Asynchronous Scientific Workflows for Simulation-Driven Inference Across the Edge–HPC Computing Continuum

UCSB Technical Report #2026-01

Liubov Kurafeeva[a,*], Ryan Hartung[b], Benjamin C. Carter[a], Alan Subedi[c], Avhishek Biswas[c], Michael Fay[c], Shantenu Jha[d,e], Chandra Krintz[a,*], Andre Merzky[e], Douglas Thain[b], Mehmet Can Vuran[c], Rich Wolski[a]

[a]*Department of Computer Science, University of California, Santa Barbara, Santa Barbara, CA, USA2104 Harold Frank Hall, Santa Barbara, CA, 93106-5110,*
[b]*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA84 Fitzpatrick Hall of Engineering, Notre Dame, IN, 46556,*
[c]*School of Computing, University of Nebraska–Lincoln, Lincoln, NE, USA256 Avery Hall 1144 T Street Lincoln, NE, 68588-0115,*
[d]*Princeton Plasma Physics Laboratory, Princeton, NJ, USAP.O. Box 451, Princeton, NJ, 08543-0451,*
[e]*Department of Electrical and Computer Engineering, Rutgers University–New Brunswick, New Brunswick, NJ, USA94 Brett Road, Piscataway, NJ, 08854-8058,*

Abstract

Scientific workflows are increasingly executed across heterogeneous computing continuum spanning edge devices, dedicated clusters, cloud resources, and shared high-performance computing (HPC) systems. Many emerging applications require continuous low-latency inference from streaming observations while periodically incorporating improved models generated by computationally intensive simulations. These competing requirements make workflow orchestration challenging: inference must proceed continuously even though simulation and model generation are delayed by resource availability, workflow scheduling, and batch queue dynamics.

We present RBF (Reverse Backfill), an asynchronous scientific workflow architecture for simulation-driven inference across the edge–HPC computing continuum. RBF decouples workflow stages through the Workflow Artifact Manager (WAM), which maintains persistent workflow state and versioned workflow artifacts, enabling resilient asynchronous coordination across heterogeneous resources. Dedicated computing resources provide predictable model regeneration while shared HPC systems opportunistically augment workflow throughput without modifying existing batch schedulers. To exploit this opportunistic computation effectively, RBF introduces a decision-time scheduling heuristic that determines whether a queued HPC workflow stage is expected to improve the currently deployed surrogate model, allowing shared resources to be consumed only when the anticipated benefit justifies the allocation.

We instantiate RBF using a digital agriculture application that couples edge sensing with computational fluid dynamics simulations to infer spatial airflow within a large citrus screenhouse. Experimental results characterize end-to-end workflow behavior, including simulation latency, model regeneration cadence, workflow staleness, opportunistic execution, and edge deployment overhead. Our results show that asynchronous workflow execution maintains continuous low-latency inference while opportunistic HPC execution significantly reduces model staleness and selectively improves model quality through decision-time adaptation. More broadly, RBF demonstrates how scientific workflows can effectively integrate shared HPC resources into operational cyber-physical systems across the computing continuum without requiring changes to existing scheduling policies.

Keywords: Scientific workflow architecture, Hybrid physics–data modeling, Edge–HPC computing continuum, Computational fluid dynamics (CFD), Surrogate modeling, Cyber-physical systems, Controlled-environment agriculture

1. Introduction

Scientific workflows increasingly span the computing continuum, integrating edge devices, dedicated clusters, cloud resources, and shared high-performance computing (HPC) systems into end-to-end computational pipelines. These workflows must coordinate persistent workflow state, exchange intermediate artifacts, and tolerate heterogeneous execution latencies while continuously producing useful results. An emerging class of such workflows supports simulation-driven inference, in which streaming sensor observations collected at remote facilities parameterize high-fidelity simulations whose outputs are used to construct surrogate models for low-latency inference near where the data are generated.

Simulation-driven workflows present a fundamental orchestration challenge. Edge applications require continuous, low-latency inference, while the high-fidelity simulations needed to improve surrogate models execute on remote HPC systems with variable queue delays and execution times. Consequently, workflow stages cannot be tightly synchronized. Instead, applications must continue operating with existing surrogate models while incorporating improved models asynchronously as new, more up-to-date, workflow artifacts become available.

To address these challenges, we present RBF (Reverse

*Corresponding author
Email addresses: liubov_kurafeeva@ucsb.edu (Liubov Kurafeeva), ckrintz@ucsb.edu (Chandra Krintz)

Backfill), an asynchronous scientific workflow architecture for simulation-driven inference across the computing continuum. Rather than treating HPC systems as synchronous backends, RBF exploits opportunistically available HPC resources to continuously improve surrogate models while edge applications perform uninterrupted inference. Persistent workflow state is maintained through the Workflow Artifact Manager (WAM), which enables versioned workflow artifacts, asynchronous coordination, and resilient communication across distributed workflow stages. As improved simulation results and surrogate models become available, they are incorporated into the workflow without interrupting inference.

Unlike traditional scientific workflows that execute a predefined sequence of tasks, RBF supports continuously executing workflows whose stages repeatedly exchange evolving workflow artifacts as new sensor data, simulation outputs, and surrogate models become available. This enables the workflow to adapt continuously to changing operating conditions while remaining resilient to the scheduling variability inherent in shared HPC systems.

The workflow architecture is built around three key principles. First, it decouples workflow stages through persistent workflow state and asynchronous artifact exchange, allowing edge inference to proceed while simulation and model training execute independently. Second, it treats simulation outputs, training data, and surrogate models as versioned workflow artifacts, enabling pluggable simulation and surrogate modeling components while supporting reproducibility and flexible tradeoffs among accuracy, latency, and computational cost. Third, it coordinates workflow execution across heterogeneous edge, cluster, cloud, and HPC resources, enabling continuous execution despite highly variable simulation and job scheduling latencies. Concretely, these principles are realized by three elements: persistent workflow state, held by the Workflow Artifact Manager; asynchronous workflow execution across tiers; and a decision-time scheduling heuristic that determines whether a queued HPC workflow stage is expected to improve the currently deployed surrogate model. The first two make opportunistic HPC participation possible; the third decides when it is worthwhile.

We demonstrate the workflow architecture using a digital agriculture application that couples edge sensing with computational fluid dynamics (CFD) simulations to infer spatial airflow patterns in a large agricultural screenhouse. This application serves as a realistic testbed for evaluating workflow execution under representative operating conditions, including HPC queue delays, asynchronous model updates, and variable simulation throughput.

Our evaluation focuses on workflow-level tradeoffs in simulation-driven inference. We characterize the latency and throughput of the end-to-end workflow, evaluate the impact of artifact staleness on prediction accuracy, and measure the overhead associated with workflow artifact dissemination and edge inference. The results show that overall workflow performance is dominated by simulation latency, but this latency can be masked effectively, and that networking, communication, and inference overheads are negligible relative to simulation latency.

In summary, this paper makes the following contributions:

- **Asynchronous scientific workflow architecture.** We present an architecture for simulation-driven inference that decouples continuous edge inference from simulation-driven model evolution, allowing workflow stages to execute asynchronously across the computing continuum.
- **Persistent workflow state and artifact management.** We introduce the Workflow Artifact Manager (WAM), which maintains versioned workflow artifacts and enables resilient, asynchronous coordination among distributed workflow stages.
- **Workflow orchestration across the computing continuum.** RBF coordinates sensing, simulation, model training, and inference across edge devices, dedicated clusters, cloud resources, and shared HPC systems while maintaining continuous low-latency inference at the edge.
- **Execution-time workflow adaptation of opportunistic HPC stages.** RBF exercises workflow-level execution control at the moment an opportunistic HPC workflow stage reaches the head of the batch queue, predicting whether that execution should proceed from its expected effect on the currently deployed surrogate model, and thereby transforming opportunistic HPC resources into selectively executed workflow stages.

Taken together, these contributions demonstrate that persistent workflow state and asynchronous artifact management enable continuously executing simulation-driven workflows that span edge devices, dedicated clusters, and shared HPC resources while maintaining uninterrupted low-latency inference. RBF further improves workflow throughput by selectively exploiting opportunistic HPC execution without disrupting ongoing workflow execution.

The remainder of the paper is organized as follows. Section 2 presents the RBF architecture, including its design goals, workflow model, Workflow Artifact Manager, and mechanisms for orchestrating simulation, training, and model deployment across the computing continuum. Section 3 describes an instantiation of this workflow architecture in a real-world agricultural screenhouse deployment. Section 4 evaluates the workflow using this deployment and characterizes the resulting end-to-end performance and tradeoffs across the simulation, training, artifact management, and inference pipeline.

2. Workflow Architecture

RBF is designed to support continuously executing, simulation-driven scientific workflows that span edge devices, dedicated clusters, cloud resources, and shared HPC systems. Rather than viewing these resources as independent computing platforms, RBF organizes them into a single workflow in which persistent workflow artifacts are exchanged asynchronously among workflow stages. This decouples workflow execution from resource availability, allowing stages to execute independently while maintaining continuous low-latency inference despite variable simulation and scheduling delays.

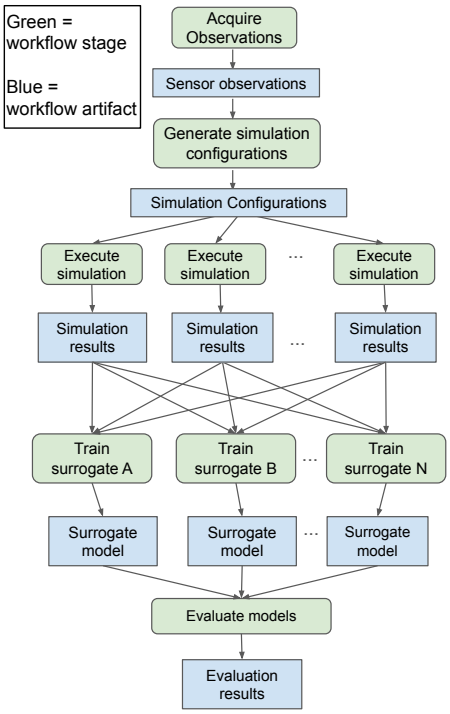

Figure 1: RBF Workflow. One iteration of the continuously executing simulation-driven workflow. Workflow stages (green) consume and produce persistent workflow artifacts (blue). Parallel simulation stages generate simulation artifacts that are used to train multiple surrogate models, which are subsequently evaluated for deployment. Persistent workflow artifacts are maintained by the Workflow Artifact Manager (WAM), enabling fault resilient, asynchronous execution and coordination across iterations of the workflow.

Figure 1 illustrates one iteration of the RBF workflow. Workflow stages (green) consume and produce persistent workflow artifacts (blue), which are maintained by the Workflow Artifact Manager (WAM). Streaming sensor observations are transformed into simulation configurations that drive multiple parallel simulations. The resulting simulation artifacts are used to train multiple surrogate models, which are subsequently evaluated before deployment. Although Figure 1 shows a single iteration, the workflow executes continuously as new observations arrive and updated workflow artifacts become available.

