University of California

Santa Barbara

Real-Time Data-Driven Decision Support for Remote Edge Deployments

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy

in

Computer Science

by

Liubov Kurafeeva UCSB TR 2026-03

Committee in charge:

Professor Chandra Krintz, co-Chair

Professor Rich Wolski, co-Chair

Professor Frederic Gibou

September 2026

The Dissertation of Liubov Kurafeeva UCSB TR 2026-03 is approved.

Professor Frederic Gibou

Professor Chandra Krintz, Committee Co-Chair

Professor Rich Wolski, Committee Co-Chair

August 2026

Real-Time Data-Driven Decision Support for Remote Edge Deployments

Liubov Kurafeeva UCSB TR 2026-03

Acknowledgements

I am deeply grateful to my advisors, Chandra Krintz and Rich Wolski, for their guidance, patience, and unwavering support throughout my Ph.D. journey. Chandra's ability to see the big picture while keeping research grounded in real-world impact has shaped the way I think about systems research. Rich's rigor and his insistence on understanding problems from first principles have made me a better scientist. I could not have asked for a more thoughtful or encouraging pair of mentors.

I owe a great deal to my lab mates and colleagues in the Department of Computer Science at UCSB. The hallway conversations, whiteboard sessions, and paper deadlines shared over coffee made the long hours of research genuinely enjoyable. They were also patient practice audiences for every presentation I gave, listening to more rehearsals of these talks than anyone should have to endure. Thank you all — you know who you are.

The work in this dissertation grew out of collaborations with a wonderful group of co-authors across multiple institutions. I am deeply grateful to that team for their energy, expertise, and dedication to this shared project.

Finally, my deepest thanks go to my friends, near and far, who kept me grounded and reminded me that there is a world beyond the terminal. Your encouragement carried me through every difficult stretch of this work.

Curriculum Vitæ

Liubov Kurafeeva UCSB TR 2026-03

Ph.D. candidate in Computer Science at the University of California, Santa Barbara (RACELab). Research combines real-time IoT sensing with high-fidelity physical simulation, focusing on edge–HPC systems for real-time decision support in controlled-environment agriculture.

Education

2026 Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2020 M.Sc. in Computer Science, Peter the Great St. Petersburg Polytechnic University.
2018 B.Sc. in Computer Science, Peter the Great St. Petersburg Polytechnic University.

Selected Publications

- L. Kurafeeva, R. Wolski, C. Krintz, and T. Smyth. "Airflow Modeling for Citrus under Protective Screens." *Sensors*, 24(19):6200, 2024.
- L. Kurafeeva, A. Subedi, R. Hartung, M. Fay, A. Biswas, S. Jha, O. Kilic, C. Krintz, A. Merzky, D. Thain, M. C. Vuran, and R. Wolski. "xGFabric: Coupling Sensor Networks and HPC Facilities with Private 5G Wireless Networks for Real-Time Digital Agriculture." *Proceedings of the SC'25 Workshops*, pages 2317–2327, 2025.
- L. Kurafeeva, R. Hartung, B. C. Carter, A. Subedi, A. Biswas, M. Fay, S. Jha, C. Krintz, A. Merzky, D. Thain, M. C. Vuran, and R. Wolski. "Hybrid Edge–HPC Systems for Low-Latency Data-Driven Inference." arXiv preprint arXiv:2605.20532, 2026.
- L. Kurafeeva, R. Hartung, B. C. Carter, A. Subedi, A. Biswas, M. Fay, S. Jha, O. Kilic, C. Krintz, A. Merzky, D. Thain, M. C. Vuran, and R. Wolski. "GrowFlow: Low-latency Computational Fluid Dynamics at the Edge for Real-time Agricultural Decision Support." *IEEE Smart Connected Systems and Emerging Technologies Conference* (IEEE SCSETech), 2026. In submission.
- L. Kurafeeva, R. Hartung, B. C. Carter, A. Subedi, A. Biswas, M. Fay, S. Jha, C. Krintz, A. Merzky, D. Thain, M. C. Vuran, and R. Wolski. "Asynchronous Scientific Workflows for Simulation-Driven Inference Across the Edge–HPC Computing Continuum." *Future Generation Computer Systems* (Elsevier), special issue, 2026. In submission.

Awards

- Outstanding Teaching Assistant Award, Department of Computer Science, University of California, Santa Barbara, 2022.

Abstract

Real-Time Data-Driven Decision Support for Remote Edge Deployments

by

Liubov Kurafeeva UCSB TR 2026-03

This dissertation develops a generalizable architecture for real-time data-driven decision support in remote edge deployments where model fidelity depends on computationally intensive, high-latency simulation. Cyber-physical applications increasingly require low-latency spatial inference while continuously adapting models to evolving environmental conditions. High-fidelity physics simulations and retraining workflows, however, are computationally expensive and typically executed on remote high-performance computing (HPC) systems under batch scheduling, creating a fundamental mismatch between the responsiveness required at the edge and the cost and availability of simulation-driven model updates.

We ground this architecture in a concrete application example: a real-world digital agriculture deployment. We first develop and validate a computational fluid dynamics (CFD) model for predicting spatial airflow within a Citrus Under Protective Screens (CUPS) deployment — a novel agricultural structure that modifies the microclimate within a field of citrus trees and requires modified farm management strategies as a result. The model and its validation methodology form the science driver for all subsequent system work.

We then present a closed-loop hybrid modeling system that couples edge-based surrogate inference with continuous simulation-driven model updating across dedicated cluster and shared HPC resources. The system deploys lightweight surrogate models at the edge while incorporating improved models asynchronously as they become available

from HPC. We instantiate this architecture in the CUPS deployment over a private 5G wireless fabric that connects field sensors, edge nodes, and remote computing facilities across the continuum. Results demonstrate edge inference in about a second or less and a 2.7× improvement in model publication cadence during an active allocation, when opportunistic HPC execution augments the dedicated pipeline.

Finally, we characterize model accuracy decay as a function of staleness* and introduce decision-time heuristics for opportunistic HPC execution. Each heuristic estimates the percentage improvement a queued job would deliver over the currently deployed model from observations already available when the job reaches the head of the batch queue, and gates release primarily on the *agreement* among three independently trained surrogates rather than on the confidence of any one of them. The No-Regret heuristic eliminates all degrading releases across the 102 decisions of the study while avoiding roughly 89% of the HPC allocation that unconditional release would consume—some 560,000 core-hours—and the complementary High-Yield heuristic captures more of the improvement that was available than No-Regret does, 26% of what perfect foresight would collect, while more than doubling the number of opportunistic model updates delivered to the edge and still avoiding 72%, demonstrating that selective prediction-guided opportunistic execution substantially outperforms unconditional release.

Although grounded in digital agriculture, the architecture, methodology, and analytical framework are domain-agnostic. Together, these contributions form a reusable foundation for real-time data-driven decision support across cyber-physical systems where edge responsiveness must be reconciled with the latency and cost of high-fidelity simulation.

*Staleness is measured either from the training cutoff or from publication to the edge, the two differing by the compute time of the iteration that produced the model; see the Glossary of Terms for the distinction and for which measure each result reports.

Contents

List of Figures

List of Tables

Glossary of Terms

Agricultural and Environmental Terms

CUPS *Citrus Under Protective Screens.* The primary deployment environment studied in this dissertation: citrus groves enclosed within nets or shade screens to protect from pests, hail, and frost. The screen geometry creates a complex enclosed airflow environment.

CEA *Controlled Environment Agriculture.* A broad category of agricultural production systems that use enclosures (greenhouses, tunnels, screens) to regulate growing conditions, including temperature, humidity, CO_2, and airflow.

Anemometer A sensor for measuring wind speed and direction. Full-coverage anemometer arrays are cost-prohibitive in large agricultural structures; this dissertation addresses the sparse-sensor inference problem that results.

Microclimate The localized atmospheric conditions (temperature, humidity, wind speed) within a small region, distinct from ambient outdoor conditions. CEA structures create complex microclimates driven by structure geometry and crop state.

Simulation and Modeling Terms

CFD *Computational Fluid Dynamics.* Numerical simulation of fluid flow governed by the Navier-Stokes equations. Used in this dissertation to model airflow inside CUPS screenhouse structures via OpenFOAM. CFD runs are computationally expensive (hours per scenario) and serve as the training data source for surrogate models.

OpenFOAM An open-source C++ CFD solver used throughout this work to generate ground-truth airflow simulations for CUPS screenhouse geometries.

Surrogate Model A fast-to-evaluate approximation of an expensive simulator. In this work, surrogate models are trained on CFD simulation outputs and deployed at the edge for real-time airflow inference. A surrogate can be implemented as a regression model (e.g., PC-R) or a neural network (e.g., PINN, FNO), so the term is not tied to a specific model class. Throughout this dissertation the following terms are used interchangeably for the same concept: *fast inference model*, *lightweight model*, and *lightweight surrogate model*. These all refer to a deployed surrogate running on an edge device under latency constraints.

PC-R *Principal Component Regression.* The surrogate modeling technique used in this dissertation. Applies PCA to reduce the dimensionality of CFD output fields, then fits a linear regression model in the reduced space. Enables inference in milliseconds after off-line training.

PCA *Principal Component Analysis.* A dimensionality reduction technique that projects high-dimensional data onto the directions of maximum variance. Used as the first stage of PC-R.

Digital Twin A continuously synchronized virtual replica of a physical asset, process, or system. In this dissertation, the digital twin concept frames the full pipeline from CFD model to real-time edge inference.

Reduced-Order Model (ROM) A family of techniques for approximating high-fidelity simulations with faster, lower-dimensional representations. PC-R is one such technique.

Computing Infrastructure Terms

Edge Computing Computation performed close to the data source (sensors, actuators) rather than in a centralized cloud or HPC facility. Reduces latency and network bandwidth requirements for real-time control loops.

HPC *High-Performance Computing.* Large-scale parallel computing clusters used for computationally intensive tasks such as CFD simulation and surrogate model training. In this work, HPC resources are accessed via job schedulers (e.g., SLURM).

GrowFlow GrowFlow. The hybrid edge–HPC system developed in this dissertation (Chapter 3) for managing real-time CFD inference workloads across an edge node and an HPC backend. It decouples real-time edge inference from asynchronous HPC-driven surrogate model updates, tolerating batch scheduling variability without interrupting inference.

RBF *Reverse Backfill.* The opportunistic HPC execution system presented in Chapter 4. It augments the dedicated model-update pipeline with jobs submitted against shared HPC allocations, and applies decision-time heuristics that commit an allocation only when a queued job is predicted to improve the currently deployed model.

xGFabric The distributed systems platform developed in this dissertation for coordinating sensing, edge, cloud, and HPC resources under variable connectivity and scheduling delays (Chapter 3). It provides delay-tolerant data movement, workflow

coordination, and model synchronization end-to-end, from devices on a private 5G wireless network to a batch-controlled HPC machine. The private 5G network is one component of the fabric, not the whole of it. GrowFlow and RBF are both built on this substrate.

WAM *Workflow Artifact Manager.* The distributed, fault-resilient log through which every stage of RBF exchanges persistent workflow artifacts — sensor observations, simulation outputs, and versioned surrogate models — rather than communicating directly (Chapter 4). Built by extending CSPOT with artifact versioning, it assigns a monotonically increasing sequence number to each entry, so a stage may retrieve either a specific artifact version or the latest available one. Because every inter-stage dependency is a durable, versioned artifact instead of a direct connection, no two stages need be co-resident or co-scheduled, and stages separated by firewalls exchange artifacts through the shared log using "push" and "pull" operations.

CSPOT A distributed runtime system providing reliable, delay-tolerant multi-node communication built on log-based persistent storage, with a serverless (FaaS) execution model in which log appends trigger functions. It is the runtime beneath xGFabric, and because all program state is logged to persistent storage, network interruption, power loss, and HPC queuing delay are all handled by the same retry-until-success mechanism (Chapter 3).

LAMINAR A network-transparent, strongly-typed distributed dataflow programming environment layered over CSPOT, which hides CSPOT's synchronization complexity behind a conventional dataflow framework. Used to express multi-stage sensing and change-detection computations that may be deployed in any combination across the edge and remote sites (Chapter 3).

FaaS *Function as a Service.* A serverless execution model where compute resources are allocated per-invocation. Used at the edge in this work for low-overhead sensor data processing.

SLURM A widely used HPC job scheduler. Used to manage CFD and surrogate training jobs on HPC cluster resources in this work.

