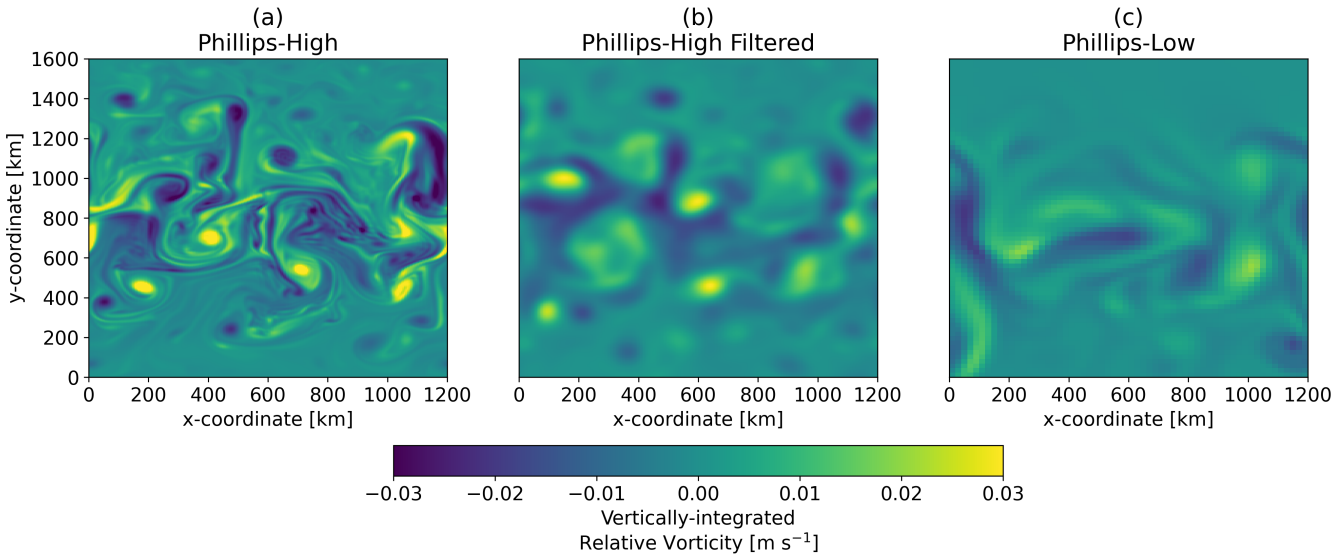

Figure 3: The relative vorticity (one of the four features used to predict eddy kinetic energy) from the Phillips-high at the original resolution (a), the filtered output from Phillips-high used in the training (b), and from the Phillips-Low case. Comparing (b) and (c) shows similar scales of resolved features with similar magnitudes.