The workflow is realized across the computing continuum using the three-tier architecture shown in Figure 2. The edge tier (left) generates sensor data, performs real-time data ingestion, and executes low-latency inference. The dedicated cluster tier (middle) executes simulation and surrogate model training using recent data. The shared HPC tier (right) provides additional, opportunistic simulation and surrogate training capacity. The WAM is a fault-resilient, delay-tolerant distributed log that serves as the primary mechanism for data exchange, model versioning and dissemination, and workflow coordination. By persisting workflow state, the WAM enables these heterogeneous resources to operate independently while coordinating through versioned workflow artifacts.

At a high level, RBF operates as a continuous inference loop. Edge devices ingest streaming sensor data and perform low-latency inference using the most recently available surrogate model. In parallel, simulation and training pipelines consume this data to generate updated models. These pipelines execute at different cadences: the dedicated cluster provides regular but resource-constrained updates, while the HPC tier contributes additional models opportunistically as resources become available. As new models are produced, they are published to the WAM and asynchronously deployed to the edge, where they replace older models and reduce prediction error by reducing model staleness.

This design enables RBF to maintain continuous inference despite delays in simulation and training, while improving model fidelity over time. The remainder of this section describes the key elements of the workflow architecture, including the edge inference tier, the simulation and training pipeline, the use of reverse backfilling in HPC systems, and mechanisms for model orchestration and deployment. This architecture distinguishes RBF from continuous learning systems, in which models are updated directly from streaming observations. In RBF, model updates are mediated by simulation, making update latency and variability a fundamental design consideration.

2.1. Edge Inference Tier

The edge inference tier is responsible for continuous data ingestion, low latency prediction on fresh data, and interaction with the physical environment. Sensor devices deployed within the target system generate streaming measurements (e.g., environmental, operational, or physical state data) that are transmitted over a local wireless network and published to the WAM. The WAM provides a fault-resilient, decoupled interface between producing and downstream workflow stages, enabling sensor data to be persistently recorded and asynchronously consumed by both inference services at the edge and remote simulation and training pipelines. The WAM APIs include both "push" and "pull" data movement styles to enable deployments to span environments where firewalls do not allow bidirectional network connectivity between all devices.

Edge inference services subscribe to both data and model WAM streams. Local sensor data is processed in near real time using the most recently available surrogate model, enabling low-latency prediction for decision support, actuation, and control at the edge. Concurrently, edge services receive publish events from the model stream. When updated surrogate versions generated by the simulation and training tiers become available, they are retrieved and deployed asynchronously without interrupting inference. This mechanism allows the system to improve model fidelity over time while maintaining uninterrupted operation, ensuring that predictions reflect the most current available system state despite upstream delays. We refer to the time elapsed since the currently deployed model was published as its *staleness*; each new model deployment resets it to zero, and the accuracy of a served model degrades over time as the staleness increases.

The edge tier is designed to support heterogeneous and evolving networking environments. In particular, it accommodates private wireless infrastructure, including production and experimental 5G/6G deployments, to enable reliable, low-latency communication between sensors, inference devices, and upstream resources. Advanced wireless capabilities such as network slicing, dynamic resource allocation, and programmable radio access

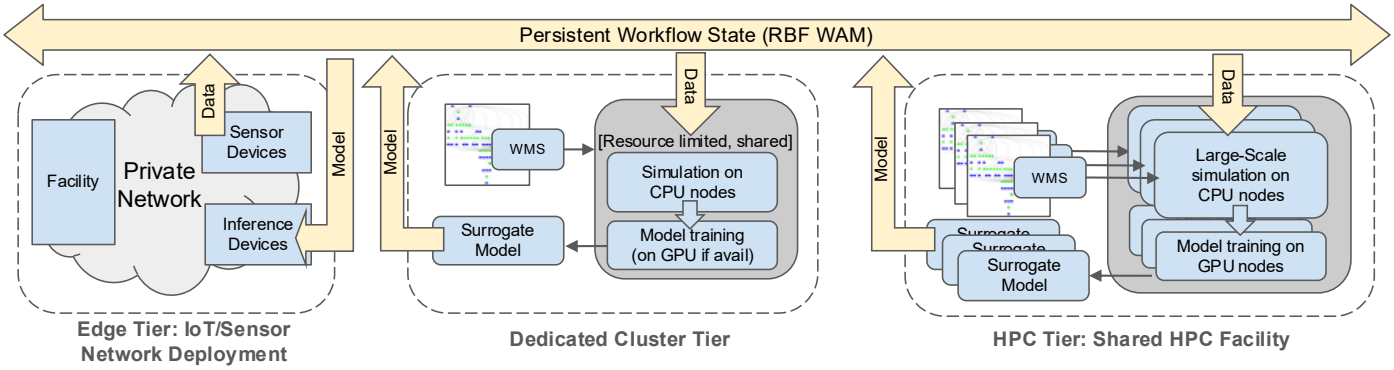

Figure 2: RBF System Architecture. The Reverse Backfill architecture spans three tiers connected through the Workflow Artifact Manager (WAM), which provides persistent workflow state and versioned workflow artifacts across the computing continuum. (Left) At the remote facility, sensor observations are collected and published to the WAM over a private wireless network. The edge inference service retrieves the latest published surrogate model from the WAM and performs low-latency inference in place of simulation. (Middle) A dedicated, resource-limited cluster consumes sensor observations, executes facility simulations and trains updated surrogate models by leveraging a local workflow management system (WMS), and publishes the resulting workflow artifacts back to the WAM. (Right) A shared HPC facility is used to execute additional simulations asynchronously (using the facility WMS), increasing workflow throughput despite variable resource availability and queue wait times. The resulting simulation artifacts augment those generated by the dedicated cluster by filling intermediate time points.

networks can be incorporated to isolate traffic classes, prioritize inference-critical data, and explore new communication patterns for the cyber-physical applications. By decoupling networking from application logic, RBF allows these capabilities to be integrated and evaluated without requiring changes to the overall system architecture.

Finally, the edge inference tier supports pluggable surrogate models, enabling multiple model types to be deployed and evaluated within the same operational environment. Models may vary in complexity, accuracy, and computational cost, and can be dynamically selected or replaced based on application needs. This flexibility allows RBF to serve as a platform for exploring tradeoffs in model design, inference latency, and communication overhead, and for integrating new modeling techniques and wireless capabilities into next-generation IoT, cyber-physical, and digital twin deployments.

2.2. Simulation and Training Pipeline

The simulation and training pipeline is responsible for generating updated surrogate models from streaming sensor data and high-fidelity simulation outputs. Conceptually, this pipeline transforms observations of the physical system into improved predictive models that are deployed at the edge. Unlike traditional tightly coupled workflows, RBF decouples data ingestion, simulation, and model training, allowing these stages to execute asynchronously across heterogeneous resources.

Sensor data published to WAM is consumed by simulation services, which construct and execute high-fidelity models of the underlying system. The resulting simulation outputs are then used to train surrogate models that approximate system behavior while enabling low-latency inference. Both simulation and training may be performed on the dedicated cluster tier or opportunistically on shared HPC resources, as described in the following subsection. The outputs of this pipeline are versioned surrogate models that are published back to the WAM for asynchronous deployment at the edge.

Similar to the edge tier, a key feature of the simulation and training pipeline is the pluggability of both components. The pipeline is designed to support multiple simulation frameworks (e.g., CFD solvers, multi-physics models, or domain-specific simulators) and a range of surrogate modeling approaches, including physics-informed neural networks, operator-learning methods, and lightweight regression models. These stages are integrated through well-defined data and model interfaces, allowing new simulation workflows and learning methods to be incorporated without modifying the overall workflow architecture.

This modular design enables RBF to support a broad class of simulation-driven applications and to facilitate exploration of accuracy–latency–cost tradeoffs across different modeling approaches. By decoupling simulation and training from inference and exposing them as interchangeable components, RBF provides a flexible foundation for integrating emerging simulation technologies and machine learning methods into distributed cyber-physical systems.

2.3. Reverse Backfill on HPC Systems

The RBF architecture incorporates shared HPC systems as an asynchronous, opportunistic source of simulation and model improvement. Unlike the dedicated cluster tier, which provides predictable but resource-limited execution, HPC systems perform batch scheduling with queue delays, variable start times, job priorities, personal access limits, and limited guarantees on execution cadence. Rather than treating these characteristics as constraints to be avoided, RBF explicitly embraces them in the system design.

In traditional HPC environments, “backfilling” schedules smaller jobs into unused gaps in the execution timeline, serving objectives such as utilization, makespan, or deadline satisfaction. RBF is not an alternative to such schedulers but a complement to them: it leaves scheduling policy untouched and instead applies the capacity backfilling exposes to a different end, improving model accuracy within a continuous inference pipeline. Specifically, RBF submits simulation and training jobs to shared HPC facilities and allows them to be scheduled and executed when-

ever resources become available. These jobs may run concurrently with or independently of those in the dedicated cluster tier, and complete at irregular intervals due to queue contention and system variability.

When HPC jobs complete, the resulting simulation outputs are used to retrain surrogate models, which are then published to the distributed log and made available to the edge inference tier. This process effectively injects additional model updates into the system at intermediate time points, increasing the frequency of model refresh relative to the baseline provided by the dedicated cluster. Although the timing of these updates is unpredictable, each completed job contributes to reducing model staleness and improving inference accuracy.

Importantly, RBF does not assume synchronization between inference and HPC execution. Edge inference proceeds continuously using the most recent available model, while HPC-generated updates are incorporated asynchronously as they arrive. This decoupling allows the system to tolerate long-running simulations, queue delays, and network variability without interrupting operation. Conceptually, the HPC tier functions as an eventually consistent model-improvement engine: simulation and training refine model fidelity over time, but are not required for immediate inference.

To support opportunistic execution across heterogeneous HPC systems, RBF employs a pilot-based resource acquisition strategy. Rather than binding simulation workloads to a single target system, RBF evaluates available HPC resources based on factors such as predicted queue wait time, current availability, and allocation constraints, and submits pilot jobs to multiple candidate systems. The size and duration of these pilot requests are selected to balance responsiveness with resource utilization. Once a pilot becomes active, simulation tasks are dispatched to the corresponding resource, enabling execution to proceed on whichever system becomes available first. This approach allows RBF to exploit variability in queue behavior across systems and to preferentially utilize higher-performance resources, such as GPU-enabled nodes, when available. Intermediate data and simulation inputs are staged through the WAM, ensuring consistent data access and decoupling task placement from data movement.

This design enables RBF to leverage large-scale HPC resources without introducing blocking dependencies into the inference loop. Moreover, this approach provides a mechanism for integrating batch-oriented HPC workflows into asynchronous cyber-physical systems without requiring changes to underlying scheduling policies or execution environments.

2.4. Data and Model Orchestration

RBF uses a distributed, fault-resilient log called the *Workflow Artifact Manager (WAM)*, as the primary mechanism for coordination and orchestration across the system. Our use of a distributed log as the underlying workflow abstraction is a key design decision in RBF. The log serves as the shared coordination mechanism for all workflow stages, providing an immutable, append-only record of workflow artifacts that establishes a consistent ordering of events, enables independent reconstruction of persistent workflow state, and eliminates the need for direct interoperation among distributed services.