Networking Terms

Private 5G A dedicated 5G network deployed within a facility or campus, providing cellular-grade wireless connectivity without relying on a public carrier. Used in this dissertation to connect CUPS sensor arrays to edge compute.

Network Slicing A 5G feature that partitions a physical network into multiple virtual networks with dedicated QoS guarantees. Used in xGFabric to provide bounded latency for sensor-to-edge data flows.

QoS *Quality of Service.* Network scheduling policies that prioritize traffic to meet latency, bandwidth, or reliability guarantees.

System-Specific Vocabulary The following terms are common English words that carry precise, system-defined meanings throughout this dissertation.

Iteration One complete execution of the off-line loop: data collection, CFD simulation, data transformation, surrogate model training, and model publication. Each iteration produces a newly trained model that replaces the previously deployed one.

Cutoff (also *cutoff time*) The datetime of the last sensor record used as input to a given iteration. The cutoff first feeds the Simulation Stage: sensor data collected up to this timestamp determines the boundary conditions dispatched to HPC for

CFD runs. Data collected after the cutoff belongs to the next iteration; it sets the temporal boundary between what the current model was trained on and the real-world conditions it must serve.

Compute Time The elapsed time between the cutoff and the moment the resulting model is published to the edge node. Encompasses the Simulation, Transformation, and Training stages in full. Compute time varies across deployments and across iterations of the same deployment because it is dominated by HPC batch scheduling: a job queued during high cluster load experiences longer compute time than one dispatched into an idle queue. Longer compute time means a newly published model is already older on arrival, shifting the decay curve upward and reducing the effective useful life of each iteration's model.

Cadence The wall-clock time between successive model publications; equivalently, the period of the iteration loop. Cadence is a property of the machines the off-line loop runs on and of how many simulations each iteration requires, not of the design, so this dissertation reports it more than once; the measured values appear in Chapter 3 and Chapter 4. The simulation stage is the largest single contributor. Cadence determines how quickly the system can respond to changing environmental conditions.

History Window The length of historical sensor data (measured backward from the cutoff) used as training input for a given iteration. Empirically determined to be six hours in the CUPS deployment. A window that is too short yields an under-trained model; one that is too long includes stale data from conditions that no longer apply.

Closed Loop A property of the on-line/off-line architecture: the sensor observations

that the on-line path serves are the same observations that parameterize the next off-line iteration, whose output replaces the model the on-line path is serving from. Sensing, simulation, training, and deployment therefore form a cycle that regenerates the deployed model from current conditions with no human step in it. The loop is closed but not synchronous — closure describes what feeds what, not how fast (Chapter 3).

Off-line Loop The periodic batch computation pipeline that runs one iteration: it collects sensor data up to the cutoff, dispatches CFD simulations to HPC, transforms the outputs to the surrogate grid, trains the PC-R model, and publishes it to the edge node. Runs on a fixed cadence in the background, independent of the on-line serving path.

On-line Inference The real-time serving path in which the currently published surrogate model answers incoming wind-field queries from the CUPS sensors. Runs continuously and independently of the off-line loop; always uses the most recently published model.

Model Staleness The age of the currently deployed surrogate model. Two reference points are used in this dissertation, and they differ by the *compute time* of the iteration that produced the model:

- $staleness_c$ (cutoff-relative): the elapsed time between the *cutoff* — the most recent sensor record used to produce the model's training data — and the current wall-clock time. A newly published model already has $staleness_c$ equal to the compute time of its iteration when it arrives at the edge.
- $staleness_p$ (publication-relative): the elapsed time since the model was published to the edge. Each new deployment resets $staleness_p$ to zero.

The two are related by $staleness_c = staleness_p +$ compute time. Earlier work in this line treated staleness as publication-relative, which is adequate when compute time is effectively fixed. Once multiple execution sites and multiple surrogate model types are in play, compute time varies from one iteration to the next, and only the cutoff-relative measure describes how far a model's training conditions lag the present environment. Both measures are used in this dissertation, and which one applies depends on the result rather than on where it appears: the accuracy decay characterization of Chapter 4 is reported in $staleness_p$, where a compute time that is constant across runs makes the two measures equivalent up to that offset; the opportunistic-execution heuristics built on those curves are expressed in $staleness_c$, which is what allows models produced at different sites and at different compute costs to be compared. Unqualified uses of *staleness* are cutoff-relative, and each figure and result states which measure it reports. Staleness is formally defined in Chapter 3. High staleness under either measure means the model was trained on conditions that may no longer match the present environment, leading to increased prediction error. Note: *model freshness* is used as the antonym in some contexts (high freshness = low staleness); both refer to the same dimension of model age.

Publish The act of making a newly trained surrogate model available to the on-line inference path. Publication is the final step of an iteration and atomically replaces the previous model without interrupting inference.

Scenario One CFD simulation run under a specific set of boundary conditions (wind speed and direction). A collection of scenarios constitutes the scenario library used to train the surrogate model.

Scenario Library The full set of pre-computed CFD scenarios available for surrogate training at the time of a given iteration. Defines the input distribution the

trained model can interpolate over.

Simulation Stage The phase of the off-line loop in which CFD jobs are dispatched to the HPC cluster. The dominant contributor to iteration cadence.

Transformation Stage The phase of the off-line loop in which CFD output fields are spatially resampled onto the surrogate subgrid. Required because the CFD mesh and the PC-R grid have different resolutions and point layouts.

Training Stage The phase of the off-line loop in which the PC-R surrogate model is fit to the transformed simulation outputs. Produces the model artifacts that are then published to the edge node.

Subgrid A coarser spatial discretization of the screenhouse digital model used during surrogate training and inference, derived from the full CFD mesh by relaxation. Trading spatial resolution for training and inference speed.

Decay The degradation of surrogate model accuracy over time as the physical system drifts away from the conditions represented in the scenario library (e.g., crop growth, structural changes, seasonal shifts). Motivates the periodic re-training cadence.

Chapter 1

Introduction

Applications that couple physical sensing with prediction and actuation increasingly span the edge–cloud–HPC continuum. They must ingest streaming data from distributed sensors, maintain models that reflect evolving environmental conditions, and deliver spatially resolved inferences with low latency to support operational decision-making. Achieving these goals is difficult because (i) high-fidelity physics simulations and model retraining are computationally intensive and typically executed on remote HPC systems under batch scheduling, (ii) edge connectivity can be intermittent and bandwidth-limited, and (iii) environmental dynamics can invalidate previously trained models. As a result, end-to-end systems must not only integrate heterogeneous resources across the continuum, but also adapt their behavior autonomously as conditions and infrastructure change.

The central problem is that accuracy and responsiveness come from opposite ends of that continuum. A spatially resolved prediction of a physical environment is only as good as the physics that produced it, and physics of that quality is computed by simulations that occupy dozens of compute nodes for the better part of an hour, on machines that are remote, shared, and reached through a batch queue whose delay is

itself unpredictable. The decisions such a prediction supports, by contrast, are made in the field, on modest hardware that must answer in a second or less and that may lose its network link for hours at a time. Neither end can simply be moved to the other: the simulation cannot be shrunk to fit the edge without surrendering the fidelity that justifies running it, and the edge cannot wait on the queue without surrendering the responsiveness that defines it.

A common approach is to precompute models off-line and deploy them at the edge for fast inference. This resolves the latency half of the problem: a model trained once on simulation output answers in a second or less on modest hardware, with no dependence on a remote machine at the moment a decision is made. It does not resolve the other half. A model fixed at training time degrades as conditions drift away from those it was trained on, and the obvious remedy — recomputing the simulation whenever a new observation arrives — is infeasible on cost, latency, and queue variability alike. What is needed is a way to keep the deployed model current without making the edge wait on the machine that updates it. The goal of our work is to build such a system.

Thesis statement: *A new distributed systems architecture — combining high-fidelity physics-based simulation with fast surrogate inference computed asynchronously — can deliver real-time, simulation-quality decision support.*

That architecture is intended for an edge–HPC setting, in which sensing and decision support take place at the edge while the computational resources required for simulation and model generation are located in a remote data center. System-level decisions about resource allocation are driven by the accuracy of surrogate models with respect to current deployment conditions.

Our investigation proceeded in four stages. We chose an application setting that allowed this claim to be tested empirically in the field, and we validated the components of that application using real-world measurement data from the field. We developed a

new architectural model capable of delivering results for this application in real time, continuously, encompassing data delivery, inference, and the on-line/off-line loop. We investigated the accuracy of model inference relative to the pace at which the system is able to deliver updated models. Finally, we developed a set of optimizations to improve inference accuracy to the level of field measurement error, and a strategy for utilizing HPC resources within the system profitably.

We make four contributions: a validated simulation model of airflow inside a novel agricultural structure, a distributed platform that connects a remote field site to supercomputing facilities, a design that serves predictions at the edge continuously while slower computation refreshes them in the background, and decision heuristics that commit shared supercomputing time only when the refreshed model is likely to be better. The first is the science driver — the concrete prediction problem that the remainder of the work exists to serve. The second and third build the system that delivers it. The fourth makes that system affordable to operate on resources it does not own.

We develop and validate a simulation model that predicts airflow throughout a novel agricultural structure from a small number of external measurements (Chapter 2). The structure is a Citrus Under Protective Screens (CUPS) deployment: a field of citrus trees enclosed in a permeable screen that alters the microclimate within it, and with it the farm management strategy the grower must follow. We build a computational fluid dynamics (CFD) model of the enclosed airflow and validate it at two scales — a laboratory-scale physical replica of the structure under controlled conditions, and a commercial 4-acre screenhouse in California's Central Valley using historical IoT sensor data. In both settings, modeled airflow values predominantly fall within the statistical confidence intervals of the corresponding sensor measurements. A single run of the model takes over seven hours on a single core, which is what makes the remaining

contributions necessary.

We build a distributed platform that connects sensors at a remote field site to supercomputing facilities over a private wireless network (Chapter 3). The platform spans battery-powered edge devices, intermittent wireless links, and batch-scheduled supercomputers at campus and national scale, and provides delay-tolerant data movement, workflow coordination, and reliable data delivery across all of them. We characterize its end-to-end performance, including a multi-site deployment that establishes the portability of the simulation across several HPC facilities.

We show that a fast approximate model can serve predictions at the edge without interruption while slower, more accurate computation refreshes it in the background (Chapters 3–4). The approximate model is a surrogate trained on simulation output; it answers in about a second or less on hardware at the field site, while HPC runs the full CFD simulation and retrains the surrogate on its own schedule. Because the two proceed independently, queue delays and network variability change how often the edge model is replaced but never whether it can answer. Supplementing the dedicated pipeline with spare capacity on a shared supercomputer improves the rate at which updated models reach the edge by a factor of 2.7 during an active allocation.

We measure how a deployed model loses accuracy as it ages and derive decision heuristics that commit shared supercomputing time only when the resulting model is likely to be an improvement (Chapter 4). Each heuristic estimates the percentage improvement a queued job would deliver over the model currently in service, using only observations already available when the job reaches the head of the batch queue, and decides primarily on the *agreement* among three independently trained surrogates rather than on the confidence of any one of them. The No-Regret heuristic eliminates all degrading releases across the 102 decisions of the study while avoiding roughly 89% of the HPC allocation that releasing every job would consume—some

560,000 core-hours—and the complementary High-Yield heuristic captures more of the improvement that was available, 26% of what perfect foresight would collect, while more than doubling the number of opportunistic model updates delivered to the edge and still avoiding 72%.

The remainder of this dissertation follows the order of these contributions. Chapter 2 presents the science driver: the CUPS structure, the CFD airflow model built for it, and the two-scale validation establishing both that the model is accurate and that a single run of it is far too slow to answer a decision made in the field. Chapter 3 presents the system that closes that gap, in two stages — the platform that carries sensor data from the field over a private 5G network to HPC and returns results, and then the on-line/off-line loop in which a surrogate model serves inference at the edge while HPC retrains its replacement asynchronously. Chapter 4 presents the optimization: a dual-pipeline design that supplements dedicated cluster time with capacity taken from a shared supercomputer, a characterization of how surrogate accuracy decays with staleness, and the decision heuristics that determine which of those supplementary jobs are worth their allocation. Chapter 5 draws out the design principles that generalize beyond this deployment and the questions it leaves open. Related work is surveyed within each technical chapter, alongside the contribution it bears on.

Chapter 2

Science Driver: Airflow Modeling for Citrus Under Protective Screens

Note: Text and figures in this chapter are adapted in part from: Kurafeeva et al., "Airflow Modeling for Citrus under Protective Screens," Sensors 24(19), 2024 [79].