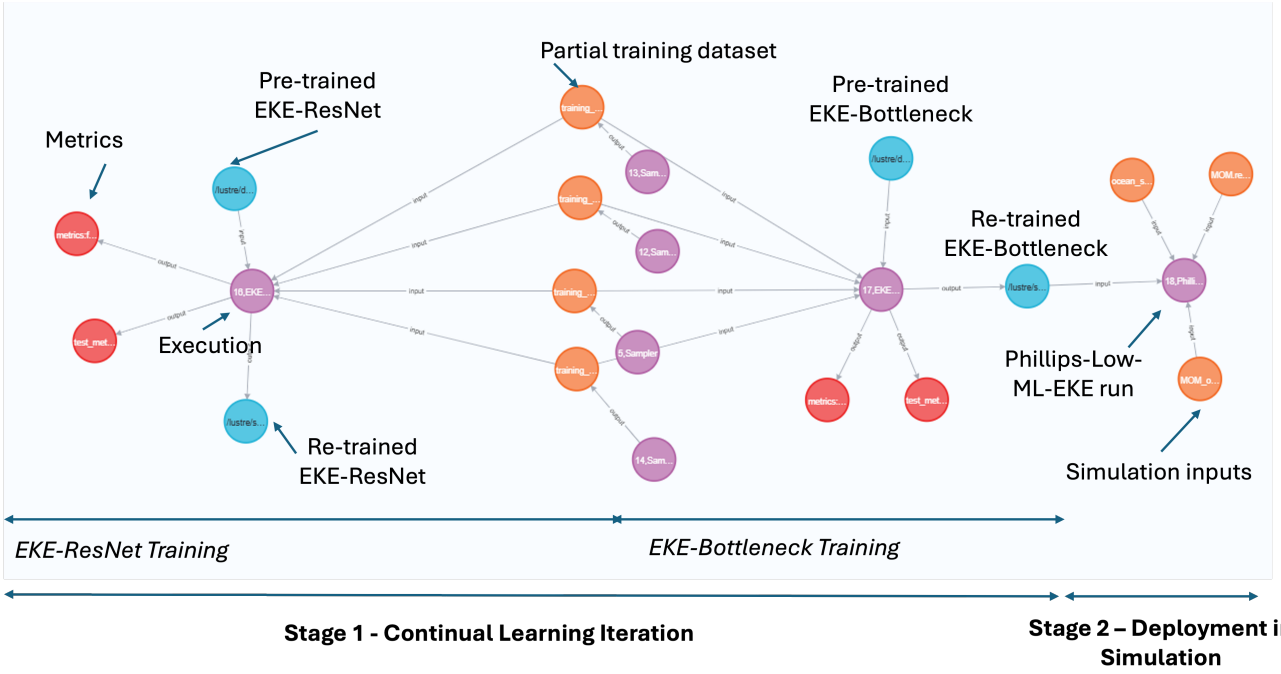

Figure 4: Relationship graph for one iteration of the continual learning workflow which results in the deployment of EKE-Bottleneck into the Phillips-Low-ML-EKE. Every node represents an artifact of the workflow. This data lineage has a direct correspondence to workflow shown in Figure 2, but from a data-centric point of view.

As is the case here, oftentimes it is only important to capture metadata that captures the characteristic of the dataset. For example, in the described workflow we can track the configuration of the simulation and the additional parts of the data that were used in the continual learning. This is sufficient to recreate a single loop of the workflow and better understand which aspects of that data contributed to improved model performance. Tracking metadata which is orders of magnitude smaller than the data itself allows for

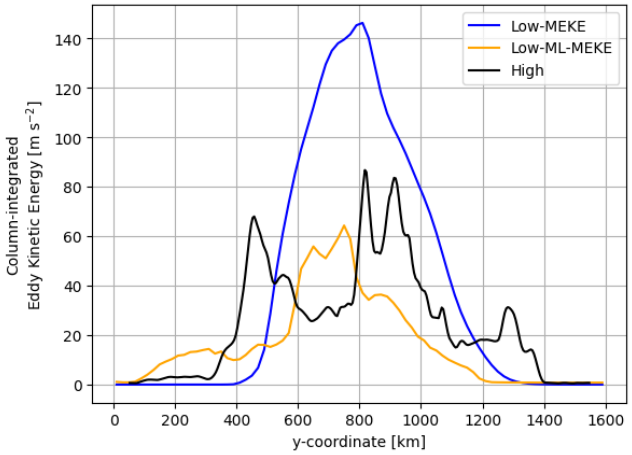

Figure 5: Eddy kinetic energy averaged along the x-direction from the Phillips-High (black line), Phillips-Low-MEKE (blue line), and Phillips-Low-ML-EKE (orange line). Each of these cases are chaotic and these figures are made from snapshots, so exact alignment cannot be expected.

this workflow to scale horizontally, i.e. we can train many times more model architectures on large ensembles of simulations without significantly increasing the storage footprint needed to archive the salient parts of the workflow.

Another key aspect of this workflow and the libraries used here is the ability to keep many aspects of the data in-memory as opposed to using the filesystem as a cache or swap. Despite increases in storage performance, high speed interconnects can transfer data between nodes at least an order of magnitude faster than the read/write speeds of modern NVME drives. The downside is that the total data store is limited by the amount of RAM available to the in-memory datastore. In the case here, the datastore can be sharded across multiple nodes which, for many HPC-machines, have multiple terabytes of RAM.