As such, the WAM provides a unifying abstraction for publishing, persisting, and consuming both sensor data and model artifacts in RBF, enabling loose coupling between workflow stages and allowing each tier to operate asynchronously. This makes the WAM the orchestration substrate for the architecture: because every inter-stage dependency is a durable, versioned artifact rather than a direct connection, no two stages need be co-resident or co-scheduled.

Sensor data generated at the edge is published to WAM and made available to downstream workflow stages (via triggered functions), including the simulation and training pipelines executing on the dedicated cluster and HPC tiers. By decoupling producing stages from consuming stages, RBF allows simulation workflows to be triggered and executed based on data availability rather than fixed schedules, supporting both continuous ingestion and opportunistic processing. This same mechanism is used to stage other workflow artifacts (e.g. simulation inputs and intermediate data), ensuring consistent and portable access across heterogeneous execution environments.

The WAM also serves as the distribution mechanism for surrogate models. Models produced by the simulation and training pipeline are versioned and published to the log, where they can be asynchronously discovered and retrieved by edge inference services. This enables seamless model updates without interrupting inference and provides a natural mechanism for managing model lifecycles, including versioning, replacement, and rollback. Because model updates may arrive at irregular intervals, the WAM ensures that the most recent available model is always accessible, while allowing older versions to be retained for comparison and evaluation.

By unifying sensor data, simulation inputs, and model artifacts within a single abstraction, RBF also enables consistent workflow orchestration across all tiers of the system. This design supports fault tolerance through persistent data storage, simplifies integration of new stages, and facilitates experimentation with alternative dataflow patterns, simulation pipelines, and model deployment strategies. More broadly, the log-based architecture allows RBF to integrate diverse simulation frameworks, learning methods, and networking technologies without introducing tight coupling between workflow stages.

3. System Instantiation

We instantiate RBF using a real-world digital agriculture deployment that couples edge sensing with CFD simulations to infer airflow within a large agricultural screenhouse. This deployment serves as a representative realization of the software architecture described above and enables evaluation under realistic sensing, networking, and HPC constraints. It is one instantiation of a general workflow architecture; nothing that follows is specific to agriculture.

As shown in Figure 3, the test deployment implements an asynchronous, simulation-driven pipeline in which data collection, simulation, and training stages overlap across multiple pipeline instances, and model updates are published opportunistically as they complete. The goal is to maintain continuous,

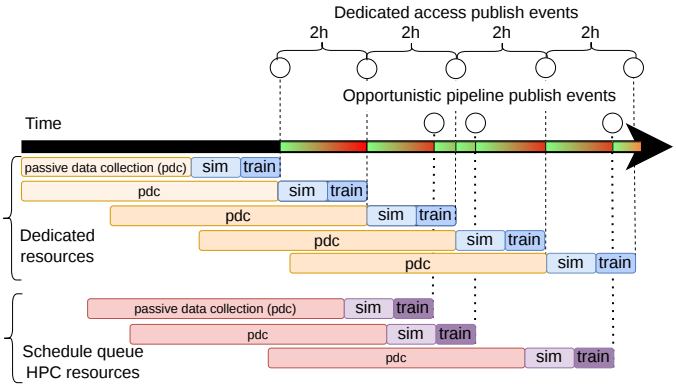

Figure 3: Timeline of the RBF instantiation illustrating asynchronous, simulation-driven model updates. Passive data collection (pdc), simulation (sim), and training (train) stages overlap across multiple pipeline instances, while model updates are published opportunistically upon completion. This approach enables continuous inference despite irregular and delayed HPC execution.

low-latency inference while mitigating model staleness under irregular and delayed HPC-driven model updates.

A set of simulations (*sim*) are launched in parallel on dedicated resources using data collected up to the time of execution (denoted *pdc* (for passive data collection) in Figure 3). Training begins only after all simulations complete and simulation outputs are transformed for training and transferred to training resources (e.g. GPUs). Once training finishes, a new pipeline instance is initiated using the most recent data, resulting in overlapping pipeline executions.

In our deployment, the combined *sim* and *train* stages require approximately two hours using the dedicated cluster tier. Each completed training stage produces a new surrogate model that is published to the edge. As a result, models deployed using only dedicated resources may be up to two hours old.

To reduce model staleness, the pipeline is also executed opportunistically on the HPC tier using shared HPC systems. Jobs are submitted to batch queues using workflow management systems and parameterized with the most recent data at the time of execution. Models produced by these runs arrive at the edge between those generated by the dedicated pipeline, effectively increasing the rate of model updates despite irregular execution. This approach increases the frequency of model updates without requiring execution prediction and directly mitigating the effects of model staleness.

Because pipeline execution time can be different for batch and dedicated systems, an opportunistic model may occasionally arrive after a dedicated model yet carry an older training cutoff date—the latest sensor timestamp included in the training data. To handle this, RBF records the training cutoff time with each model artifact. Before updating the deployed model, the edge service compares the training cutoff time against that of the currently deployed model and skips the update if the incoming model is older. This ensures that the deployed model's training data is monotonically non-decreasing in freshness, regardless of the order in which jobs from different resource tiers complete.

3.1. RBF *Evaluation Facility*

We implement this pipeline using an Evaluation Facility that makes continuous meteorological inferences for a 4-acre Citrus Under Protective Screens (CUPS) facility. CUPS is a large ($200 \times 100 \times 6$ meter) agricultural screenhouse that is designed to protect citrus trees from the insect that vectors the causal agent of the deadly Huanglongbing (HLB) disease (also known as citrus greening) [1]. The screen mesh is fine enough to protect the crop but also modifies the environmental conditions under which citrus is grown [2, 3]. The CUPS application monitors and predicts these conditions to inform, actuate, and control irrigation planning, spray management, and frost prevention for a CUPS installation.

The deployment consists of a network of sensors that collect meteorological data, including wind speed, wind direction, temperature, and humidity inside and outside the CUPS. Measurements are used to parameterize high-fidelity computational fluid dynamics (CFD) simulations that model airflow within the screenhouse. The resulting simulation outputs are used to train surrogate models that enable low-latency inference of the conditions inside the CUPS.

We use OpenFOAM [4] to perform CFD simulations, configured to model the screenhouse geometry and boundary conditions derived from the sensor data. The prototype implements the *sim* stage as 72 parallel OpenFOAM simulations each configured with the SnappyHexMesh mesh generator [5] and PorousSimpleFOAM [6] solver, to capture the CUPS screen-filtered atmospheric effects of the CUPS growing conditions across approximately 7 million irregularly shaped cells at the centimeter level.

Each of the 72 simulations corresponds to one five-minute sensor reading in a 6 hour history (we analyze and evaluate this history size in Section 4.2). Each simulation uses the sensor readings (wind speed and direction) from outside the CUPS as inlet boundary conditions, holds the geometry and solver configuration fixed, and outputs a velocity field at the 9690 points inside the CUPS.

Simulation outputs are used to train multiple surrogate models, including a physics-informed neural network (PINN) [7, 8], a Fourier neural operator (FNO) [9], and a principal component regression (PCR) [10] model. This instantiation leverages the pluggable design of RBF, enabling multiple surrogate models to be trained and evaluated under identical system conditions. The models vary in complexity and computational cost, enabling evaluation of accuracy–latency tradeoffs at the edge.

We implement the HPC tier using the NERSC Perlmutter system [11]. RBF runs *sim* and *train* stages using a RADICAL-Pilot [12] job which acquires the most recent data available. The *sim* stage requests 72 nodes with 32 cores each, matching the dedicated cluster tier. The *train* stage reuses 2 CPU nodes. Jobs are submitted to the `regular` Perlmutter queue with a 24-hour limit. RBF keeps at most one job of each type queued, resubmitting only after the previous job reaches its limit. During each allocation, RBF continuously executes workflow iterations (using the most recent data each time) until the allocation is exhausted. We refer to this as "back-to-back" runs or model generations, herein.

Trained models are deployed to edge inference devices (Raspberry Pi single board computers [13]), where they are used to make predictions of airflow within the screenhouse. Sensor devices stream sensor observations to the WAM for consumption by both the simulation pipeline and inference devices.

3.2. Workflow Artifact Management

RBF orchestrates workflow execution by exchanging persistent workflow artifacts between computational stages rather than requiring direct communication between executing components. In the current implementation, each workflow artifact is produced by one workflow stage and consumed by downstream workflow stages. For example, the simulation stage generates simulation results artifacts that, after optional transformation, become inputs to the surrogate model training stage.

To provide persistent workflow state across the edge–cloud–HPC continuum, we extended the open-source CSPOT log-based event system [14] with a lightweight workflow artifact management capability. The resulting RBF WAM stores workflow artifacts as versioned objects within persistent CSPOT logs, enabling asynchronous, delay-tolerant communication between workflow stages. Workflow stages publish artifacts using an artifact "push" operation, while downstream workflow stages retrieve them using a corresponding "pull" operation. Because the CSPOT repositories reside outside local firewall boundaries, geographically distributed workflow stages exchange artifacts through the shared persistent log without requiring direct network connectivity.

Each workflow artifact is versioned automatically. CSPOT assigns a monotonically increasing sequence number to every log entry, allowing the WAM to associate each artifact version with the sequence range over which it is stored. Workflow stages may retrieve either a specific artifact version or the latest available version through the WAM API. Consumers detect newly published artifacts by monitoring or being triggered by an append of the latest available version, enabling downstream workflow stages to execute asynchronously as new workflow state becomes available.

All RBF components that consume or produce persistent workflow artifacts use the WAM interface. For example, edge devices retrieve the latest surrogate models through the WAM, simulation workflows publish their outputs for downstream training, and training stages publish updated models for deployment. The same mechanism also supports software distribution: statically linked executables generated by the CI/CD system are published as versioned workflow artifacts and automatically deployed when newer versions become available. Secure communication is provided across RBF using CSPOT capabilities (called CAPlets [15]). RBF relies on this directly: a job canceled before it computes leaves the state it would have consumed unchanged, so cancellation does not impede workflow progress.

3.3. Private 5G Network

In this study, we deploy RBF across two different Private 5G networks. The first is a 5G private cell sited at an agricultural facility that we use for empirical evaluation of RBF. It uses a Celona 5G cell, model AP25-48 [16] of which is an off-the-shelf commercial solution for 5G private cellular networking. Because the site is located in a rural setting, the backhaul network is implemented using satellite Internet service provided by Starlink [17]. This 5G network is an example of commercially available 5G technologies that are designed for outdoor deployment in remote areas and long-term unattended operation.

We also include experiments that use a laboratory-sited private 5G network built with two open source components: srsRAN Project [18] for the Radio Access Network (RAN) and Open5GS [19] for the 5G core. Both are deployed as Docker containers on a dedicated edge host, communicating over an internal bridge network.