This chapter presents the physics-based science driver for the dissertation. We establish that high-fidelity CFD simulation accurately models the internal microclimate of a large-scale agricultural screenhouse under real field conditions, and quantify the computational cost — over 7 hours of wall-clock time per run on a single core — that makes direct simulation unsuitable as a real-time inference mechanism. The validated model and its measured cost motivate the distributed surrogate-based architecture developed in subsequent chapters.

2.1 Airflow Modeling in CUPS: Application Context

2.1.1 Case Description

Citrus is a crop of substantial economic importance to the US agriculture industry. In the 2022–2023 season, it was valued by the USDA at $2.59 billion [121]. Florida, Texas, Arizona, and California are the primary US producers of citrus, with California accounting for 79% of US production during this period [121]. Once a leader in citrus production in the US, Florida's citrus production has declined by 92% since the 2003–2004 season, producing only 17% of US production in 2022–2023.

A key contributor to this decline is citrus greening disease, also known as Huanglongbing (HLB) [63, 95]. HLB is a devastating disease without an approved remediation. It is spread by an insect called the Asian Citrus Psyllid (ACP) and leads to premature fruit drop, fruit that never ripens, and eventual tree death. HLB infection is responsible for the loss of more than $4.5 billion in revenue by the US citrus industry over the period 2006 to 2010 [67]. In California, where HLB infection has not yet reached epidemic proportions, it has been found recently in multiple counties, primarily in southern California [16, 104]. The concern is that it will migrate to, and become endemic in, California's Central Valley, where much of California's—and the country's—commercial citrus is produced.

One way to protect citrus trees from ACP and HLB infection is to grow the trees under field-scale protective screens that create a physical barrier between the insects and the plants. Such structures are called Citrus Under Protective Screens (CUPS) [118], and they have been the subject of much research since the early 2020s [44, 118, 119]. CUPS is estimated to cost $0.69 to $1.03 per square foot ($30,000 to $45,000 per acre) and has been shown to be more effective than other methods for HLB protection in Florida [118, 119, 122]. Screen and net houses are also used to exclude

pests and to provide protection from sun and wind for a variety of crops in the US and worldwide [15]. Moreover, citrus groves in Florida have been shown to respond positively to the buffered environmental conditions of CUPS, yielding improved fruit quantity and quality [122]. However, CUPS has yet to be studied for HLB protection in California, where growing conditions and crop varieties differ substantially from those in Florida.

This dissertation investigates the use of CUPS in California using the first, and currently only, commercial-scale (4-acre) research CUPS, located at the Lindcove Research and Extension Center (LREC) [133] in the Central Valley of California. The LREC CUPS is instrumented with a large-scale Internet-of-Things (IoT) deployment consisting of wireless communications, resource-constrained (battery-powered) edge computers, cameras, and multiple weather stations equipped with wind sensors. Section 2.2.1 describes the structure and its sensing equipment in detail. The goal is to use this system to develop modeling, prediction, and data-driven actuation and control systems that help growers adapt their agricultural practices for CUPS deployments.

The broader goal of the LREC facility is to establish the growing environment and the commercial agricultural viability of screenhouse citrus production. Two properties of that goal shape everything that follows. First, citrus trees have useful production lifetimes exceeding 20 years, so a CUPS is effective only as long as the trees introduced into it are disease free *and* the screen remains intact for that entire period. Second, commercial viability requires the structures to be large — covering several acres each — and to accommodate tree canopy and harvesting equipment that need 15 to 20 feet of vertical space. The result is an enclosure whose interior volume is far too large to instrument densely, and whose protective function depends on the continued integrity of a screen spanning several acres.

We treat the commercial-scale CUPS facility at LREC as a living laboratory. That is, this work does not evaluate the efficacy of CUPS from a biological or agricultural

perspective. Instead, it assumes that CUPS can, and will, play an increasingly important role in the future of citrus production, and it shows how IoT can be used to support that role from a climate control and prediction viewpoint.

2.1.2 Motivation

Screen structures modify the environmental conditions within them—shading, temperature, relative humidity, and airflow—in ways that affect plant health and the growing cycle. Because of this, farm management and treatment strategies developed for open fields must be measured, predicted, and adapted before they can be applied under a screen.

Among these conditions, we focus specifically on airflow, for three reasons. The first is that airflow (i.e., wind) can have a dramatic effect on other environmental factors, such as water evaporation. The second is that, unlike relative humidity or temperature, airflow changes dynamically and its effects are often localized, owing to turbulence caused by obstacles and topological features in and around the growing area. The third is that understanding and predicting airflow is critical to managing important farming practices such as irrigation, the application of sprayed inputs, and sprayed treatment strategies including pesticides, fertilizers, and sprayed water for frost prevention.

Airflow under a CUPS is also the hardest of these conditions to observe. Because the CUPS is neither a climate-controlled structure nor a growing area fully exposed to the weather, its porous screen walls and roof, combined with the structural elements needed to support it, create airflow patterns that are difficult to measure and predict. Instrumenting the interior densely enough to observe those patterns directly is not economically viable at field scale. This motivates the central question of this chapter:

is it possible to accurately model and predict the wind flow patterns inside the structure using measurements taken outside of it?

We answer this question with a computational fluid dynamics (CFD) model of airflow through and around the CUPS. CFD models provide accurate representations and predictive capabilities in a variety of engineering and environmental disciplines [33, 92, 99], including mesoscale weather forecasting [80, 123]. The model, its requirements, and the software and hardware used to execute it are the subject of Section 2.2.

The practical payoff is a change in the spatial resolution available to decision support. Current practice is to make crop management decisions using a combination of sensor measurements covering many square kilometers, such as nearby airport meteorological data, possibly supplemented by a single local weather station. The validation presented in this chapter shows model accuracy at the meter scale within the CUPS, which should lead to more accurate decision support and actuation. In this way, this work lays the foundation for new intelligent actuation and control systems for spraying activities, irrigation scheduling, and frost prevention; Section 5.1 returns to what building that control layer would require.

2.2 CFD Model Design for Screenhouse Microclimate Modeling

In this section, we formalize our research task and detail the methods that we used to accomplish it. We first overview the CUPS IoT infrastructure and its sensing equipment (Section 2.2.1) and state the modeling task formally (Section 2.2.2). We then describe the historical sensor data used in this study (Section 2.2.3) and the

requirements that the model must satisfy (Section 2.2.4); the latter covers, in turn, our choice of physics model, our choice of software, and the hardware on which the model executes, including our deployment and our plans for using this model as part of an IoT decision support system for growers. Finally, Section 2.2.5 describes the scaled-down, controlled setting in which we developed and calibrated the model. Section 2.5 discusses the related efforts that inform this approach.

Figure 2.1: CUPS structure side view

Figure 2.2: LREC CUPS edge and screen close-up photo

2.2.1 CUPS physical infrastructure and sensing equipment

Citrus Under Protective Screens (or CUPS) is a rectangular, field-scale, porous structure for growing citrus trees. The distinct feature of this structure is a regular screen covering it on four sides and the roof. The overall view of this structure is depicted in Figure 2.1 with a close-up view of the edge of the screen in Figure 2.2. The screen excludes the admission of insects larger than the screen size but admits attenuated sunlight and ventilation. For HLB prevention, the screen size should be no larger than 40 microns to prevent Asian Citrus Psyllid infestation [53].

The commercial-scale research structure located at LREC is approximately 5.5 **meters tall,** 100 **meters wide, and** 190 **meters long.** The roof of the structure is supported by poles in a grid approximately 4.8 meters and 8.5 meters apart. Trees inside are planted every 2.4×4.3 meters (for a total capacity of 32 trees per row and 390 trees per acre). This research began before trees were introduced as part of an effort by growers to understand the growing conditions within the structure. As such, our data, model, and validation reflect the pre-tree period of the study. We plan to validate our model further with young and mature trees as part of future work.

We collected sensor data on the climate conditions inside the CUPS using AcuRite 01075RM 5-in-1 weather stations [1] and Davis Instruments Vantage Pro 2 weather stations [10], both equipped with cup anemometers. The weather stations are part of another IoT research project in the CUPS. Thus, we leveraged the sensors and deployment infrastructure without introducing changes that may impact other projects (and increase costs). The weather stations that we used in this study were located in five locations (two outside the structure and three inside) as shown in Figure 2.3.

The *north out* location was located approximately 100 **meters (m) north,**

outside of the north end of the CUPS. The anemometer at this location was mounted at a height of $10m$. The *north in* location was located $160m$ due south of *north out* ($60m$ inside the northern boundary of the CUPS). The anemometer at this location was mounted at a height of $5m$. The *middle in* location was $214m$ due south of *north out* ($114m$ inside the northern CUPS boundary). The anemometer at this location was mounted at a height of $1.5m$. The *south in* location was $276m$ south of *north out* ($176m$ inside the northern CUPS boundary). The anemometer was mounted at $1.5m$. Finally, *south out* was located $302m$ south of *north out*, immediately outside the southernmost boundary of the CUPS, at a height of $1.5m$. While the sensors were oriented along the north-south axis of the CUPS, they were not aligned perfectly due to structure requirements dictated by the agricultural layout.

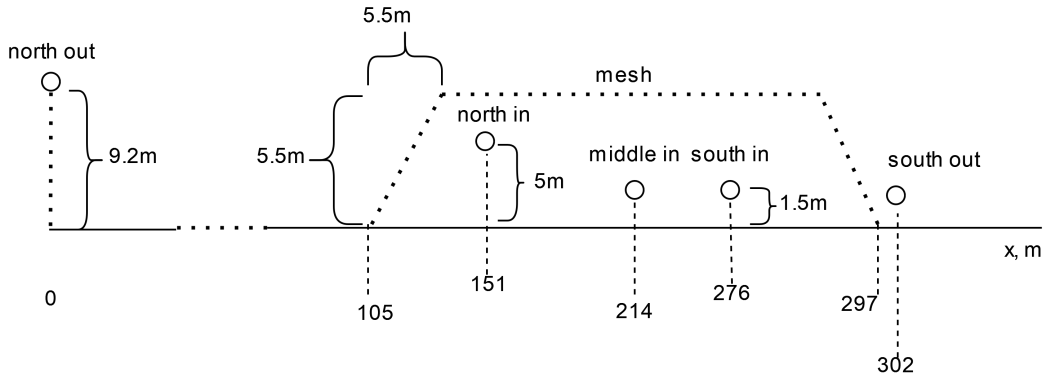

Figure 2.3: CUPS Structure and sensor positioning

2.2.2 Formalization of the task

The specific research task we pursue with this effort is *to develop and validate a CFD model for the CUPS that can be parameterized by local atmospheric measurement values (from IoT sensors) to produce a representation of airflow throughout the structure and to describe the accuracy associated with this validation.*

We focus on model validation, as opposed to specific use cases, because we believe that an accurate model of airflow is foundational to multiple applications including frost prevention, spraying activities, irrigation scheduling, and CUPS maintenance. Further, we wish to understand whether relatively inexpensive meteorological sensors can be used at commercial scale or whether more expensive (and more precise) sensors are necessary. We validate only horizontal wind speed because the weather stations to which we had access for this study are both inexpensive enough for an agricultural deployment, and designed for long-lived outdoor use. As a result they only support cup anemometers.

We employed two separate validation methodologies. In the first, we constructed a scaled-down physical replica of the CUPS which we instrumented with small-scale anemometers that were more accurate than the weather station anemometers. We developed, tested, and calibrated the CFD model under controlled indoor conditions using fans to generate simulated wind (Section 2.2.5.1). Note that the results presented in Section 2.3.1 are for the model that generates results within the statistical uncertainty bounds of the anemometers used in the controlled experiments. We made heavy use of the controlled setting to explore different models with different hyperparameter settings that were not as accurate. In the second, we back-tested the model identified in the controlled environment using historical wind measurements taken from the full-scale CUPS structure (Section 2.3.2).

2.2.3 Historical Data Description

For back-testing validation, we employ a historical archive of IoT sensor data from the LREC CUPS. The weather stations of the CUPS are connected to a co-located server (on-farm) via a wireless network. The server sends the data to campus

cloud storage to which it is connected via the Internet. Each weather station is equipped with sensors for temperature, humidity, wind speed, wind direction, rain rate, UV level, and other characteristics. The weather station collects the data from the sensors and reports the records to the server every 5 minutes. This passive data collection has been ongoing since June 15, 2021 and has approximately 170,000 records for each weather station. We filter the data to identify contiguous periods of measurement and to remove periods when one or more sensors were malfunctioning. For the purposes of this work, we extract wind speed and wind direction from the archive.