In the simple CFD example discussed in this work, the outputs from the simulation have been minimally processed before being used to train the AI models. In more complex use cases, the simulation outputs may undergo further transformations or computations before a model is built. An example is density of states (or electronic band structure) computation on molecular structures from molecular dynamics simulations. In those cases it is even more critical to select the most important simulation data for model training as this not only makes model training more practical but also saves computation time and resources.

In iterative workflows that involve data subset selection, the quality of AI model will depend on the acquisition function or algorithm performing the data selection. Historical backtracking enabled by CMF allows to inspect how the AI model performance evolved after expanding the training data set and optimize the acquisition function for future data selection. A variation of this workflow is also possible where a broader subset of simulation data is preserved near the data selection decision boundary to enable a post-hoc, what-if analysis with different acquisition functions, while data far from the decision boundary are discarded similarly to the workflow presented in this work. CMF enables tracking of the simulation and post-hoc experiments in one unified framework.

Overall, driving these workflows using metadata allows domain scientists to construct their applications according to the principle of least information, i.e. tasks should be accomplished with only the minimum amount of information sharing. While this is typically used in the context of security, it has direct relevance to complex workflows at scale where the data generated by the various components may be challenging to share (e.g. synchronizing petabytes of data across datacenters). By focusing on the characteristics of the data generated by each component applications, users can build workflows that follow a producer-consumer model where state synchronization, data discovery, and data movement rely only on small amounts of metadata.

5 Summary and future work

In this work we have developed a framework that enables tracking of artifacts and metadata across all components of a hybrid simulation-AI workflow, including from the workflow manager, the HPC simulation code, the AI inference serving orchestrator, and the AI model training. We demonstrated the benefits of an end-to-end metadata tracking enabled by this framework for AI surrogate model continual learning in an example CFD workflow. This CFD workflow, while simplified, captures many of the characteristics of more complex CFD workflows, namely the simulation is parallelized and deployed on multiple nodes, and generates data at similar frequencies and volumes.

Adaptive switching between different AI models driven by testing the model on previously withheld data was employed during the workflow during the simulation to improve inference output at coarse spatial and temporal resolutions. Given the accuracy on the withheld data the model was periodically based on the detection of out-of-distribution samples encountered during the integration of a high-resolution simulation. CMF was used to track simulation results, the incorporation of new data, and training data across multiple iterations of continuous learning (see Figure 2). The inherent capabilities of CMF allow scientists to unwind changes when results deteriorate, and capture model provenance for future reuse.

The abstractions and concepts introduced here are applicable to broad range of other domain sciences. To facilitate this, all the source code necessary to run the example here is made available as open-source [5] with the initial training dataset available on request, but can also be re-generated using the MOM6 configurations included in that repository. While the particular application discussed in this paper focuses primarily on a CFD-like case, the abstracted workflow maps onto other hybrid HPC/AI motifs currently being studied [3]. We note in particular this study emphasizes the role of metadata in driving the workflow itself instead of being driven by data availability or specific events. We are planning on using the workflow developed here as a template for the Fusion Data Platform (funded by the US Department of Energy) to advance thermonuclear fusion research specifically for extending this work to kinetic equilibrium studies and plasma instability risk modeling.

Acknowledgments

Foltin, Shao, Sharma, Kulkarni, Koomthanam, Tripathy, Dong, Xu, Battacharya, and Faraboschi all gratefully acknowledge the support of internal funding by HPE. In addition, some of the work by Foltin, Sharma, Kulkarni, Koomthanam, Tripathy and Sammuli were supported under Department of Energy award DE-SC0024409 for A Fusion Machine Learning Data Science Platform to Support the Design and Safe Operation of a Fusion Pilot Plant