The 5G base station (gNB) is driven by an Ettus Research USRP B210 software defined radio operating on 3GPP Band n78 with a 40 MHz channel bandwidth and 30 kHz subcarrier spacing in TDD mode. The core network implements a full 5G Standalone (SA) architecture, including the AMF, SMF, UPF, NSSF, and supporting network functions.

Together, these devices provide a self-contained cellular network with no dependency on external carrier infrastructure. Thus, this srsRAN cell (deployed in a controlled laboratory setting) captures forward-looking technological improvements that will eventually be available commercially in a weatherproof package for unattended outdoor deployment.

Two Raspberry Pi 4 Model B single board computers serve as user equipment (UEs). Each is fitted with a Quectel 5G modem connected over USB. The modem exposes a QMI control interface and AT command ports to the host operating system. Each modem carries a programmable SIM provisioned with a unique IMSI and authentication credentials registered in the Open5GS subscriber database. On power up, the modem attaches to the NR SA cell and establishes a PDU session, which surfaces as a `wwan0` network interface on the Raspberry Pi with a statically assigned IP address. The two Raspberry Pi nodes serve distinct roles, each defining a separate data path across the RAN.

The *model distribution path* delivers surrogate model updates from the WAM to the edge inference node, a flow that demands low transfer latencies. The *general data path* carries all other network data from other users or applications. As these two paths share the same radio link, contention between them can degrade model transfer latency. This motivates the use of network slicing to isolate the two flows, which we evaluate in Section 4.5.2.

This instantiation demonstrates how RBF supports simulation-driven applications and emerging edge technologies. Critically, RBF transparently tolerates delayed and irregular timings of model updates, while enabling continuous inference through asynchronous and opportunistic model improvement.

4. Empirical Evaluation

We next use the Evaluation Facility and CUPS application to empirically evaluate RBF end-to-end. The application supplies concrete workloads and error measures, but the quantities we report (cadence, staleness, decision quality, and dissemination cost) are properties of the workflow system rather than of

the application domain. We demonstrate the effectiveness of RBF by presenting our results in four stages. First, we establish the operational cadence of our dedicated-access computational pipeline. Second, we show how model accuracy decays over time, detailing the effects of "backfilling" the production of inference models by the dedicated pipeline with opportunistic computations run at different sites. Third, we examine the value of a *single* opportunistic run and evaluate a decision-time heuristic that predicts, at the head of the batch queue, whether each one should proceed. Finally, we evaluate the deployment of the inference models onto edge devices, showing that they can be delivered and executed with minimal overhead.

4.1. Dedicated-Access Pipeline Performance

We refer to the dedicated cluster and GPUs in the Dedicated Cluster Tier as the "dedicated-access pipeline." In this study, the cluster consists of 72 machines, each of which executes a separate simulation in parallel. The results of this parallel execution phase are gathered, spatially referenced, and used to train three different surrogate models using two dedicated GPUs within the dedicated cluster. The operational flow is as follows: CFD simulations run in parallel, producing simulation data that is transferred to GPU nodes via the WAM; training jobs (PINN and FNO on separate GPUs, PCR on CPU) execute concurrently; trained models are returned via the WAM and deployed at the edge for use for inference.

On average across 15 runs, the dedicated-access pipeline completes in 134.8 ± 32.9 minutes. Of this total, the *sim* stage accounts for approximately 52 minutes for CFD computation (72 nodes) and 14 minutes for data transformation of simulation outputs (for use in model training). The *train* stage executes PINN, FNO, and PCR training in parallel on CPU/GPU resources and requires approximately 55 minutes in total. Individually, PINN training requires 50.0 ± 21.6 minutes, FNO requires 54.8 ± 18.2 minutes, and PCR requires 15.9 ± 3.4 minutes. The remaining time is due to data fetching from the edge, model transfer to the edge, logging, and other miscellaneous system overhead. Thus, a new inference model is available at the edge, on average, every 134.8 minutes. This end-to-end cadence represents the fastest achievable retraining cycle with dedicated resources, completely independent of HPC use and queue contention.

The ±32.9 minute standard deviation (24% of the mean) arises from two sources: (i) synchronization overheads during data staging and model transfer across the cluster, and (ii) variable training iteration counts when the 72 simulation cases produce different numbers of unique data points. Despite this inherent variability, the dedicated-access pipeline remains operationally predictable and quantifiable, enabling reliable operational planning.

4.2. Inference Model Accuracy Decay

This rate at which the dedicated-access pipeline can publish models for use by edge inference forms the baseline cadence of RBF. We next evaluate the value of these published models in terms of accuracy and investigate the rate at which they become stale.

Because the inference models are trained from simulations using "live" data and data immediately preceding it from the recent past, they are most accurate for the immediate future and their accuracy decays with time. In this study, the dedicated pipeline produces an inference model that is trained from data that is (on average) at least 134.8 minutes old – the data that was available when the simulations were initiated.

Given this 134.8-minute dedicated pipeline cadence, the critical questions are:

- What accuracy do the deployed inference models maintain when produced at the maximal cadence?
- Does retraining more frequently improve accuracy and, if so, by how much?

Figure 4 shows the decay of model accuracy over time for all three models (PINN, FNO, PCR). In the figure, the x-axis shows the time, in hours, since the model became available at the edge, and the y-axis shows the *cumulative* mean absolute error (MAE) in meters per second.

We obtain a single error sample by parameterizing the inference model with the current data and comparing the inference it makes to the next available measurement. The value plotted at elapsed time t is the mean of every such sample recorded from the origin through t, aggregated across the test points in the field. Thus, the figure depicts the "decay" in accuracy (measured as cumulative MAE) as a model is served over time. Because a model is published only when its pipeline has completed, the origin of the x-axis is the moment the model reaches the edge and therefore already accounts for the pipeline compute latency reported in Section 4.1; a model is never fresher than the one in service. Each curve aggregates samples drawn from selected operating periods between 2022 and 2026 rather than from a single continuous deployment window, so the curves describe typical behavior across seasons rather than one run.

Note that the CFD simulations all take the latest real-time sensor data and a short "history" of measurements immediately preceding it when they are initiated. In Figure 4 we show the accuracy decay for each model using a different color, depending on history length. These results show the opportunity for dynamic and automatic hyper-parameter tuning.

No single history length dominates. PINN is most accurate in the first hours after generation when trained with 6 hours (blue) or 24 hours (green) of history, but its curves cross near the 6-hour mark, after which 48 hours (red) is most accurate; FNO and PCR favor the 12- and 24-hour windows. We therefore select the history length for *sufficiency over the window a model actually serves* rather than for lowest error overall.

At the model-update cadences reported in Section 4.1, a deployed model is typically replaced within six hours. Over this interval, the four evaluated history lengths exhibit similar prediction accuracy, making longer histories unnecessary. We therefore use a 6-hour history for all experiments, as it provides comparable accuracy over the served lifetime of each model while minimizing training time. History length affects model training time—and consequently the maximum achievable pipeline cadence—but does not affect simulation time. Longer history

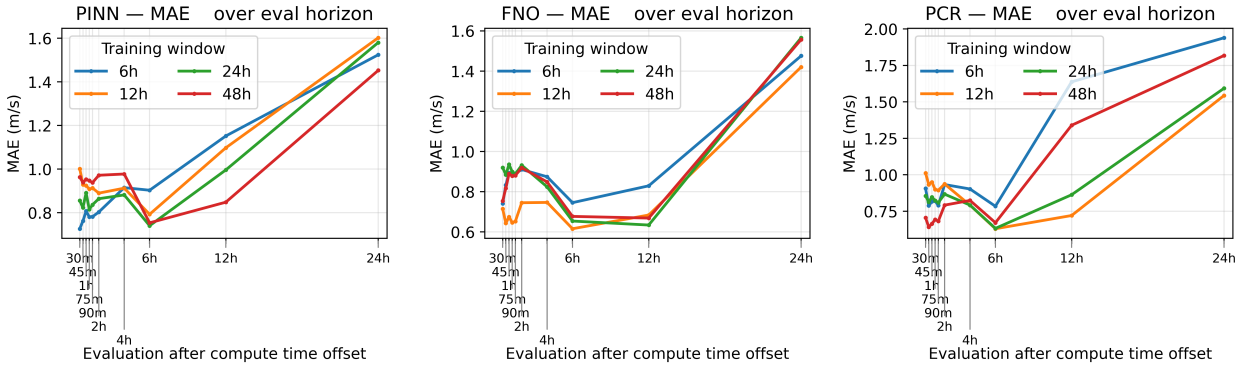

Figure 4: Model accuracy decay over time using different history windows for all three models (PINN, FNO, PCR). The x-axis shows elapsed time, ranging from 30 minutes to 24 hours, after a model becomes available at the edge; because a model is published only once its pipeline completes, the origin already includes the pipeline compute latency of Section 4.1. The y-axis shows *cumulative* MAE (m/s): the value plotted at elapsed time t aggregates every prediction error recorded from the origin through t, across the three sensor test locations in the field, into a single mean. Each subplot shows the decay curves for one model type.

windows become beneficial only when model updates are delayed well beyond the nominal cadence. Selecting the history length dynamically based on expected model lifetime is an interesting direction for future work.

4.3. Opportunistic Model Regeneration to Improve Delivery Cadence

The goal of augmenting the dedicated execution pipeline with additional simulation and training jobs submitted to one or more batch-controlled HPC systems is to reduce the time between successive model updates delivered to the edge. As illustrated in Figure 4, more frequent model publications reduce the period over which model accuracy decays. Each newly deployed model effectively resets the inference system to the point of maximum accuracy on the decay curve, thereby maintaining higher prediction accuracy over time.

Recall that, using dedicated resources, fresh models are available every 134.8 minutes (the *sim+train* time in Figure 3). Each additional generation, regardless of source, shifts the operational system's current point leftward on the decay curve, reducing model *staleness*. Because each additional model generation by the HPC resource reduces staleness in some measure (unless it coincides with a pipeline-generated model publication), submitting and executing HPC jobs to generate new models improves accuracy.

However, the magnitude of the benefit (i.e. the improvement in MAE) is difficult to predict, and each HPC execution carries a *cost* in time allocation apportioned to the user. The frequency and duration of HPC executions also affect queuing delays, since HPC administrators implement their own user priorities. The NERSC system, for example, uses a fair-share scheduler that balances occupancy between competing users, so a user frequently consuming a share of the resource has later jobs deprioritized. Thus, a useful opportunistic scheduler must accurately predict the cost-benefit tradeoff associated with each specific HPC submission. We investigate doing so in Section 4.4.

First though, we present an empirical analysis of the benefit from adding HPC jobs opportunistically without consideration of the scheduling costs associated with each job submission. Note that without the ability to predict batch queue delays, model generation events occur at times that are randomly distributed between the maximal cadence generations.

In our Evaluation Facility, new observations become available from the sensors every 5 minutes. Moreover, the measurement error for wind speed, reported by the sensor manufacturer [20], is between ±0.44 and ±0.88 meters per second. These physical parameters bound RBF's effectiveness: no improvement in accuracy decay is possible when a model is generated faster than every 5 minutes or the inference error is below 0.44 meters per second.