2.2.4 The CFD Model Requirements

We require that the CFD model accurately predicts wind motion (velocity and direction) within the CUPS structure given wind sensor values upwind from the structure (which we used as inputs to the model). Using such a model, it should be possible to predict internal wind flows based on localized measurements of wind conditions. The model should also be one that can be optimized (perhaps through the application of high-performance computing resources) so that it can generate these internal predictions in real-time or near real-time.

Many popular model choices for agricultural tasks [129] or wind prediction tasks [134] are AI-based solutions, but at this stage we choose to adopt a physics-based approach because it is able to make an inference from relatively parsimonious data. In contrast, the use of AI typically requires very large amounts of data for training and establishment of initial values [24]. For the commercial scale CUPS, generating datasets of sufficient size and quality necessary for training appears, at this time, to be infeasible. We thus employ an approach that is physics-based and capable of predicting wind speed and direction across a farm-scale screen structure using more

limited data that can be obtained in the commercial CUPS setting.

Our experience with IoT systems for agriculture and digital agriculture applications has led us to believe that complex use cases such as frost prevention, spray control, irrigation scheduling, etc. within the CUPS, require an airflow model that is capable of representing conditions in both low and high-pressure zones (e.g. screen fabric edges) and the turbulent flows that occur in these zones. Moreover, many of these applications require real-time or near-real-time inferences and predictions. Our ability to optimize the CFD model (possibly using large-scale computing resources such as the cloud) has also influenced our choice of model.

2.2.4.1 Choice of Model

To address the challenges imposed by these requirements, we propose to use a physics-based computational model to model airflow under CUPS. Because our task involves simulation of the interaction of the wind inside the CUPS structure, we employ a physical model for fluid and gas interactions with other fluids and rigid bodies. The governing equations are the Navier–Stokes equations, which we solve in their steady-state, Reynolds-averaged, incompressible form. Airflow through the structure is far below the compressible regime, so density is treated as constant and conservation of mass reduces to the continuity condition of Equation 2.1:

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{2.1}$$

Conservation of momentum gives the Navier–Stokes momentum equations, Equation 2.2:

$$\frac{\partial}{\partial x_j}\left(u_i u_j\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\left(\nu + \nu_t\right)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] + S_i \tag{2.2}$$

Here u_i is the Reynolds-averaged velocity, p the modified kinematic pressure (pressure normalized by the constant density, into which the isotropic part of the Reynolds stress is absorbed), ν the kinematic viscosity, and ν_t the turbulent eddy viscosity supplied by the turbulence closure described in Section 2.2.4.2. The term S_i is a momentum source, and it is the sole point at which the protective screen enters the model; the remainder of this section is concerned with determining it.

The main factor contributing to the complexity of our research task is the permeable nature of the screen. Because the screen has a complicated geometry, it is computationally infeasible without very large-scale computing resources to treat each screen element as a rigid body. The screen comprises polymer threads that are interwoven to create rectangular openings no larger than 40 microns on a side. In a commercial structure measuring 190 meters by 100 meters by 5.5 meters, there are more than 1.4×10^{13} screen openings. This scale is too large to make modeling of each screen element computationally practical. To reduce simulation time, we treat the screen as a porous media [26] — a material containing voids through which fluid can flow, to reduce the computational cost of the model. This theoretical concept has been broadly used in industry [17, 143] and research [38, 64] shows that using such emulation structures that are penetrative by target liquid or gas but have rigid base are effective (and significantly more efficient) alternatives to rigid bodies.

We use the Darcy-Forchheimer equations for fluids through porous media [98] to model screen porosity. Written in the kinematic form consistent with Equation 2.2, the momentum source of Equation 2.3 is

$$S_i = -\left(\nu D_{ij} + \frac{1}{2}|u|F_{ij}\right) u_j \tag{2.3}$$

where D_{ij} is the Darcy coefficient tensor, which sets the viscous resistance and dominates

at low pore-scale Reynolds number, F_{ij} is the Forchheimer coefficient tensor, which sets the inertial resistance and whose contribution grows quadratically with velocity, and $|u|$ is the velocity magnitude. The unknown factors for Darcy-Forchheimer are the screen description coefficients. We calculate the Darcy and Forchheimer coefficients using the geometrical calculation method from a screen sample that we obtained from the CUPS structure. To enable this, we take a high-resolution photo of the screen stretched over a dark sheet. From this image, using image processing algorithms, we calculate the relation between the holes and the material of the screen. We use this relation to estimate the coefficients as parameter $\phi = \frac{Open\ area\ of\ the\ perforated\ plate}{Total\ area\ of\ the\ perforated\ plate}$. The steps to calculate the Forchheimer coefficient f are performed using the following equations, which apply to a screen the analytical treatment of perforated plates given by Idelchik [2, 70]. Equation 2.4 states the range over which that treatment applies, where L is the thickness of the screen, d_h is the hydraulic diameter of the flow passage the screen spans, and Re is the Reynolds number of the approach flow incident on the screen. Both conditions hold for the CUPS: the screen fabric is sub-millimeter in thickness against an interior that is meters across, and at the wind speeds measured at the site (Section 2.3.2) the incident flow carries a Reynolds number of order 10^6 — well inside the fully turbulent regime in which the resistance of a screen is quadratic in velocity.

$$0 < \frac{L}{d_h} < 0.015 \text{ and } Re > 10^5 \tag{2.4}$$

$$k = \left[0.707 \cdot (1-\phi)^{0.375} + 1 - \phi^2\right] \cdot \frac{1}{\phi^2} \tag{2.5}$$

where k is the dimensionless pressure-loss coefficient of the screen, from which the Forchheimer coefficient follows directly:

$$f = \frac{k}{L} \tag{2.6}$$

Because k is a quadratic resistance coefficient, the whole of the screen resistance is carried by the Forchheimer term and the Darcy coefficient D_{ij} is taken to be zero [2, 70]. We apply the resulting resistance along the screen normal, which is the direction in which the porous zone stands for the fabric.

Another way to estimate the coefficients is to run a parameter sweep of possible coefficient values in a controlled setting and to choose the coefficients that correlate with the most accurate registration between measurements and model outputs. We discuss this approach more completely in Section 2.2.5.

2.2.4.2 Choice of Software

The software we employ must accomplish three interacting tasks. It must generate a 3D representation of the solid and porous objects that comprise the CUPS. It must solve the physics equations that describe the airflow propagation through the structure. It must plot and render the results of the physics modeling for visualization purposes.

Blender: Modeling the Structure in 3D We first create a full 3D model of the CUPS structure that captures the required properties and allows us to apply numerical methods and simulations on the model. Using Blender [55], we created a 3D model of the structure, following the guidelines and measurements that growers provided. The model consists of a plane that represents ground in the area, a grid of poles in the shape of cylinders, and 5 planes to represent different screen sections. All parts were imported separately in STL format to apply specific interaction rules for OpenFOAM [3] use.

OpenFOAM: Meshing and calculation Next, we discretize the 3D model onto a grid of cells. The step involves running multiple distinct algorithms to build the environment around the structure, importing each of the 3D model components (grid of poles and screen sections), and applying their properties such as materials and their parameters. The mesh itself is generated by OpenFOAM's *snappyHexMesh* [8] adaptive refinement tool, which fits progressively finer cells around the imported geometry. Discretization is an essential part of the processing because all of the discussed models calculate changes cell by cell, which means that the number of calculations required grows with the number of cells, but also increases the accuracy of predictions. We found the effective trade-off between speed and precision using the *surfaceFeatureExtract* [4] tool. The area around solid objects has 5 layers with a cell size of approximately $5cm \times 10cm \times 10cm$. The area surrounding the screen has 8–10 layers with a cell size of approximately $2cm \times 5cm \times 2cm$. The empty areas have the largest cell size of approximately $25cm \times 50cm \times 25cm$. The layers of different size cells are separated by 1–3 layers to make the transition gradual. This balance in large and small cells and the gradient in between helps to reduce the calculation time while maintaining the accuracy of the calculation when transitioning between areas. Applied to the commercial-scale CUPS, this scheme produces a mesh of approximately 6.5 million irregularly shaped cells covering the structure and the region surrounding it, resolving conditions at the centimeter level.

To implement the CFD calculation using the cellular representation of the CUPS structure, we used the open source software OpenFOAM [3]. This software includes numerous solvers (algorithms to calculate the spreading of the forces, and input conditions in the modeled environment) for fluids and rigid body interaction as well as ways to include the Darcy-Forchheimer model. OpenFOAM is also broadly used to study the fluid dynamics of wind in industry [92, 135] and architecture [33] for safety testing. OpenFOAM requires that we select a primary solver that will determine the

main paths of calculations and parameters that will be propagated, such as wind speed and pressure. It also takes as input the rules for these calculations and the addition of other, more specific solvers for enhanced model features (such as turbulence).

We use *porousSimpleFoam* [6] as the primary solver. This solver is steady-state (i.e. the calculations are expected to stabilize over time). The solver propagates the forces and changes, through the modeled space, until the most probable average conditions are achieved. The *porousSimpleFoam* is part of the Simple family of solvers included with OpenFOAM. However, this model does not include a turbulence calculation. We thus augment the primary solver with the k-epsilon Reynolds Averaged Simulation (RAS) method [5] to consider turbulence.

The overall algorithm is iterative. It advances the solution in discrete steps that do not overlap. Because *porousSimpleFoam* is a steady-state solver, this step is a pseudo-time increment that indexes successive iterations rather than an interval of physical time; we retain the time notation because it is the notation OpenFOAM uses for it. For the CFD computations in this study, we set this iteration interval to be $250ms$. This value was chosen experimentally. Larger values led to solver calculation failure (e.g. failure to converge) and smaller values significantly increased the overall computation time. For each iteration interval, the solver computes the new target parameters (e.g. wind speed, wind velocity, and pressure) for each cell in the spatial representation of the CUPS, based on the values in the cell and its neighboring cells at the previous iteration interval. The model iterates until values in each cell do not differ by more than a fixed threshold, after which time it is said to have "converged" to a steady state. Thus, for a set of input values, the model converges to a corresponding steady state for each cell in the modeled space.

The convergence threshold for the residual differences at each time step is a hyperparameter of the CFD model. However, degenerate spatial tessellations of

cells can lead to premature and inaccurate convergence for some input values. For these reasons, we set the threshold to be very low and monitor the residuals. We terminate the CFD computation after 240 iterations. In all cases presented herein, the model reached convergence by this iteration interval. We also export the model state every 4 intervals to reduce the storage footprint required to analyze the model output. Once converged, the model output describes a set of atmospheric conditions (e.g. wind velocity, pressure, and turbulence quantities) at a centimeter scale (i.e. in each cell) within the CUPS structure.

Data Processing We use the *ParaView* [7] software package for plotting residuals, displaying and exporting model results, and performing other data visualizations. This software is compatible with the output format used by OpenFOAM so model data can be ingressed and manipulated directly without modification.

2.2.4.3 Hardware

Our experimental setup consists of a Linux-based computer with Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz, 32Gb RAM. In this initial study, and because we were validating the model accuracy (i.e. we did not have real-time deadlines) we used one core of the CPU. As a result, modeling the commercial-scale CUPS required over 7 hours of wall-clock time to compute the model's output. It is possible to improve this computational latency using shared-memory parallelism and/or a GPU implementation of the OpenFOAM solvers. Real-time or near-real-time response will likely require cloud-based high-performance computing resources. We plan to pursue these improvements as part of future work.

2.2.5 Modeling a Controlled Setting

For model validation, we constructed a scaled-down "physical model" of the CUPS structure (shown in Figure 2.4). Figure 2.5 shows a digital rendering of this scaled test structure. This scaled version was also useful for the validation of the Darcy-Forchheimer coefficients estimations.

Figure 2.4: Scaled-down Physical Model of the LREC CUPS

The scaled-down test structure used the same screen material as is installed on the commercial-scale CUPS. We placed the test structure in an indoor controlled setting (a storage area with climate control but no forced air) and instrumented it using anemometers placed at different locations inside and outside of the test structure.