References

[1] Alistair Adcroft, Whit Anderson, V. Balaji, Chris Blanton, Mitchell Bushuk, Carolina O. Dufour, John P. Dunne, Stephen M. Griffies, Robert Hallberg, Matthew J. Harrison, Isaac M. Held, Malte F. Jansen, Jasmin G. John, John P. Krasting, Amy R. Langenhorst, Sonya Legg, Zhi Liang, Colleen McHugh, Aparna Radhakrishnan, Brandon G. Reichl, Tony Rosati, Bonita L. Samuels, Andrew Shao, Ronald Stouffer, Michael Winton, Andrew T. Wittenberg, Baoqiang Xiang, Niki Zadeh, and Rong Zhang. 2019. The GFDL Global Ocean and Sea Ice Model OM4.0: Model Description and Simulation Features. *Journal of Advances in Modeling Earth Systems* 11, 10 (2019), 3167–3211. doi:10.1029/2019MS001726 arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019MS001726

[2] Abhijeet Awasthi and Sunita Sarawagi. 2019. Continual Learning with Neural Networks: A Review. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data* (Kolkata, India) *(CODS-COMAD '19)*. Association for Computing Machinery, New York, NY, USA, 362–365. doi:10.1145/3297001.3297062

[3] Wes Brewer, Ana Gainaru, Frédéric Suter, Feiyi Wang, Murali Emani, and Shantenu Jha. 2024. AI-coupled HPC Workflow Applications, Middleware and Performance. arXiv:2406.14315 [cs.DC] https://arxiv.org/abs/2406.14315

[4] Hewlett Packard Enterprise. 2025. CMF. https://github.com/HewlettPackard/cmf.

[5] Hewlett Packard Enterprise. 2025. Metadata Driven Workflow CUG 2025. https://github.com/ashao/metadata-driven-workflow-cug-2025.

[6] Hewlett Packard Enterprise. 2025. SmartSim. https://github.com/CrayLabs/SmartSim.

[7] I. Grooms, N. Loose, R. Abernathey, J. M. Steinberg, S. D. Bachman, G. Marques, A. P. Guillaumin, and E. Yankovsky. 2021. Diffusion-Based Smoothers for Spatial Filtering of Gridded Geophysical Data. *Journal of Advances in Modeling Earth Systems* 13, 9 (2021), e2021MS002552. doi:10.1029/2021MS002552

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Malte F. Jansen, Alistair Adcroft, Sina Khani, and Hailu Kong. 2019. Toward an Energetically Consistent, Resolution Aware Parameterization of Ocean Mesoscale Eddies. *Journal of Advances in Modeling Earth Systems* 11, 8 (2019), 2844–2860. doi:10.1029/2019MS001750 arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019MS001750

[10] Nora Loose, Ryan Abernathey, Ian Grooms, Julius Busecke, Arthur Guillaumin, Elizabeth Yankovsky, Gustavo Marques, Jacob Steinberg, Andrew Slavin Ross, Hemant Khatri, Scott Bachman, Laure Zanna, and Paige Martin. 2022. GCM-Filters: A Python Package for Diffusion-based Spatial Filtering of Gridded Data. *Journal of Open Source Software* 7, 70 (2022), 3947. doi:10.21105/joss.03947

[11] Tomislav Maric, Mohammed Elwardi Fadeli, Alessandro Rigazzi, Andrew Shao, and Andre Weiner. 2024. Combining machine learning with computational fluid dynamics using OpenFOAM and SmartSim. *Meccanica* (2024), 1–20.

[12] Sam Partee, Matthew Ellis, Alessandro Rigazzi, Andrew E. Shao, Scott Bachman, Gustavo Marques, and Benjamin Robbins. 2022. Using Machine Learning at scale in numerical simulations with SmartSim: An application to ocean climate modeling. *Journal of Computational Science* 62 (2022), 101707. doi:10.1016/j.jocs.2022.101707

[13] Norman A. Phillips. 1951. a Simple Three-Dimensional Model for the Study of Large-Scale Extratropical Flow Patterns. *Journal of the Atmospheric Sciences* 8, 6 (Dec. 1951), 381–394. doi:10.1175/1520-0469(1951)008<0381:ASTDMF>2.0.CO;2

[14] Rishabh Tiwari, Krishnateja Killamsetty, Rishabh Iyer, and Pradeep Shenoy. 2022. Gcr: Gradient coreset based replay buffer selection for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 99–108.

Received 8 April 2025