We validate this analysis with a live multi-site experiment. Figure 5 shows model publish events for PCR, PINN, and FNO on a shared time axis spanning the night of May 6–7, 2026. Blue circles denote dedicated pipeline cluster publish events and orange squares denote NERSC publish events. Dedicated cluster events appear at regular intervals that average approximately 135 minutes each across all model types; PCR events are offset from PINN and FNO because PCR is trained on CPUs (and not GPUs) and converges more quickly.

NERSC events (orange squares in the figure) are less frequent. In this experiment, RBF submits one batch job to the NERSC queue at a time (i.e. it does not "flood" the queue with jobs continuously). Once a job is selected from the queue, it runs simulations until its time limit expires, after which RBF submits a new job to the queue. This scheduling policy is intended to minimize RBF's impact on the NERSC queuing system. The fair-share scheduler does not consider queue submission occupancy so it is possible to submit jobs continuously, thereby occupying queue resources with jobs that will wait a long time before their fair-share is available. Instead, RBF assumes that once a job is scheduled, it will use the resources as efficiently as possible during its execution period. That is, without consideration of the cost-benefit tradeoff, RBF will consolidate multiple model productions into "back-to-back" model productions for as long as a job executes (rather than using separate queue submissions for each model).

During the test period, the average wait for a new allocation is approximately 17 hours, creating gaps of at least 18 hours during

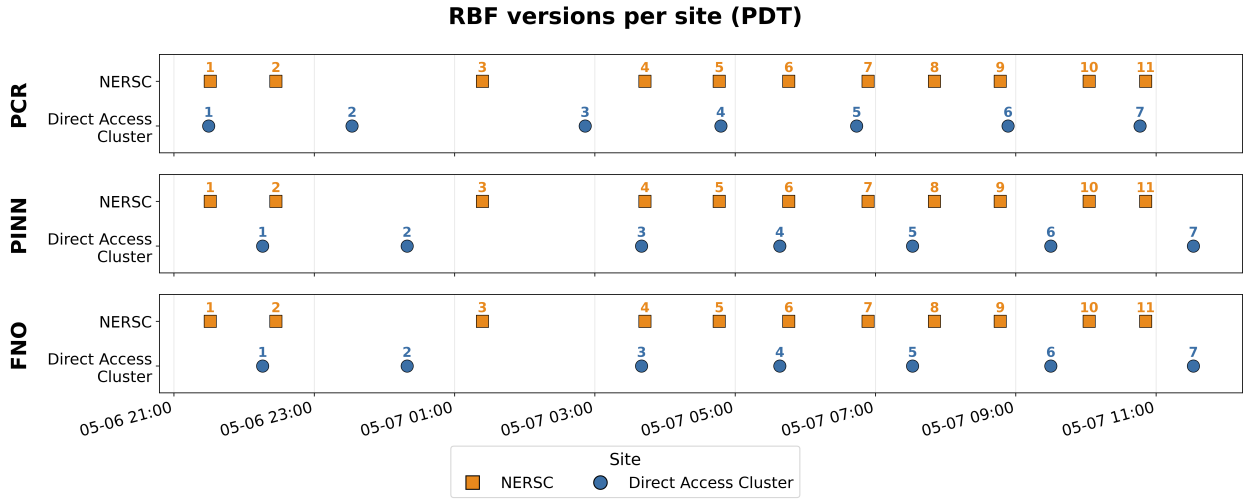

Figure 5: Timeline of model publish events for all three model types (PINN, FNO, PCR) during a simultaneous live experiment on two resources: the dedicated-access cluster and the NERSC batch-queue HPC.

Table 1: Minutes between FNO model publish events

Combination	Min	Avg	Max	Std
dedicated cluster	113.4	134.8	200.4	32.9
NERSC	47.9	80.0	176.5	40.4
dedicated cluster + NERSC	3.3	50.0	135.8	34.3

which no NERSC publish events occur. For these experiments, we observed queue wait times for the NERSC Perlmutter HPC system to be 17-19 hours for the 72-machine 32-CPU jobs and 11-38 minutes for the 2-GPU jobs [21]. However, as shown in Figure 5, when a NERSC job is scheduled, the back-to-back model generations occur more frequently compared to the dedicated pipeline cluster generations (orange squares between blue circles in the figure).

Together, dual tier use by RBF reduces the average time between consecutive publish events, keeping the effective model age below 2 hours. Doing so maintains a model accuracy for all models that is below the 0.88 m/s sensor measurement error bound. The longer cadence events for both resources (longer gaps between blue circles and/or orange squares in the figure) are due to wind condition differences, which lead to more variability in wind speed and consequently longer training windows.

Table 1 summarizes the interval between successive FNO model publications for the three execution configurations. The dedicated cluster alone produces a new model every 134.8 minutes on average, consistent with its approximately two-hour execution cadence. NERSC alone reduces the average publication interval to 80.0 minutes, although queue variability produces a wider distribution of completion times (40.4-minute standard deviation).

Combining the dedicated cluster with opportunistic NERSC execution further reduces the average interval between model publications to 50.0 minutes—a 2.7× improvement over the dedicated pipeline alone. The minimum publication interval falls to just 3.3 minutes, illustrating that independently executing workflow stages can produce closely spaced model updates whenever the two pipelines complete near one another. Much of this improvement is attributable to iterations 4–11, during which several NERSC jobs completed in rapid succession after entering execution around 3:30 AM on May 7.

More importantly, these results demonstrate the benefit of treating HPC as an asynchronous workflow stage rather than a synchronized execution resource. Each completed model publication reduces the time that deployed models remain stale, allowing edge inference to operate closer to peak accuracy by shortening the interval over which model accuracy decays. As shown in Figure 4, increasing the publication rate therefore directly improves the expected accuracy of the deployed surrogate models.

Because models may complete out of submission order across independent execution tiers, the edge deploys a newly published model only if its training cutoff time is newer than that of the currently deployed model. This lightweight version check preserves workflow correctness while allowing all execution tiers to operate asynchronously.

4.4. Predicting the Benefit-Cost Tradeoff

Figure 5 and Table 1 from the previous subsection illustrate the potential benefit of back-to-back executions on the faster NERSC resource once it begins executing a RBF submission. Each opportunistic run is, however, *costly*: it consumes a scarce shared-HPC allocation and, on a fair-share batch system, decreases the scheduling priority of later jobs. Further, at the granularity of a *single* input the benefit is no longer guaranteed—an individual run does not always improve the accuracy of a specific model already in service over the window it would serve (although the average improvement over time is positive).

To determine whether an individual submission should incur the cost of execution, we investigate two heuristics: one that releases only jobs predicted to improve on the most recently published model, and one that targets large improvements at

the expense of a few "false positive" predictions. For the experiments reported here, each heuristic makes its decision at the same point in a job's lifecycle. Once the batch scheduler selects an RBF submission and it begins executing, the job first contacts the RBF orchestrator, which, using up-to-date data gathered from the sensors in real time, determines whether the job should proceed to produce a new model publication or exit immediately. A job that exits at this point releases its allocation before any simulation or training compute is expended, thereby avoiding a "charge" against the user's allocation and the concomitant fair-share penalty.

More specifically, RBF predicts the value of each *isolated* opportunistic run. Within the dedicated cluster's constant publication cadence (Section 4.1), each opportunistic HPC job is treated as a single publication that could possibly improve the accuracy of the edge model from that moment forward. Although this study measures job utility using prediction accuracy, this approach is agnostic to the specific utility metric. It requires only that the workflow estimate the expected benefit of an updated model using observations already available when the job reaches the head of the queue. This assumption is satisfied by a broad class of simulation-driven workflows.

Note that RBF does not attempt to predict queue wait times or choose *when* to submit a job. Instead, it makes a decision when a submitted job reaches the head of the queue. At that moment the system either *releases* the job to proceed and complete its execution which will result in a new model publication, or *cancels* it, thereby leaving the last published model in place at the edge. The time spent before the decision point (queue residency) is not charged by the NERSC batch scheduler to the user allocation.

To empirically evaluate this approach and these heuristics we perform *back-testing*. We replay a recorded window of the deployment in which opportunistic HPC jobs executed at the times chosen by the batch scheduler. Unlike the results shown in Subsection 4.3, however, each HPC job executes a single iteration of the model-production pipeline (rather than back-to-back iterations) thereby incurring only the fair-share penalty for that one iteration. A job was "profitable" if the model it published improved on the model available when it began executing, and unprofitable otherwise. We therefore generate the edge-model error for *both* cases—proceeding with the execution and canceling it—so that the quality of each decision can be scored. Note that without back-testing, the effects of a cancellation could not be observed at all.

During the replay interval, the HPC batch scheduler selected 19 jobs for execution. We evaluate the decisions that the heuristics would have made for these 19 jobs. We chose this window because it is unusually dense in opportunistic publications—the shared facility published at nearly the rate of the dedicated cluster—and because most of its isolated runs turn out not to have helped, making it a demanding test of whether the useful runs can be distinguished.

As a baseline, allowing each of the HPC jobs to proceed unconditionally (i.e. without a decision by the heuristic) averages **−27.9%** improvement per model published by each HPC job when compared to the models produced by the dedicated pipeline. Eight of the 19 models were more accurate and 11 were less accurate. We also observe severe losses for a small number (two iterations account for nearly two-thirds of the total) which overwhelm the many modest gains. This contrast is not a contradiction: continuous back-to-back publication still lowers staleness in aggregate, but it shows that the marginal value of one additional, isolated run is not assured. This analysis shows that per-run decision-making is a worthwhile study: avoiding a small number of specific runs recovers most of the avoidable loss.

4.4.1. RBF *Conditional HPC Execution Decisions*

To predict the value of an individual submission, RBF implements an automatic model selection heuristic. Each pipeline trains all three model types (PINN, FNO, PCR) every iteration, and chooses a single *winner* model based on MAE computed from the most recently available data from the edge. There is a winner among the dedicated pipeline's most recent models and one among the HPC system's. Because the input data has been updated since the HPC submission, RBF computes the dedicated winner at the decision point using the latest data available; the HPC winner is the one that won the previous opportunistic iteration. The winner, in each case, is taken to be the lowest-error model type.

The RBF heuristic computes the *benefit* of an opportunistic model generation on the HPC resource as the percentage improvement of its winner over the dedicated winner across the interval it would serve, $[t_{\text{opp}}, t_{\text{ded, next}})$:

$$\%\text{improve} = 100 \times \frac{\text{MAE}_{\text{ded}} - \text{MAE}_{\text{opp}}}{\text{MAE}_{\text{ded}}} \tag{1}$$

Positive means the opportunistic model outperformed the dedicated-pipeline winner; negative means the run made the model published to the edge worse.

This prediction assumes that the error observed just before the decision is a good estimate of what each winner will produce over the window in question, justified by the model's *decay curve* (Section 4.2). Its error as a function of staleness which, over a window this short, is typically flat. Comparing the two winners in terms of recent error yields a predicted %improvement.