2.2.5.1 CFD Model Development, Testing, and Calibration

The region of interest surrounding the test structure measures $240 \times 122 \times 67$ **centimeters (cm) — the full extent of the structure, from its windward edge to**

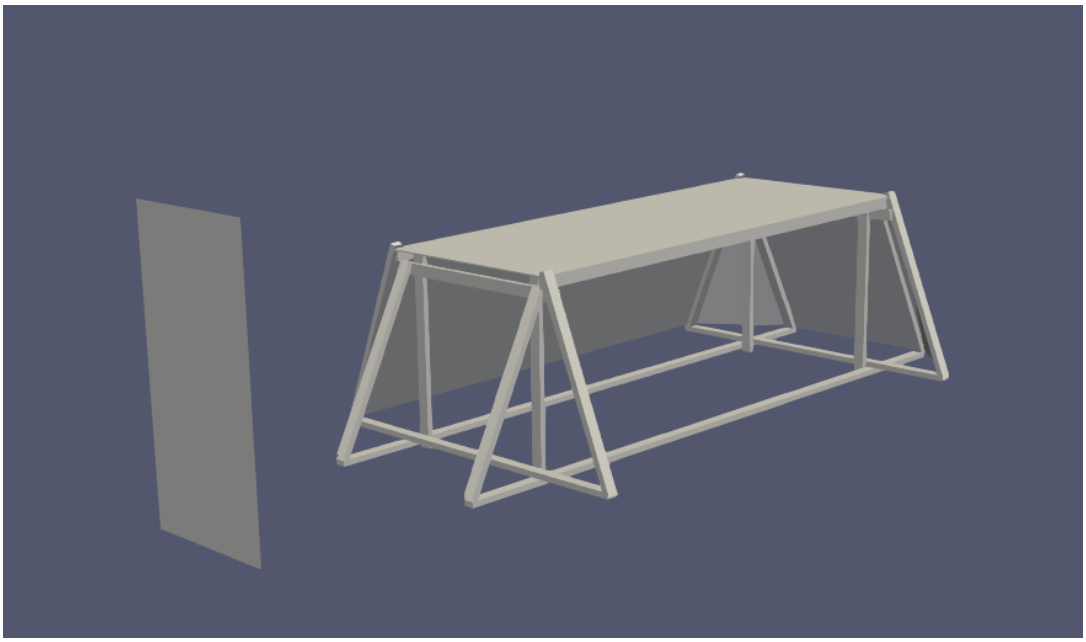

Figure 2.5: Depiction of the 3D model test set with a fan inlet (on the left) in Blender

its leeward edge (Figure 2.6) — which is approximately $1/50000^{th}$ the scale of the region we model surrounding the actual CUPS. We located two fans stacked vertically (used as a wind source) at a distance of $80cm$ from one end of the test structure along its longest dimension perpendicular to the structure's longest axis and placed the structure in the center of the test room for all experiments.

Note that the origin wind velocity (the wind velocity on the surface of each fan blade) used to parameterize the CFD model is unknown and a specification that included maximum windspeed was unavailable for the fan models in this study (we used fans purchased at a local home-improvement store). Further, measuring the origin wind speed exactly at the fan blade surface is infeasible. Thus we calibrated the measurements by starting the model with different origin wind speeds and comparing the average measured speed (after stabilization) $70cm$ from the fan blades to the modeled average for that location. We tested modeled origin wind speeds ranging from $3.0m/s$ to $4.5m/s$ in increments of $0.1m/s$, and found that $3.2m/s$ most closely

matched the average measured wind speed at $70cm$.

While both the test structure and the CUPS itself have angled side walls, we measured wind flow outside the structure (between the wind source and the windward edge) and within the maximal interior rectangular volume. This volume measured $170cm \times 63cm \times 63cm$ within the interior of the test structure.

Note that the $1/50000$ ratio given above compares volumes rather than linear dimensions, and that the test structure is not a geometric replica of the CUPS. The commercial-scale structure is far flatter relative to its footprint ($190 \times 100 \times 5.5$m) than the maximal interior volume of the test structure ($170 \times 63 \times 63$cm). What the controlled setting reproduces is the porous-media treatment of the screen and the flow regime through it under conditions we can hold fixed, rather than the geometry of the structure itself.

We instrumented the test structure using UNI-T UT363BT Mini LCD Digital Bluetooth Anemometers [9], equipped with a magnetic inductive wind speed sensor, which we used to measure wind speed at four locations oriented linearly along the centerline of the long axis. They were all suspended at a height of $30cm$ from the floor.

Figure 2.6 further details the placement configuration. Location 1 (Loc1) was $70cm$ in front of the wind source and $10cm$ in front of the leading edge of the test structure. Locations 2 through 4 (denoted Loc2, Loc3, and Loc4) were inside the structure at a distance of $138cm$, $172cm$, and $248cm$ from the wind source, respectively.

2.3 Validation of the CFD Airflow Model

In this section, we describe our validation results. We first present the results for the scaled-down CUPS model in a controlled setting. We then present a validation

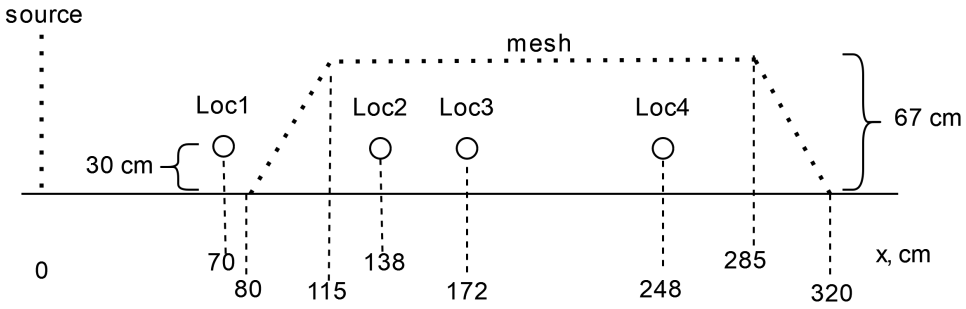

Figure 2.6: Calibration Measurement Locations Relative to the Test Structure

study for the commercial-scale CUPS using back-testing.

2.3.1 Controlled Validation Results

We set the fans to a constant speed and took measurements within the scaled-down CUPS at each location every 5 **seconds.** Each experiment lasted 1000 seconds and we recorded measurements during the last 30 seconds of the experiment to give the conditions within the structure ample time to stabilize. This interval was determined empirically. During experimentation, we observed that a change in the fan speed required as much as 20 seconds before the measured values recentered to a new average speed. We chose 30 seconds to be certain of stability and verified that longer intervals after a change did not alter the standard deviation of the measurements substantively. We chose 1000 seconds of experimentation time fearing that conditions in the room outside the structure would need time to stabilize as well. Over the course of many validations, we did not observe variation caused by the length of the experimentation interval. We chose the last 30 seconds of the experimental period to be maximally conservative with respect to conditions stabilizing but shorter experimental intervals would likely produce a similar result. The room temperature during the experiments was 18°C.

We modeled this structure using the OpenFOAM model (and the *porousSimpleFoam* solver) as described in Section 2.2.4.2. *porousSimpleFoam* is a

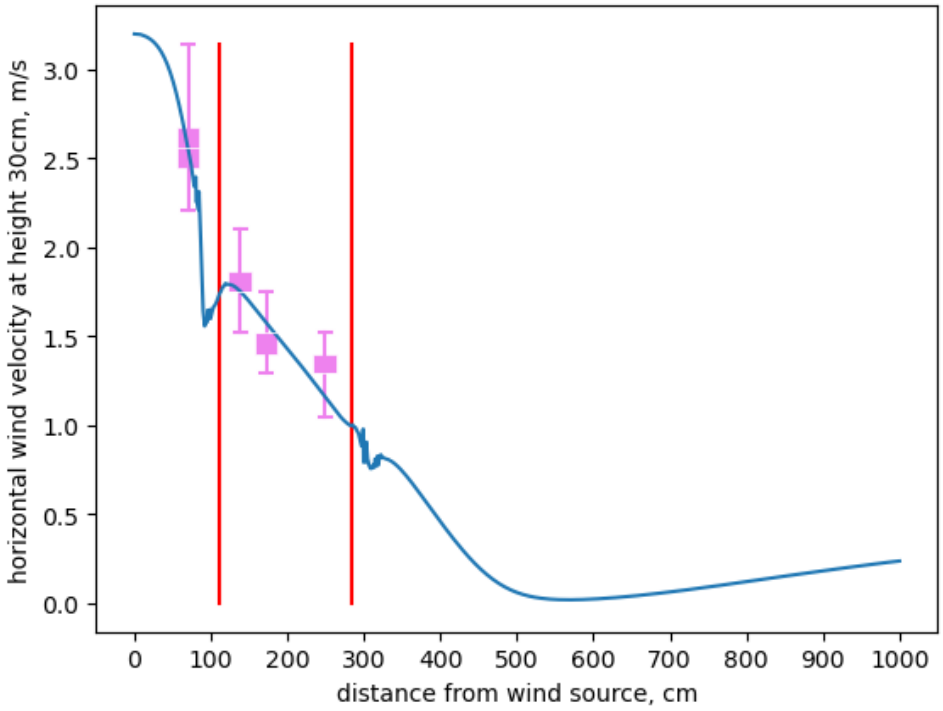

Figure 2.7: Controlled Test Structure Model Validation Results. The blue line shows model prediction, the red lines depict the edges of the structure and the pink boxplots show sensor measurements.

steady-state solver, which means it returns average values once it converges. The model output is a gridded 3-dimensional rectangular volume of wind speed vectors. We used OpenFOAM's adaptive mesh refinement feature [8] to generate the model grid that was eventually used to compute the wind velocities based on the internal geometry of the test structure. For validation purposes, we considered only the modeled values nearest the sensor locations.

Figure 2.7 shows the horizontal wind speed (on the y-axis) corresponding to the horizontal distance from the wind sources (along the x-axis) in centimeters. The vertical red lines indicate the boundaries of the maximal rectangular inner volume of the test structure. Data taken from each sensor location during the last 30 seconds of the experiment period are summarized as a "box-and-whiskers" plot. The x-axis is the horizontal distance in centimeters from the wind source and y-axis is the wind speed in meters per second (m/s).

Table 2.1: Controlled Test Structure Measurements

Loc	Measurement (m/s)	Model avg (m/s)	conf. int.
Loc1	avg: 2.58 sd: 0.20	2.65	(2.37,2.79)
Loc2	avg: 1.78 sd: 0.14	1.87	(1.64,1.94)
Loc3	avg: 1.50 sd: 0.11	**1.68**	(1.37,1.61)
Loc4	avg: 1.29 sd: n/a	**1.24**	n/a

We depict the sensor measurement data for each location using "box-and-whisker" plots in which the vertical box boundaries show upper and lower quartiles and the whiskers depict the extrema. The blue line in the figure shows the average horizontal wind speed computed by the OpenFOAM steady-state solver along the centerline of the test structure (i.e. through the sensor locations).

Table 2.1 further details these results. For each sensor location, column 2 shows the average measurement and standard deviation in meters per second, and column 3 gives the corresponding modeled average value in meters per second and column 4 gives

the two-sided 0.95 confidence interval on the mean from column 2. The modeled values for Loc1 and Loc2 fall within the corresponding confidence intervals. The two modeled values shown in boldface do not, but in both cases the discrepancy is smaller than the uncertainty of the anemometers themselves. For Loc3, the modeled value lies $0.07m/s$ above the upper bound of the interval, which is within the manufacturer's error interval of $\pm 0.1m/s$. For Loc4, the measurement values recorded during the last 30 seconds of the experiment are all 1.29 making the confidence interval computation degenerate, and the modeled value differs from the measured average by $0.05m/s$. If we substitute the manufacturer's error interval of $\pm 0.1m/s$ for the sample standard deviation associated with measurement uncertainty, both Loc3 and Loc4 are within statistical tolerance.

The lack of a sample standard deviation for Loc4 illustrates an important factor with respect to the practicality of the approach we validated. In this controlled setting, we chose small ruggedized anemometers with uncertainty intervals reported by their manufacturer that were substantially narrower than the error intervals for weather stations available in the full-scale CUPS. In neither setting (small-scale or full-scale) did the manufacturers report an uncertainty distribution for their anemometers nor did they indicate whether the interval (given as a $\pm$ value) corresponds to a specific α confidence level. Following best practices for standard uncertainty [52], we assumed that the measurement averages were Normal with an unknown variance and used a Student's t-statistic to compute confidence intervals. We chose $\alpha = 0.05$ as a standard confidence level [52]. For Loc4, we expected, but did not observe, measurements that exhibited sample variation. More precise laboratory-grade anemometers (to which we did not have access for this study) would allow for a more precise validation of Loc4. Thus, as it is, the validation is most properly construed as validating the CFD-modeled results against anemometers that are designed to be relatively inexpensive and deployed outdoors.