4.4.2. Heuristics

We evaluated many candidate rules over the study window; two fit this system best and bracket the operating range. Both act only on decision-time information.

The No-Regret heuristic, given in Algorithm 1, allows a job that is ready for execution on the HPC resource to continue only when the predicted %improvement is positive *and* the predicted opportunistic winner is a *different* model type than the current dedicated winner. The intuition behind the success of this heuristic is that it is predicting a model improvement based on a change in model type which indicates that the current conditions (at the decision point) have changed since the HPC job was submitted.

In the example period under study, this heuristic resulted in **no HPC executions that caused a negative improvement**. That is, when RBF decided to allow a submission to continue

Algorithm 1 The No-Regret heuristic, computing the head-of-queue decision for opportunistic iteration n. Both error vectors are measured over elapsed windows—u since the last dedicated publication, p over the window iteration $n-1$ was served—no quantity depends on the candidate model.

Require: $\mathcal{T} = \{\text{FNO}, \text{PINN}, \text{PCR}\}$: model types
Require: $\text{ded}_n[\tau]$: published dedicated model of type τ
Require: W: sensor data observed between the last dedicated publication and the decision point
Require: $p[\tau]$: error the type-τ model of the last completed opportunistic iteration achieved over the window it served; already measured, since the candidate opp_n does not exist yet
Ensure: RELEASE or CANCEL
1: **for** $\tau \in \mathcal{T}$ **do**
2: $\quad u[\tau] \leftarrow \text{MAE}(\text{ded}_n[\tau], W)$
3: **end for**
4: $w_{\text{ded}} \leftarrow \arg\min_{\tau \in \mathcal{T}} u[\tau]$
5: $U \leftarrow u[w_{\text{ded}}]$ {incumbent error}
6: $w_{\text{opp}} \leftarrow \arg\min_{\tau \in \mathcal{T}} p[\tau]$ {winner of iteration $n-1$'s own outcome}
7: $\hat{g} \leftarrow 100\,(U - p[w_{\text{opp}}])\,/\,U$ {predicted %improve}
8: $shift \leftarrow (w_{\text{opp}} \neq w_{\text{ded}})$
9: **if** *shift* **and** $\hat{g} > 0$ **then**
10: $\quad$ **return** RELEASE
11: **else**
12: $\quad$ **return** CANCEL
13: **end if**

Algorithm 2 The High-Yield heuristic, which reuses the decision-time quantities of Algorithm 1: the shift branch is No-Regret's own condition, whereas the no-shift branch tests only inter-type disagreement.

Require: lines 1–8 of Algorithm 1, yielding p, U, $\hat{g}$, and *shift*
Require: $\theta = 125$: spread threshold, in percentage points
Ensure: RELEASE or CANCEL
1: **if** *shift* **then**
2: $\quad$ **return** RELEASE if $\hat{g} > 0$, otherwise CANCEL (No-Regret release)
3: **else**
4: $\quad spread \leftarrow \max_{\tau \in \mathcal{T}} 100\,(U - p[\tau])/U - \min_{\tau \in \mathcal{T}} 100\,(U - p[\tau])/U$
5: $\quad$ **return** RELEASE if $spread > \theta$, otherwise CANCEL
6: **end if**

execution, it always yielded an improvement in accuracy (not just in aggregate). In this study, the No-Regret heuristic decided to continue 3 of the 19 total submissions which generated an average improvement of +18.2% and an overall improvement of +54.6% in terms of MAE. From the user's perspective, No-Regret ensures that every submission charged to an allocation yields an improvement, and that avoiding a false positive keeps a model that would lower accuracy from ever reaching the edge.

The High-Yield heuristic relaxes this guarantee in exchange for capturing more total improvement. Because No-Regret avoids individual false positives, it may be overly conservative in aggregate: accepting a few degrading releases can also admit runs that raise overall accuracy.

The High-Yield heuristic, given in Algorithm 2, uses the No-Regret heuristic to determine whether a job should proceed or cancel at the decision point. If No-Regret indicates cancel, and there is not a change in model type between the winners evaluated by No-Regret, High-Yield then computes the degree of disagreement between the three model types in terms of their predicted %improvement. Disagreement is measured as the spread between the highest and lowest of the three estimates, and the HPC job is allowed to proceed when that spread exceeds a configuration-time threshold (set to 125 percentage points in this study).

Table 2: Performance of No-Regret and High-Yield heuristics over the 19 isolated opportunistic publications—each inserted at a randomized position within the dedicated cluster's constant cadence—against an unconditional-execution baseline. Totals are taken over all 19 isolated runs, whereas averages are per individual execution; for the unconditional executions all 19 HPC jobs are allowed to proceed. "Compute avoided" is the savings in terms of HPC core-hours by each heuristic and "degrading releases" counts HPC executions that ultimately resulted in negative improvements.

	Uncond.	**No-Regret**	**High-Yield**
Released / 19	19	3	8
Total %improve	−529%	+**54.6%**	+**71.1%**
Avg %improve / released run	−27.9%	+**18.2%**	+8.9%
Degrading releases	11	**0**	2
Compute avoided (core-h)	—	~83,000	~57,000

The rationale for this heuristic is that a single strongly positive model-type estimate standing far apart from negative peers is more likely to reflect a real gain than a consensus clustered near zero. This captures more total improvement—+**71.1% in total** allowing 8 of the 19 HPC submissions to proceed. However, the average individual improvement is +8.9% due to two models that were allowed to proceed despite an unfavorable aggregate prediction and, ultimately, generated negative improvements (i.e. two false-positive releases).

RBF exposes the heuristic algorithm to use and the disagreement threshold for the High-Yield algorithm as configuration options to the user. Table 2 summarizes these results.

Each canceled submission avoids the cost of an entire opportunistic HPC pipeline execution. An individual run reserves 72 compute nodes (2,304 cores) for approximately one dedicated pipeline iteration (134.8 minutes; Section 4.1), consuming roughly 5,180 core-hours. Launching all 19 opportunistic submissions unconditionally therefore requires nearly 98,400 core-hours.

The No-Regret heuristic launches only 3 of the 19 candidate jobs, canceling 16 and avoiding approximately **83,000 core-hours** of HPC computation while never degrading the accuracy of the deployed model. The High-Yield heuristic launches 8 jobs and cancels 11, reducing HPC consumption by approximately **57,000 core-hours**. Because the dedicated execution pipeline continues to produce models throughout, these canceled HPC executions do not reduce served accuracy. Instead, the heuristics preferentially eliminate simulations that, by construction, would have provided little or no improvement to the deployed model.

Beyond the measured core-hour savings, these results have two broader implications. First, the reported savings are conservative. On a fair-share HPC system, every canceled job also preserves the project's scheduling priority, increasing the likelihood that subsequent submissions begin execution sooner. Second, because this analysis evaluates the value of an individual job independent of any particular queuing behavior, the approach is not tied to a specific HPC facility or scheduler and should generalize to other batch-managed systems.

4.5. Edge Tier Performance

The benefits of increasing model publication cadence are realized only if updated surrogate models can be disseminated to edge devices and deployed with sufficiently low latency. Artifact dissemination and edge inference therefore form the final stage of the end-to-end workflow, bounding how quickly improvements generated by upstream simulation and training become available for operational inference.

This section evaluates these final workflow stages. First, we measure model publication latency to Raspberry Pi edge devices deployed at the CUPS facility using commercial off-the-shelf private 5G infrastructure and the satellite backhaul available during Spring 2026. Second, to evaluate the potential of next-generation edge networking, we measure model dissemination latency using a laboratory private 5G deployment based on the srsRAN Project [18] radio access network and Open5GS [19]. Finally, we evaluate surrogate-model inference performance on the edge devices, demonstrating that the deployed models provide sufficiently low-latency inference for continuous operation on resource-constrained hardware.

4.5.1. Satellite Long-Haul and Cellular 5G at CUPS

When a new model is published via the WAN, edge-inference devices fetch it and update the locally available model. Each surrogate model traverses a Starlink [17] satellite Internet long-haul network connecting to a 5G cellular link at the CUPS facility. The facility is equipped with a Celona 5G cell, model AP25-48 [16], which is an off-the-shelf commercial solution for 5G private cellular networking. Cellular 5G provides wide network coverage and high throughput ideal for the large scale agricultural sensor network present at the CUPS facility.

Two Raspberry Pi 4's sit at the CUPS facility. The first Raspberry Pi (termed henceforth cellular-RPi) is connected to a Celona Cradlepoint (a mobile hotspot), which has a programmable SIM provisioned to connect to the Celona AP. This Raspberry Pi provides continual predictions of the airflow across the screenhouse and fetches models using the WAM. The second Raspberry Pi is connected directly to the CUPS backhaul of Starlink, avoiding cellular networking entirely. This second Raspberry Pi serves as the control (termed henceforth control-RPi) allowing for the effects of cellular to be isolated from Starlink.

To test network performance of model transport to the edge, we consider two conditions: an empty versus a loaded network (load introduced with `iperf3`), and a path with and without the Starlink backhaul. The Starlink-cellular path is the standard HPC-to-edge delivery route. To enable this, we run four experiments. All model transport is initiated and targets the cellular-RPi, the site for inference.

- *Cellular Only* - Model retrieval from a WAM host running on the control-RPi
- *Cellular w Contention* - The above plus `iperf3` network saturation on cellular link.
- *Starlink + Cellular* - Model retrieval from the remote WAM host to get model updates from HPC
- *Starlink + Cellular w Contention* - The above plus `iperf3` network saturation from cellular-RPi to dedicated cloud VM with 10 Gbps network capacity.

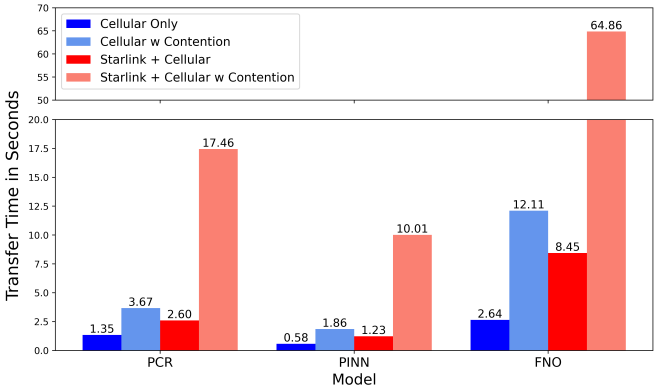

Figure 6: P-95 Model Transfer Time of 100 runs. Transfer time is in seconds. Each bar represents a different network path (cell only or Starlink and cellular) for model delivery. Note the gap between 20 and 50 seconds only contains the FNO bar.

Figure 6 compares the model transport times across the four experiments. All numbers represent P-95 values for worst-case tail latency. Model transfer times correspond to model size. PINN models are the fastest due to their small size (290 KiB). FNO is the largest model (9.1 MiB) and takes the most time to transfer. PCR sits in between, being only 1.07 MiB. Without contention, using the Starlink path introduces a penalty for all models, requiring 2-3× more time. With network contention, the backhaul connection via Starlink becomes the bottleneck rather than the cellular network. For the cellular-only transfer, contention increases download time for all models by a factor of about 2.7–4.5×. However, with Starlink, contention results in an increase of approximately 6-8× compared to Starlink without contention.