Note that the controlled-setting results represent the end-point of a con-

siderable experimental exploration. Formulating the CFD model that ultimately generated the comparison shown in Figure 2.7 and Table 2.1 required a number of design choices typical of CFD model construction. That is, the process of formulating a CFD model is neither "turn-key" nor "black-box." By working with a scaled-down apparatus we were able to develop and validate a model through repeated experiments that were not possible in an uncontrolled setting.

From the data, however, it is clear that the eventual model we developed is within statistical tolerance for the sensors we deployed in the small-scale, controlled setting. It is this model that we used with historical data to validate the results for the full-scale structure.

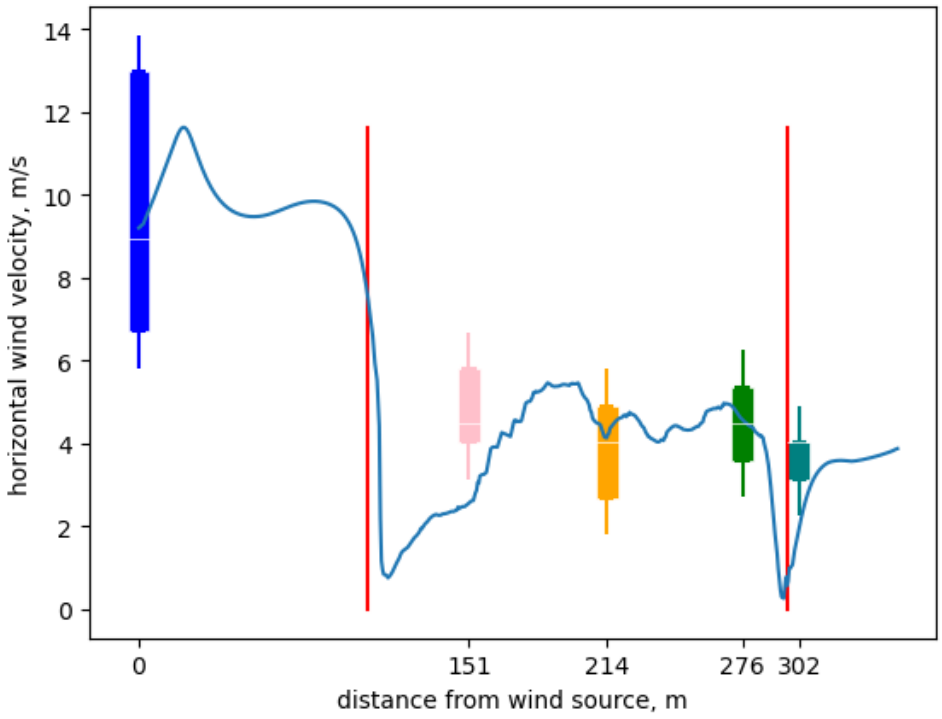

Figure 2.8: CUPS Structure Model Back-Test Results. The blue line shows model prediction, the red lines depict the edges of the structure and the boxplots show sensor measurements.

2.3.2 Back-testing Results for Commercial-scale CUPS

For validation in the full-scale CUPS structure, we selected a 25-minute period from the historical dataset that has a stable average wind speed and direction. We define stable as varying less than $3m/s$ between subsequent measurements and within $5m/s$ of the period average. In addition, we only considered candidate periods with no more than 20 degrees of variation in wind direction between measurements. We chose these parameters because they identified a period of time in the historical data archive during which the environmental conditions in and around the LREC CUPS were most similar to the conditions we were able to maintain in the controlled setting. That is, we identified in the historical archive of sensor data from the commercial-scale CUPS a period of time during which the inter-measurement difference was less than $3m/s$ and all measurements were within $5m/s$ of the average. During this period, we also needed the wind direction to be constant (as in the controlled setting) and from the north (to align the wind direction with the orientation of the CUPS and the sensors within it). Finally, we wished to use at least 5 measurement values to compute measurement confidence intervals. Among the candidate five-measurement periods (25 minutes) that satisfied all of these conditions — wind from the north, inter-measurement differences below $3m/s$, and every measurement within $5m/s$ of the period average — the smallest variation in wind direction was 20 degrees. In addition, because of the limitations of cup anemometers, all wind speeds are measured only horizontally.

We built the OpenFOAM model in the same way as we did for the small-scale test structure. We use the dimensions of the actual CUPS structure and, again, choose the grid locations (after adaptive mesh refinement) closest to the anemometers in 3 dimensions.

Figure 2.8 shows the results of an example backtest for the 25-minute

period between 16:31 and 16:56 on June 22, 2022. We parameterized the initial conditions for the model using the wind speed measured at *north out* at the beginning of the time period. In the figure, the box-and-whiskers plots show the average, the upper and lower quartiles, and the extrema for the measurement values taken from each location. The continuous blue line shows the modeled prediction converged to an average value for the horizontal path within the simulation that most nearly intersects the anemometer positions in 3 dimensions (i.e. the location within the simulation grid closest to each sensor). The distances along the x-axis are the horizontal distances from the *north out* sensor in meters. The red vertical lines identify the boundaries of the maximal rectangular volume within the CUPS structure.

Table 2.2 compares the average measurements for the five anemometer locations to the corresponding average modeled values that were provided by the CFD computation. Note that for the historical data from the commercial-scale CUPS, the horizontal wind speed measurements were 1 minute averages reported every 5 minutes. The units for the average measurements, the standard deviations, and the modeled values are all meters per second. Column 4 gives the two-sided 0.95 confidence interval on the average measurement given in column 2.

Table 2.2: Average CUPS Modeled Values Compared to Corresponding Average Measurements

Location	Measurement (m/s)	Model average (m/s)	conf. int.
north out	avg: 10.2 sd: 2.37	9.2	(7.27,13.1)
north in	avg: 4.91 sd: 1.41	**2.74**	(3.17,6.65)
middle in	avg: 4.73 sd: 1.56	4.16	(2.80,6.66)
south in	avg: 4.91 sd: 0.83	4.95	(3.89,5.93)
south out	avg: 3.66 sd: 0.49	**2.47**	(3.06,4.26)

In this case, we compute each average from 5 measurements taken 5 minutes apart. The t-statistic for the 0.95 two-sided confidence bounds on these averages is 2.776 with 4 degrees of freedom. Note that the modeled values for *north in* and *south*

out (boldfaced) fall outside their corresponding confidence intervals. For *north in*, the modeled value lies $0.43m/s$ below the lower bound of the interval, which is well within the $\pm 0.88m/s$ accuracy reported by the manufacturer for the Davis Instruments anemometer at that location; at the resolution of that sensor, the model and the measurement are indistinguishable. For *south out*, we believe the discrepancy is due to turbulence effects (which we do not model) caused by a growing block located immediately outside the CUPS on its south side next to the *south out* sensor. The modeled value there lies only $0.59m/s$ below the lower bound of the interval, however, and the AcuRite anemometer at that location is rated at $\pm 1.87m/s$, so — as with *north in* — the model and the measurement are also indistinguishable at the resolution of that sensor. Like the data taken from the controlled setting, this data indicates that the interior modeled values are within the confidence intervals surrounding the corresponding average measurements — or, for *north in* and *south out*, within the stated accuracy of the anemometers themselves — under standard assumptions about the measurement uncertainty.

Given the resolution of the measurement data, this comparison is quite good. Taken with the data shown in Table 2.1, the evidence is that the OpenFOAM model is accurate to within the statistical tolerances of the measurement data in both the controlled setting and the *in situ* full-scale setting. Further, the respective manufacturers report the anemometer accuracy for the AcuRite sensors (*middle in*, *south in*, and *south out*) as $\pm 1.87m/s$ and for the Davis Instruments sensors (*north out* and *north in*) as $\pm 0.88m/s$. Thus confidence bounds on the average measurements are likely wider than those given by a t-statistic [124]. At some small level of measurement uncertainty, the model and the measurements must differ, but determining this difference would require significantly more sophisticated and expensive anemometers than is practical for this setting.

2.4 Discussion

We performed two separate model validations of the same CFD model — one using a controlled "mock-up" of the commercial structure and the other using the structure itself. Each is intended to provide a different level of assurance that the CFD model represents airflow within the CUPS accurately. Our expectation was that the model would register with measurement more closely in a small-scale, controlled setting than in the field with the commercial structure. In the controlled setting, we were able to vary a single parameter of interest (the external wind vector) while holding all other interacting parameters relatively constant. We also used more accurate (but less durable) anemometers in the controlled setting to try to minimize the effects of measurement uncertainty on model validation. These results, summarized in Figure 2.7, provide confidence that the model correctly represents airflow through the structure at the meter scale.

We also wanted to understand the degree to which model accuracy was attenuated (versus the results shown in Figure 2.7) by the additional factors we could not control for the commercial-scale CUPS at LREC and by physical scale. Figure 2.8 shows that the CFD model's accuracy is decreased relative to the controlled setting. However, Table 2.1 and Table 2.2 indicate that the modeled values fall within the confidence intervals for the corresponding measurements, or within the stated accuracy of the anemometers themselves, in each setting. Thus, in the experiments using the commercial-scale CUPS and back-testing, it is possible to treat the effects of uncontrolled factors and scale as statistical measurement uncertainty for the purposes of evaluating model accuracy.

While our validation results are promising, it is important to acknowledge several limitations of the current study to make space for future work. The next

two subsections describe directions that this dissertation does not pursue: the alternative validations we might have performed in place of the one reported above, and a further application that the validated model would enable. Neither is addressed in the chapters that follow, and we present both as openings for subsequent research rather than as results. By addressing the challenges that remain, we aim to develop a robust and reliable tool that can support decision-making in agriculture, ultimately leading to more sustainable and productive farming practices.

2.4.1 Limitations of the Validation Approach

In this study, we have focused on validating our proposed model rather than directly implementing it as part of an agricultural application. We made this decision for two reasons. First, we were unsure of the extent to which the modeling technique would be "successful" in terms of its accuracy and we did not want to build an application without first understanding how well the model would perform. Secondly, our anecdotal experience in fielding IoT systems for agriculture led us to believe that no application we developed based on such a model would be considered trustworthy without this validation.

Another important aspect of this work is that we chose a validation approach based on what we anticipated would be the ultimate requirements for the model, namely that it was able to predict the final conditions in the structure based on conditions outside the structure. That is, in all validation experiments, we used the measured airflow outside the structure (upwind) as the initial inputs in the model, and compared the modeled values to measurements taken from within the structure.

Our view, in this regard, was that this approach was the most practical

validation since many growing regions are already instrumented with local weather stations. However, other validations are possible. For example, it is possible to parameterize the model with inputs from all but one of the sensors (not just the windward outside sensor). If our results had not indicated that the modeled results were within the bounds of statistical uncertainty, we would have conducted this validation. The consequence, however, would be that modeling a CUPS would depend critically on an array of sensors inside and outside the structure and not just sensors of external conditions.

The number of external sensors that will ultimately be necessary is an open question. Based on the results of this study, we believe that a single upwind external sensor is sufficient to parameterize the model. Because the upwind side changes with wind direction, however, this result argues for at least 4 such sensors — one located outside each of the four sides of the structure — in a practical setting, with the incident wind computed from two of the sensors. However, nearby structures and growing blocks might create external turbulence effects that we have not modeled or validated.

2.4.2 Future Work: Detecting Screen Breaches

The operational decisions discussed above — spray planning, irrigation scheduling, and frost mitigation — all consume the model's airflow predictions directly. A validated model also enables a second class of application that consumes the *residual* between prediction and measurement, and it is worth describing here because it motivates a requirement that the remainder of this dissertation must satisfy.

Detecting and rapidly repairing screen breaches in a commercial-scale CUPS is a critical open problem. The protective function of the structure depends entirely on screen integrity over a multi-decade tree lifetime (Section 2.1.1). Industrial

accidents that damage the screen are generally observed and reported by workers, but unobserved events — bird strike, foraging fauna, damage concomitant with theft — can open breaches that no one witnesses, across several acres of screen that no practical inspection regime can cover continuously.

Once the CFD model is calibrated against the structure, a persistent deviation between predicted and measured airflow is a candidate signature of such a breach, and the spatial pattern of that deviation may indicate the region of the structure in which it occurred. This is the digital-twin framing in its strongest form: the true atmospheric conditions inside the structure are twinned by the modeled interior, and it is the disagreement between the two that carries the diagnostic signal. Acting on that signal in turn suggests physical follow-up. A Farm-NG [51] wheeled robot with autonomous driving capability is planned for deployment within the structure, and can be dispatched to surveil a suspected region with an on-board camera — closing the loop from sensing, through simulation, to actuation.

Detection of this kind is not evaluated in this dissertation, and we note it as motivation rather than result. Its significance here is what it demands of the system: comparing prediction against measurement in time to act requires that the model's output be available on the same timescale as the sensor observations that parameterize it. That requirement is not met by direct simulation, as the following section establishes.

2.4.3 Computational Cost of Direct Simulation

The model validated above is accurate, but it is not fast. Because this study was concerned with model accuracy rather than with meeting real-time deadlines, we ran it on a single core of a commodity workstation, and modeling the commercial-scale CUPS required over 7 hours of wall-clock time per run (Section 2.2.4.3). A decision-

support system that must respond to conditions reported every five minutes cannot invoke a computation of that duration on demand. This establishes the baseline case — a steady-state model on a single, relatively inexpensive CPU — against which the degree of optimization necessary to achieve real-time response can be calibrated.

Reducing that latency directly, by making the simulation itself faster, only goes so far. Parallel computing techniques such as the Message Passing Interface (MPI) allow the model to split tasks across multiple processors and reduce overall computation time, and Chapter 3 reports this same simulation measured across core counts, completing in approximately 7 minutes on a single 64-core node. That is a substantial reduction, but it is not enough to close a gap of this size: even parallelized, a single high-fidelity run remains orders of magnitude too slow to sit on the critical path of an inference request.

The approach this dissertation takes instead is to remove the simulation from that path altogether. Chapter 3 uses the predictions from OpenFOAM that have proven accurate in this chapter as training data for a surrogate model that requires more initial data but that computes its result significantly faster, and executes the expensive simulation asynchronously so that its latency is never observed by the edge. The validated model developed here is therefore not the end product but the source of ground truth for everything that follows.

2.5 Related Work

2.5.1 Citrus Under Protective Screens

The specific task of climate prediction for CUPS structures in California is an open question because the design and use of CUPS screen technology for citrus is in its infancy. Indeed, our team has access to the first, and currently

only, commercial-scale CUPS research structure in California, at the Lindcove Research and Extension Center (LREC). Existing research on CUPS is focused on productivity improvements enabled by CUPS, on structure design, and on the return-on-investment enabled by CUPS in Florida [44,118,119]. Our work is distinct because it creates a digital model of meteorological conditions, specifically airflow, within a commercial-scale CUPS located in California's Central Valley.

2.5.2 CFD for Enclosed and Agricultural Spaces

Work that is closest to ours describes air movement in enclosed areas such as rooms. Researchers model these settings using computational fluid dynamics, e.g. [99]. These approaches grew in popularity during the COVID-19 pandemic, where they were used to understand human safety in indoor settings; examples of successful models can be found in the review by Mohamadi et al. [93]. Work by Bhattacharyya et al. [29] and others [18,92] also informs our approach to wind and airflow prediction, and most of these efforts use CFD models as we do. However, they focus on slow air movement rather than wind speed, and they work with enclosed, non-porous areas having limited inputs and outputs such as vents, windows, and doors. The methods applied in those works influence our approach but cannot be applied directly, given the porosity and scale of the CUPS.

A second body of work attempts to model the interaction between trees and wind. Schindler et al. [117] discuss the effects of different wind speed ranges on trees. Endalew et al. [47] validate wind and tree canopy interaction using CFD methods similar to those above, but in a strictly controlled environment, which allows their model to use solvers that are too slow for commercial-scale agriculture and capable only of small-scale computation. Other efforts [83,120] employ related CFD approaches. These all use different solvers, options, and turbulence properties from those we found to be

useful for commercial-scale CUPS, and others use core calculation parameters that are not appropriate for the setting we target. Although not directly applicable, these approaches informed our model exploration.

A third body of work models wind speed and air movement interacting with large solid objects such as buildings. The goal of these efforts is to improve the architectural design of buildings and streets by addressing climate, air quality, and noise issues. Huang et al. [68] is one example; van Druenen et al. [135] consider building geometry, and Cao et al. [33] consider ventilation performance. These approaches operate at a larger scale than other related work on modeling wind flow patterns with CFD, but their models are less accurate and less detailed than what the settings we target require.

2.5.3 CFD-Based Surrogate Modeling

Computational Fluid Dynamics (CFD) enables high-fidelity modeling of airflow and environmental conditions in buildings and agricultural structures, but its computational cost limits direct use in real-time or interactive settings. To address this limitation, recent research has explored machine learning surrogates trained on CFD outputs to accelerate inference while preserving accuracy. Chen et al. [36] survey machine learning approaches for CFD surrogate modeling in building-related physical field prediction, emphasizing that insufficient or poorly distributed training data can lead to overfitting and degraded generalization. Zhao et al. [145] propose a two-stage CFD-GNN framework that corrects the RANS solver outputs using graph neural networks, achieving predictions several orders of magnitude faster than large-eddy simulation while maintaining fidelity. Tallet et al. [130] demonstrate an on-line/off-line surrogate modeling paradigm using Proper Orthogonal Decomposition (POD) to enable real-time temperature prediction in ventilated rooms based on CFD-generated databases. The building-geometry

study of van Druenen et al. [135] applies validated CFD models in the same systematic manner, but without a surrogate.

While these approaches achieve impressive speedups and accuracy, they generally assume static surrogate models and centralized execution environments. As a result, they do not address how surrogate models should be refreshed as conditions evolve, nor how model updates can be orchestrated across distributed and intermittently connected computing resources.

2.5.4 Data-Driven Environmental Prediction

Wind speed and climate prediction outside the agricultural sector are popular research tasks, and they can be categorized by the spatial scale over which the predictions are made. Global climate and wind speed prediction, such as Slingo et al. [123], concentrates on city, state, and larger aggregated areas. Smaller-scale outdoor wind monitoring and modeling have been the focus of wind farm settings [42, 56, 113]. Lawan et al. [80] provide a comprehensive review of local point wind-speed prediction models and divide them according to the interval between the known data and the required prediction; none of the cited works, however, takes the wind map of the local area into account. Li and Shi [82] compare artificial neural network approaches to the same task, an early instance of the AI and machine learning methods now common in wind speed prediction.

Purely data-driven approaches prioritize fast inference and low computational overhead but often struggle with generalization and interpretability when deployed outside the conditions represented in their training data. Hakimi et al. [62] propose an AI-based wind speed forecasting method using neighborhood-based features and correlation-based pruning, achieving efficient inference with compact neural

networks and demonstrating generalization across diverse geographic regions. Rajaperumal and Columbus [108] compare machine learning and deep learning techniques for wind power forecasting, showing that stacked ensemble models combining Random Forest, XGBoost, and LSTM can achieve R^2 values exceeding 0.99 under favorable conditions.

Although such models perform well when training data is abundant and representative, their effectiveness depends on timely retraining as environmental conditions drift. This dependency becomes particularly challenging in edge deployments, where connectivity may be intermittent and centralized retraining pipelines introduce unpredictable latency.

2.5.5 Hybrid Physics–Data Approaches

Hybrid modeling approaches seek to combine the strengths of physics-based simulation and data-driven learning. Physics-Informed Neural Networks (PINNs) embed governing equations directly into neural network training, enabling data-efficient learning for fluid mechanics problems [32, 87], while Fourier Neural Operators (FNOs) learn mappings between function spaces to solve parametric partial differential equations with mesh-independent resolution [84]. These methods improve generalization relative to purely data-driven models and are significantly faster than full physics-based simulations.

Despite these advantages, hybrid models still require careful retraining and validation to maintain accuracy as operating conditions change. Moreover, most existing work focuses on algorithmic advances rather than the distributed systems challenges associated with deploying and updating such models in real-world, edge-centric environments.

2.5.6 Digital Agriculture and Decision Support

Within agriculture, digital technologies are increasingly used to support decision-making and resource optimization. Other work has successfully employed various types of modeling to support agricultural practices [90], with many efforts targeting the optimization of resource use and costs [20, 58]. Still other work advances the state of the art in IoT platforms [73], unmanned vehicles [61, 88], sensing techniques [103], and management systems [59]. Most existing agricultural systems target specific crops, growing environments, and tasks.

More closely related work considers how the data underlying such decisions is gathered. Adamides et al. [14] and Park et al. [102] discuss in depth the options for collecting data for climate prediction for use by growers. These studies focus on open fields, without structures or enclosures, that are visible from satellite imagery and relatively cheap to instrument using solar-powered weather stations. They do not describe the instruments and mechanics used to analyze the data, and they either rely on expert human labor or omit model details.

More recent work integrates these data streams into operational decision systems. El Hachimi et al. [46] combine deep reinforcement learning with digital twins to optimize irrigation scheduling, demonstrating measurable improvements in water use efficiency. Ciampitti et al. [37] review decision support systems in digital agriculture and emphasize that successful AI adoption requires overcoming barriers related to cost, system complexity, and integration with existing practices. Huyen et al. [69] examine digital governance in European agriculture and find that limited infrastructure and digital literacy constrain the impact of public digital services.

Together, these studies highlight both the promise of AI-driven agricultural systems and the persistent gap between research prototypes and production-

ready deployments capable of operating under real-world constraints. Closing that gap for the CUPS setting is the subject of the remainder of this dissertation.

Chapter 3

System Design: An Edge–HPC Platform for Continuous Surrogate Inference

Note: Text and figures in this chapter are adapted in part from two publications: Kurafeeva et al., "GrowFlow: Low-latency Computational Fluid Dynamics at the Edge for Real-time Agricultural Decision Support," IEEE Smart Connected Systems and Emerging Technologies Conference (IEEE SCSETech), 2026, in submission; and Kurafeeva et al., "xGFabric: Coupling Sensor Networks and HPC Facilities with Private 5G Wireless Networks for Real-Time Digital Agriculture," Proceedings of the SC'25 Workshops, 2025 [78].

This chapter describes the system that converts the validated CFD model into an operational real-time inference engine. The underlying platform—a private 5G wireless network, the CSPOT distributed runtime, and HPC coupling via xGFabric—delivers continuous sensing, delay-tolerant data movement, and scalable simulation throughput. On top of it, the on-line/off-line loop (GrowFlow) runs a surrogate model at the edge that answers a query spanning the whole structure in less than a second,

while HPC asynchronously retrains updated models, sustaining real-time service without requiring synchronous HPC availability. Chapter 4 extends this design with cost-control heuristics that govern when HPC jobs should proceed.

The system described in this chapter is deployed at the commercial CUPS screenhouse in California's Central Valley introduced in Chapter 2. That deployment is the operational setting for this work and the source of the surrogate modeling results reported here: Davis Instruments weather stations inside and outside the structure feed an on-site cluster of Intel NUCs, which serves on-line inference continuously while remote HPC resources retrain models asynchronously. The AcuRite stations that contributed to the validation study of Chapter 2 were retired from the modeling pipeline once that study concluded, for reasons specific to the site; all sensor data used from this point forward comes from the Davis stations.

3.1 System Architecture

3.1.1 xGFabric Substrate Overview

The xGFabric architecture, shown in Figure 3.1, integrates field wireless sensor networks with high-performance computing (HPC) workflows in real time — to the best of our knowledge, the first end-to-end distributed system to do so. This integration results from a full-stack, multi-scale software approach that unifies devices using a private 5G wireless network architecture, with edge, cloud, and HPC systems using the CSPOT (Serverless Platform of Things in C) [139] distributed runtime and the Laminar [45, 140] dataflow system.

The architecture described in this section is what the GrowFlow loop of Section 3.1.8 below runs on. Figure 3.1 depicts it with the sensor network sited at

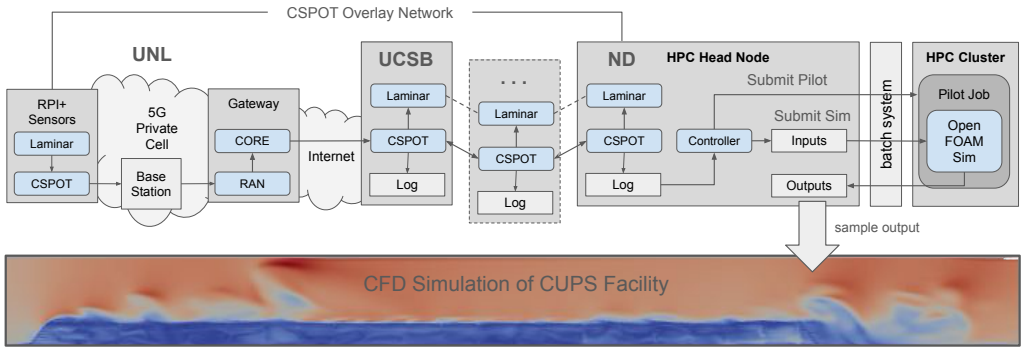

Figure 3.1: xGFabric Architecture and Sample Output

xGFabric connects a remote sensor network (left) with an HPC facility (right). In this prototype, a sensor network at UNL (U. Nebraska-Lincoln) consists of Raspberry Pis running the CSPOT distributed runtime system. These are connected by a private 5G network to the commodity Internet, and communicate with a network of CSPOT nodes at UCSB (U. California - Santa Barbara), ND (U. Notre Dame), and other facilities. At ND, the Controller process dispatches a pilot job, constructs the input data for OpenFOAM, and runs the CFD code when sufficient sensor data arrives.