Despite the large increase in model transport latency when the network is under stress, all three models transfer within 65 seconds (P-95), well within the required time for the CUPS application. Thus, polling for a new model every 60 seconds results in a model update (when a new model has been published) no more than 65 seconds later, with confidence 0.95. As described in Section 4.3, sensor data updates every 5 minutes so model delivery of a "fresher" model falls well within the data sampling interval. Therefore, with an optimal model publishing cadence of five minutes and the asynchrony of model retrieval, model transfer has a budget of 150 seconds to ensure the retrieval is of the latest model. As all three models transfer to the edge RPi at the CUPS facility with worst-case latency of under 65 seconds, satellite and standard 5G cellular networking are sufficient to ensure up to date models. The binding quantity is artifact size against link capacity, so this bound carries over to any workflow disseminating comparable artifacts over similar long-haul links.

Table 3: Model download throughput (MB/s) with and without slicing. Values are means over 100 runs. Degradation is the percentage change under contention.

	No Slicing			Slicing		
Model	**Iso.**	**Cont.**	**Deg.**	**Iso.**	**Cont.**	**Deg.**
PCR	2.68	2.15	−20%	2.67	2.50	−6%
PINN	1.37	1.06	−23%	1.28	1.31	+2%
FNO	4.92	3.88	−21%	4.72	4.62	−2%

4.5.2. Laboratory 5G Network Slicing Evaluation

To investigate the potential of network slicing, we evaluate the use of a private 5G cell in the laboratory. The cell implements the OpenRAN [22] standard. To test whether slicing shields model transfer during network load, we configure two slices. The first is an enhanced mobile broadband (eMBB) slice spanning the entire physical radio spectrum, which is broken down into Physical Resource Blocks (PRBs). The second slice uses between 0 and 10% of available PRBs.

In our experiment, a Raspberry Pi 4 repeatedly fetches the PCR, PINN, and FNO models from a remote WAM host. This Raspberry Pi is attached to the large eMBB slice. A second Raspberry Pi 4 serves as the network load generator. This Raspberry Pi is attached to the capped slice and generates contention on the radio link by saturating the RAN with UDP traffic solely between the 5G cell host and itself.

Similar to the CUPS experiment, we measure throughput with and without contention and with and without slicing to determine the benefit of network slicing. Table 3 shows the mean throughput measured over 100 runs. As in the previous results, throughput follows model sizes. Without slicing, contention results in a throughput cut by 20−23%. With slicing, this throughput cut is reduced to 6% for PCR, and 2% for PINN and FNO. We believe that the 6% for PCR is negligible and can be attributed to noise factors outside the control of network slicing, as both a larger model (FNO) and a smaller model (PINN) were within the 2% difference.

One possibility for this could be external interference or slight radio propagation changes that are outside the control of radio slicing. The contending Raspberry Pi's slice cap prevents it from consuming resources beyond its allocation. Also, because PRBs are not permanently attached to an associated eMBB, slicing does not incur a significant penalty in the isolated case. Comparing isolated throughput with and without slicing, isolated throughput is only observed to drop between 0.4−6.6% with slicing active, with PINN having the highest gap of 6.6%). Overall, we find that although current commercial off-the-shelf 5G network infrastructure is sufficient for edge model retrieval due to their small size, promising open-source private 5G networks provide additional network isolation capabilities that lower maximum-bound transfer latencies for surrogate models.

4.5.3. Cost of Inference on Edge

To ensure near real-time inference on the edge, the surrogate models selected must perform in near real-time on constrained hardware. We next evaluate the inference performance of these three models on the edge (a Raspberry Pi 4).

Table 4: Execution time (in seconds) for a inference on all points within CUPS using a Raspberry Pi 4. We report the P95 across 100 trials.

Model	**RPi 4 Wall Clock (s)**
PCR Model	
Initialization	25.554
Inference	0.003
PINN Model	
Initialization	0.698
Inference	0.408
FNO Model	
Initialization	3.779
Inference	1.100

We isolate edge inference into initialization and model execution. Initialization time captures the model cold-start costs, including disk I/O and software library initialization. Between each trial the caches are cleared to minimize effects between trials. To profile model execution, we capture the wall clock time required to evaluate a specific query. Running multiple queries on the same model does not require reinitialization. For our inference query, we run a prediction of airflow at all points enclosed by the CUPS structure. This consists of 9690 points in three-dimensional space.

The P-95 wall-clock times (in seconds) per model are presented in Table 4. We show that initialization for all models takes longer than a single inference, sometimes orders of magnitude longer (8518× for PCR). In PCR's case, each data point is loaded from a separate CSV file from disk, resulting in 9690 CSV file loads to memory. However, once loaded, its inference time is 135 times faster than PINN, and 365 times faster than FNO. FNO and PINN's initialization step involves loading the TensorFlow model object and initializing software libraries. FNO's model involves more parameters, resulting in a larger initialization and inference time than PINN.

Each surrogate model allows for fast airflow prediction at the edge across the entirety of the CUPS structure, with inferences requiring between 3 ms (PCR) to 1.1 seconds (FNO). This cost is set by surrogate complexity rather than by the application, so these figures characterize any workflow serving a comparably sized surrogate. As for initialization costs, model initialization occurs only on each model update. Combining with the results from the network transfer experiments, even with the worst case 25 seconds for P-95 PCR initialization, RBF inference time is still well below our target 5 minute threshold.

Overall, these results indicate that system performance is dominated by simulation and training latency, and that networking and edge inference do not limit end-to-end performance. This reinforces the design focus of RBF on asynchronous, simulation-driven model improvement, where reducing model staleness through opportunistic HPC execution provides the primary gains. The results also demonstrate that RBF facilitates the integration and evaluation of diverse surrogate models and networking configurations, enabling flexible experimentation across model types and communication environments without impacting system operation.

Several limitations bound these findings. They characterize a single deployment—one screenhouse, one dedicated cluster,

one shared HPC facility—so absolute magnitudes should not be read as general. The RBF heuristic study rests on 19 scorable iterations from one window, with captured totals concentrated in a few runs and its threshold fitted on that same window; it is evidence that an actionable decision-time signal exists, not a validated production policy. Finally, model error over the operating region is comparable to the sensor measurement error bound above, so small differences between model types and history lengths should be read with that floor in mind.

5. Related Work and Positioning

Early versions of the RBF concept, including the online–offline learning and inference loop, were explored in prior workshop work [23]. This paper substantially extends that effort by formalizing the workflow architecture, introducing Reverse Backfill for asynchronous HPC integration, and providing a complete implementation and experimental evaluation.

Scientific workflow systems are the established means of composing and executing multi-stage computations on shared infrastructure. Pegasus [24], Parsl [25], Swift/T [26], FireWorks [27], Balsam [28], and RADICAL-Pilot [12] provide abstractions for workflow specification, task scheduling, resource management, and fault recovery across clusters, clouds, and HPC systems. RBF addresses a complementary challenge: continuously executing simulation-driven workflows whose inputs arrive as streaming sensor observations and whose outputs must provide continuous low-latency inference while surrogate models evolve asynchronously through simulation and training. Rather than replacing these systems, RBF provides workflow semantics for persistent workflow state, evolving workflow artifacts, and decision-time execution control that could be implemented on top of many existing workflow engines.

A parallel line of work studies applications spanning the computing continuum, including edge, cloud, and HPC resources [29, 30]. Related systems such as Ray [31], InferLine [32], Jellyfish [33], EC5 [34], Neurosurgeon [35], Edgent [36], JointDNN [37], and Loconte et al. [38] study distributed inference, resource allocation, execution placement, and model serving across heterogeneous resources. These systems primarily optimize where computation executes or how inference pipelines are provisioned, whereas RBF focuses on when simulation-driven workflow stages should execute and how asynchronously generated models are incorporated into a continuously executing workflow.

Simulation-driven learning combines high-fidelity numerical simulation with surrogate modeling techniques including PINNs, Fourier Neural Operators, and DeepONets [7, 9, 39], and has been applied extensively to computational fluid dynamics and related scientific domains [40, 41]. Recent work in scientific machine learning and HPC-integrated training [42, 43] has significantly improved surrogate accuracy and expressiveness, while continual and streaming learning methods assume that models can be updated directly from incoming data [44, 45]. In contrast, RBF addresses simulation-driven workflows whose model evolution depends on expensive, batch-scheduled computation, requiring explicit management of delayed, asynchronous model updates across the computing continuum.

Recent systems have begun integrating AI workflows with HPC execution. Balin et al. [46] demonstrate asynchronous in-situ surrogate training coupled to CFD simulations, while VESTEC [47] submits urgent HPC simulations in response to live observations. Brewer et al. [48] classify AI–HPC execution motifs, and ROSE [49] explores efficient surrogate training through active learning. RBF complements these efforts by treating HPC as an asynchronous model-improvement engine within a continuously executing scientific workflow. Persistent workflow state enables simulation, training, deployment, and inference to proceed independently while incorporating improved surrogate models as they become available. The system additionally leverages private 5G networking for QoS-isolated sensor telemetry and model dissemination [50, 51, 52, 53, 54].

6. Conclusions

We presented RBF, a hybrid edge–HPC architecture for simulation-driven training of fast surrogate AI models that execute at the edge based on real-time streaming data. By decoupling inference from simulation and training, RBF enables continuous, low-latency inference while incorporating improved models asynchronously as they become available.

Although simulation latencies are large, our results show that these latencies can be masked and that networking and edge inference contributing negligible delays so that high-quality inferences are continuously available from edge devices. Further, the RBF prototype demonstrates that HPC resources can be effectively integrated into operational systems as asynchronous model-improvement engines, enabling practical deployment of simulation-driven learning, with an average of 2.7× improvement of surrogate model accuracy. We further show that this aggregate benefit does not extend to every *individual* HPC execution, and that an execution-time heuristic (applied when a job is selected for execution in a batch queue) can predict whether the subsequent execution on the HPC resource will improve model accuracy.

RBF also provides a flexible platform for evaluating diverse models Evaluated using a digital agriculture deployment, RBF is general with respect to other application domains. It demonstrates a reusable workflow architecture for continuously executing, simulation-driven scientific workflows spanning edge devices, dedicated clusters, and shared HPC resources. Persistent workflow state enables loosely coupled workflow stages to exchange evolving workflow artifacts asynchronously, while Reverse Backfill provides an efficient mechanism for incorporating opportunistically available HPC resources into long-running workflows without requiring changes to existing batch schedulers. We believe these architectural principles are applicable to a broad class of computing-continuum applications in which high-fidelity simulation continuously improves operational inference.

Upon publication, we plan to make our code and data available as open source and data. This research is supported in part by

NSF awards CNS-2444318 and CNS-2107101, and DoE award DE-SC0025541

References

[1] J. Bove, Huanglongbing: A Destructive, Newly-emerging, Century-old Disease of Citrus, Journal of Plant Pathology 88 (1) (2006).

[2] P. E. Rolshausen, G. Barry, A. Schumann, Citrus Undercover Production System (CUPS), Citrograph 10 (1) (2019).

[3] A. W. Schumann, A. Singerman, M. A. Ritenour, J. Qureshi, F. Alferez, 2025–2026 Florida Citrus Production Guide: Citrus under Protective Screen (CUPS) Production Systems, critical Issue: 1. Agricultural and Horticultural Enterprises (2025). doi:https://doi.org/10.32473/edis-hs1304-2023.

[4] OpenFOAM — openfoam.com, https://www.openfoam.com/, [Accessed 26-08-2024] (2024).

[5] O. , Openfoam: User guide: snappyhexmesh (01 2020).
URL https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh.html

[6] Openfoam.com, Openfoam: Api guide: poroussimplefoam directory reference (2026).
URL https://www.openfoam.com/documentation/guides/v2012/api/dir_03decea705bc0c0c22f85f91452a296f.html

[7] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics 378 (2019) 686–707.

[8] S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (pinns) for fluid mechanics: A review, Acta Mechanica Sinica 37 (12) (2021) 1727–1738.

[9] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).

[10] I. T. Jolliffe, A note on the use of principal components in regression, Journal of the Royal Statistical Society. Series C (Applied Statistics) 31 (3) (1982) 300–303.
URL http://www.jstor.org/stable/2348005

[11] NERSC, Perlmutter System, https://www.nersc.gov/what-we-do/computing-for-science/perlmutter, [Online; accessed 29-July-2026] (2024).

[12] A. Merzky, M. Turilli, M. Titov, A. Al-Saadi, S. Jha, Design and performance characterization of radical-pilot on leadership-class platforms, IEEE Transactions on Parallel and Distributed Systems 33 (4) (2022). doi:10.1109/TPDS.2021.3105994.

[13] Raspberry Pi, https://www.raspberrypi.org, [Online; accessed 15-Nov-2025] (2025).

[14] R. Wolski, C. Krintz, F. Bakir, G. George, W.-T. Lin, CSPOT: Portable, Multi-scale Functions-as-a-Service for IoT, in: ACM Symposium on Edge Computing, 2019, pp. 1–14.

[15] F. Bakir, C. Krintz, R. Wolski, CAPLets: Resource Aware, Capability-Based Access Control for IoT, in: ACM/IEEE Symposium on Edge Computing, 2021, pp. 106–120.

[16] Celona Orchestrator (2026).
URL https://www.celona.io/orchestrator

[17] Starlink: Reliable High-Speed Internet from Space. (2026).
URL https://starlink.com/

[18] srsRAN Project, srsRAN Project, https://www.srsran.com, [Online; accessed 15-Nov-2025] (2025).

[19] Open5GS, Open5GS, https://open5gs.org/, [Online; accessed 15-Nov-2025] (2025).

[20] Davis, Vantage Pro2 — davisinstruments.com, https://www.davisinstruments.com/collections/vantage-pro2?srsltid=AfmBOoqaY0U6K2F-X3xuWTlRZ-3yE6LfvYtnmDXRyaIeQimtSL6bPQqQ, [Accessed 23-08-2024] (2024).

[21] NERSC, Perlmutter HPC Queue Wait Times (2026).
URL https://rest.nersc.gov/rest/mynersc/frames/queuewaittimes.html

[22] O-RAN Alliance (2026).
URL https://www.o-ran.org/

[23] Blinded for Review, Blinded for Review (2026).

[24] Pegasus for computational workflows, https://pegasus.isi.edu, [Online; accessed 11-July-2024] (2024).

[25] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, K. Chard, Parsl: Pervasive Parallel Programming in Python, in: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC), 2019, pp. 25–36. doi:10.1145/3307681.3325400.

[26] Swift/T High Performance Dataflow Computing, http://swift-lang.org/Swift-T/, [Online; accessed 15-Nov-2016] (2016).

[27] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, K. A. Persson, FireWorks: a dynamic workflow system designed for high-throughput applications, Concurrency and Computation: Practice and Experience 27 (17) (2015) 5037–5059. doi:10.1002/cpe.3505.

[28] M. A. Salim, T. D. Uram, J. T. Childers, P. Balaprakash, V. Vishwanath, M. E. Papka, Balsam: Automated Scheduling and Execution of Dynamic, Data-Intensive HPC Workflows, arXiv preprint arXiv:1909.08704, presented at the 8th Workshop on Python for High-Performance and Scientific Computing (PyHPC), SC18 (2018). doi:10.48550/arXiv.1909.08704.

[29] S. Santillan, C. L. Abad, An analysis of hpc and edge architectures in the cloud (2025). arXiv:2508.01494. URL https://arxiv.org/abs/2508.01494

[30] F. P. Knebel, R. Trevisan, G. S. do Nascimento, M. Abel, J. A. Wickboldt, A study on cloud and edge computing for the implementation of digital twins in the oil & gas industries, Computers & Industrial Engineering 182 (2023) 109363. doi:https://doi.org/10.1016/j.cie.2023.109363.

[31] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, I. Stoica, Ray: a distributed framework for emerging ai applications, in: USENIX Conference on Operating Systems Design and Implementation, 2018, pp. 561–577.

[32] D. Crankshaw, G.-E. Sela, X. Mo, C. Zumar, I. Stoica, J. Gonzalez, A. Tumanov, Inferline: latency-aware provisioning and scaling for prediction serving pipelines, in: Symposium on Cloud Computing, 2020, pp. 477–491. doi:10.1145/3419111.3421285.

[33] V. Nigade, P. Bauszat, H. Bal, L. Wang, Inference serving with end-to-end latency slos over dynamic edge networks, Real-Time Systems 60 (2024). doi:https://doi.org/10.1007/s11241-024-09418-4.

[34] J. Tan, F. Liu, B. Wang, Q. Wu, C. P. Chen, Ec5: Edge–cloud collaborative computing framework with compressive communication, Future Generation Computer Systems 166 (2025). doi:https://doi.org/10.1016/j.future.2025.107715.

[35] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, L. Tang, Neurosurgeon: Collaborative intelligence between the cloud and mobile edge, Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (Apr. 2017). doi:10.1145/3093337.3037698.

[36] E. Li, Z. Zhou, X. Chen, Edge intelligence: On-demand deep learning model co-inference with device-edge synergy, in: Workshop on Mobile Edge Communications, 2018, pp. 31–36. doi:10.1145/3229556.3229562.

[37] A. E. Eshratifar, M. S. Abrishami, M. Pedram, Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services, IEEE Transactions on Mobile Computing 20 (2) (2021). doi:10.1109/TMC.2019.2947893.

[38] D. Loconte, F. Ieva, L. Pinto, G. Loseto, F. Scioscia, M. Ruta, Expanding the cloud-to-edge continuum to the IoT in serverless federated learning, Future Generation Computer Systems 155 (2024). doi:https://doi.org/10.1016/j.future.2024.02.024.

[39] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (2021).

[40] A. Marcato, G. Boccardo, D. Marchisio, A computational workflow to study particle transport and filtration in porous media: Coupling cfd and deep learning, Chemical Engineering Journal 417 (2021). doi:https://doi.org/10.1016/j.cej.2021.128936.

[41] T. Botarelli, M. Fanfani, P. Nesi, L. Pinelli, Using physics-informed neural networks for solving navier-stokes equations in fluid dynamic complex scenarios, Engineering Applications of Artificial Intelligence 148 (2025). doi:https://doi.org/10.1016/j.engappai.2025.110347.

[42] B. Van Essen, H. Kim, R. Pearce, K. Boakye, B. Chen, Lbann: livermore big artificial neural network hpc toolkit, in: Workshop on Machine Learning in High-Performance Computing Environments, 2015, pp. 5:1–5:6. doi:10.1145/2834892.2834897.

[43] ExaLearn: US Department of Energy (DOE) Exascale Computing Project (ECP) center (2020). URL https://acdc.alcf.anl.gov/exalearn/

[44] L. Wang, X. Zhang, H. Su, J. Zhu, A comprehensive survey of continual learning: Theory, method and application, IEEE Transactions on Pattern Analysis and Machine Intelligence 46 (8) (2024) 5362–5383. doi:10.1109/TPAMI.2024.3367329.

[45] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, J. Gama, Machine learning for streaming data: state of the art, challenges, and opportunities, ACM SIGKDD Explorations Newsletter 21 (2) (2019) 6–22. doi:10.1145/3373464.3373470.

[46] R. Balin, F. Simini, C. Simpson, A. Shao, A. Rigazzi, M. Ellis, S. Becker, A. Doostan, J. A. Evans, K. E. Jansen, In situ framework for coupling simulation and machine learning with application to CFD, arXiv preprint arXiv:2306.12900 (2023). doi:10.48550/arXiv.2306.12900.

[47] M. Flatken, A. Podobas, R. Fellegara, A. Basermann, J. Holke, L. Knapp, M. Kontak, N. Krullikowski, B. Nolde,

N. Brown, et al., VESTEC: Visual exploration and sampling toolkit for extreme computing—urgent decision making meets HPC, IEEE Access 11 (2023). doi:10.1109/ACCESS.2023.3301177.

[48] W. Brewer, A. Gainaru, V. R. Suter, F. Wang, M. Emani, S. Jha, AI-coupled HPC workflow applications, middleware and performance (2024).
URL https://arxiv.org/abs/2406.14315

[49] A. Alsaadi, T. Wang, A. Park, P. Bajracharya, L. Wang, F. Sun, S. Seal, V. Jadhao, G. Fox, S. Jha, Rose: Radical orchestrator for surrogate exploration, in: Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC Workshops '25, Association for Computing Machinery, New York, NY, USA, 2025, p. 61–70. doi:10.1145/3731599.3767347.

[50] M. Polese, M. Dohler, A. Dressler, M. Erol-Kantarci, R. Jana, R. Knopp, T. Melodia, Empowering the 6G cellular architecture with open RAN, IEEE Journal on Selected Areas in Communications 42 (2) (2024) 245–262. doi:10.1109/JSAC.2023.3334610.

[51] Z. Sarah, G. Nencioni, M. A. Khan, Resource allocation in multi-access edge computing for 5G-and-beyond networks, Computer Networks 227 (2023). doi:10.1016/j.comnet.2023.109720.

[52] J. Geng, M. K. Hany, R. Candell, An industrial private 5G testbed for networked automation systems, in: IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), 2024, pp. 1264–1271. doi:10.1109/AIM55361.2024.10637237.

[53] F. Zhang, J. Wang, J. Xue, R. Wang, M. Nixon, Y. Han, Time-sensitive networking (TSN) for industrial automation: Current advances and future directions, ACM Computing Surveys 57 (2) (2024). doi:10.1145/3695248.

[54] Z. Satka, M. Ashjaei, H. Fotouhi, M. Daneshtalab, M. Sjödin, S. Mubeen, A comprehensive systematic review of integration of time sensitive networking and 5G communication, Journal of Systems Architecture 138 (2023). doi:10.1016/j.sysarc.2023.102852.