UNL; at the CUPS site the same layer is realized with Davis weather stations and an Intel NUC cluster (Section 3.1.2). The runtime, data movement, and HPC coupling are identical in both.

The xGFabric architecture uses the CSPOT log-based, distributed event system to implement reliable, delay-tolerant networking, end-to-end, from devices in a private 5G network to a batch-controlled HPC machine and vice versa. xGFabric leverages this delay tolerance in three ways: first, devices operating in remote locations using 5G connectivity can be subject to frequent network interruption. Because *all* program state is logged, programs can simply pause until connectivity is restored. Secondly, CSPOT logs are implemented in persistent storage, so power-loss (which is frequent in remote IoT settings) and other device failures that do not destroy the log storage are treated in the same way as network interruption. Since all program state updates are implemented as log appends, a "failure to append" to some program log, which results either from network interruption or node failure, is simply retried until

it succeeds or the application terminates the computation. Third, xGFabric uses this delay-tolerance to mask batch-queuing delays on HPC systems that are batch-controlled. Data is "parked" in logs on storage accessible by the compute nodes of a cluster and fetched once the nodes become active.

In the following, we provide details on each of the xGFabric components: the sensor network, the private 5G wireless network, the CSPOT distributed runtime system, the Laminar data flow language, the HPC interface, and the end-to-end operation. Sections 3.1.8 and 3.1.9 then describe the on-line/off-line loop that runs on them.

3.1.2 Remote Sensor Network

The sensor network layer in xGFabric consists of edge devices used to collect, pre-process, and transmit physical-world data in real time. These devices connect through a private 5G cellular network, which provides the uplink channel for transmitting data to centralized compute or storage infrastructure. The edge device is a Raspberry Pi 4 unit equipped with a 5G USB modem and running a software agent called CSPOT, which continuously forwards sensor data using standard IP networking protocols to external endpoints. At the CUPS site each unit has a Davis Instruments weather station of Chapter 2 attached to it and forwards to an on-site Intel NUC cluster (Section 3.2); on the UNL testbed, where this layer was characterized, the same units carry CUPS observations in place of a directly attached station. The source of the readings differs; the layer does not. The system supports multiple concurrently connected user equipment (UE) nodes, each transmitting independently. The use of a private 5G network enables precise control over radio resources and performance at the edge. This sensor network forms the entry point into xGFabric's virtualized data collection pipeline,

supporting modular, replicable deployments across multiple locations and ensuring high availability and robust wireless connectivity.

3.1.3 Private 5G Wireless Network

Note: The private 5G networks described here, and their performance characterization, are the work of our collaborators at the University of Nebraska–Lincoln [78]; they establish the capacity of the link on which the rest of this chapter depends.

5G networks provide the connectivity needed to access sensor networks in remote locations. Network slicing [25], a key capability of 5G, enables the creation of multiple virtual network slices within the same physical infrastructure, each tailored to different application demands. This allows the network to simultaneously support diverse use cases such as low-latency control systems, high-throughput video, or lightweight IoT traffic.

The xGFabric prototype deploys two laboratory-sited private 5G wireless networks—one for development and one for production—built using the open-source srsRAN [126] stack and the Open5GS [101] core for standalone 5G functionality. Both run as Docker [43] containers on a single commodity host, each instance pairing a gNodeB component that handles radio access network operations with a containerized 5G core providing subscriber authentication, session and mobility management, policy enforcement, and data routing. Two Ettus Research software-defined radios [49] serve as the RF frontends, synchronized through an OctoClock timeserver. User equipment consists of Raspberry Pi devices fitted with RM530N-GL 5G USB modems [110] together with a commercial smartphone, all registering through programmable sysmoISIM-SJA5 [128] SIM cards provisioned with the pysim [106] toolkit.

Running the two instances in parallel separates experimentation from measurement: the development instance allows new features such as network

slicing to be tested safely, while the production instance maintains a consistent baseline for evaluating performance, reliability, and multi-UE scenarios. The characterization summarized in Section 3.3, and reported in full by our collaborators [78], was acquired from the production environment.

3.1.4 CSPOT Distributed Runtime System

Note: CSPOT is pre-existing RACELab infrastructure, not a contribution of this dissertation. It is described here only to the depth needed to interpret the message-latency results of Section 3.3.1.

CSPOT is a distributed runtime system that provides reliable multi-node communication built on log based storage. It is designed to function at all device scales, from microcontrollers to edge-based computers to large-scale HPC and cloud systems. It uses logs (which are simple to implement efficiently at all scales) as persistent program variables. As a result, a CSPOT program can be interrupted at any moment and the current program state will be available in persistent storage so that the program can be immediately resumed after the interruption.

Another feature of the log-based event system that underpins xGFabric is that it is highly concurrent. In particular, only the assignment of a unique sequence number with a specific log entry appended to a log must be implemented atomically. Logs are otherwise accessible concurrently. Further, to obviate the need for lock-recovery for locks that span network connections, CSPOT does not include a lock function as part of its API. Internally, the CSPOT implementation uses locks to implement atomic sequence number assignment, but it does so in a way that prevents locks from being held while a thread is waiting for network communication.

As a result, a CSPOT append operation fails in only one of two ways. Either the append fails, and the API call returns an error, or the append succeeds but the

sequence number associated with the append (to be returned from the API call) is lost, generating an error. Retrying the append until a sequence number is successfully returned ensures data integrity, but deduplication of the CSPOT logs is necessary to implement "exactly once" delivery semantics. This feature greatly enhances crash resilience and network partition tolerance, but it makes CSPOT non-intuitive for developers familiar with more conventional concurrency management mechanisms.

In particular, there is no way in a CSPOT program to fire an event only after multiple appends (to the same or different logs) have occurred. Event handlers are the only computational mechanism and a handler can only be triggered by a single log append (to avoid locking by handlers waiting for future events). As a result, a CSPOT program can always make progress (no handler blocks waiting for another handler) but handler code must parse and scan the logs to implement multi-event synchronization.

3.1.5 Laminar Dataflow System

To improve programmability over using logs and events as native programming abstractions, xGFabric includes a distributed dataflow [125] programming environment, called Laminar, which hides CSPOT synchronization complexity within a relatively conventional dataflow framework. Laminar implements a strongly-typed applicative language with strict [65] semantics using CSPOT as its runtime system. Note that CSPOT's implementation of logs makes each log a "single-assignment" variable from the programming language perspective. Thus it is possible to implement functional programming semantics (such as strict, applicative dataflow) using CSPOT. As a result, Laminar shares CSPOT's failure resiliency and crash-consistency properties [141] while implementing (on behalf of the programmer) many of the optimizations needed to avoid log scans during synchronization.

While LAMINAR is strongly-typed, it allows the developer to specify application-specific types. Thus any computation that produces the same outputs from a given set of inputs (e.g. any "stateless" computation) can be embedded within a LAMINAR computational node and "fired" as part of a LAMINAR dataflow program. For example, it is possible to treat a large-scale Computational Fluid Dynamics (CFD) application as a single node within an encompassing LAMINAR program that handles the CFD inputs and outputs.

Note that CSPOT and LAMINAR are network transparent and support multiple network protocols within the same application deployment. As such, they manage the network fabric on behalf of the application. Thus, an xGFABRIC application need not interact with the 5G network configuration interface directly. Instead, CSPOT and LAMINAR operate a network slicing interface to configure the private 5G network according to the connectivity needs of a specific deployment.

3.1.6 HPC Pilot Interface

Note: The Pilot mechanism is contributed by our Radical-Cybertools collaborators [78, 132], and the multi-site deployment it enables is the work of Ryan Hartung at the University of Notre Dame. What is described here is its integration into the off-line loop; that deployment is evaluated in Section 3.3.2.

xGFABRIC uses the Pilot [132] mechanism from Radical-Cybertools to dynamically configure the HPC environment for large-scale parallel computations. The Pilot is the path the off-line loop takes when it targets a batch-scheduled facility; where the loop instead runs on a dedicated cluster, the orchestrator of Section 3.1.9 dispatches simulations to machines directly and no pilot is involved. The aim of the pilot and its scheduling/placement algorithm is to bridge real-time data flows with

HPC simulations. Interactive pilots ensure rapid responsiveness, ideal for real-time tasks, whereas batch pilots optimize throughput and resource utilization for compute-intensive tasks at the cost of latency from scheduling. The Pilot Controller currently initiates an initial pilot using a single node and employs the following decision logic to dynamically allocate resources:

1. Assess incoming data size D and choose nodes N_{req}:

$$N_{req} = \max\left(1, \left\lceil \frac{D}{\text{threshold}} \right\rceil\right) \tag{3.1}$$

2. Evaluate currently available nodes N_{avail}:

$$N_{avail} = \sum_{p \in \text{active pilots}} \text{nodes}(p) \tag{3.2}$$

3. Decide whether to submit a new pilot:

$$\text{Submit Pilot} = \begin{cases} \text{No}, & N_{avail} \geq N_{req} \\ \text{Yes}, & N_{avail} < N_{req} \end{cases} \tag{3.3}$$

4. Determine pilot submission parameters:

$$\begin{aligned} \text{nodes} &= \min(\text{system nodes}, N_{req}) \\ \text{runtime} &= \min(\text{max system runtime}, \text{estimated task runtime}) \end{aligned} \tag{3.4}$$

The controller decides how much capacity to allocate, but not when to commit it. Proactive submission (*starting pilots early*) reduces latency but may incur idle resource overhead, while reactive submission (*starting pilots on-time*) minimizes idle

resources but can introduce startup delays. Chapter 4 takes up that question directly, deciding at the head of the batch queue whether a queued job is worth the allocation it will consume.

3.1.7 End-to-End Operation

Figure 3.1 depicts xGFabric in the context of a working, end-to-end application that dynamically triggers a CFD computation in response to changing localized atmospheric conditions in an agricultural setting. Atmospheric measurements from sensors are gathered through a private 5G network at UNL, and relayed to a data repository located at UCSB using native CSPOT. In addition, a Laminar program distributed between UNL and UCSB monitors the telemetry stream to detect when conditions change. The measurement errors from the atmospheric sensors (commodity commercial agricultural weather stations) are high enough so that consecutive readings may not be statistically determinable to be "different" and, thus, an updated CFD calculation would potentially waste HPC resources computing a new result that is statistically indistinguishable from the previous result. When the Laminar change-detection program determines that conditions have meaningfully changed, it triggers the Pilot to launch a new CFD computation on the HPC machine located at ND. The Pilot gathers the most recent atmospheric telemetry from the CSPOT logs at UCSB and launches a preprocessing pipeline to generate input files and meshing coordinates for the CFD computation (which is implemented using OpenFOAM [35]). Once these files have been prepared, the Pilot launches the computation in the ND batch queue and waits for the results to be generated as a set of output rasterized files.

Change detection is one of two policies the system supports for deciding when to run the off-line path, and both are available in the same deployment.

Under the change-triggered policy just described, an iteration begins only when the LAMINAR program judges conditions to have moved, so remote capacity is spent only once the previous result has stopped describing the site. Under the continuous policy of Section 3.1.8, the off-line path runs without pause and each iteration begins as soon as its predecessor completes, so the deployed model is as recent as the available resources allow. The two differ only in what starts an iteration: the stages, modules, and data paths described in the remainder of this chapter are the same either way. Both are exercised in the evaluation — Section 3.3.3 measures the pipeline under the change-triggered policy and reports how long a result remains valid, while Section 3.3.4 runs the loop continuously and reports the refresh cadence that results.

3.1.8 Asynchronous On-line/Off-line Loop

Our goal with GROWFLOW is to decouple the time scale of low-latency IoT inference from the much longer, and possibly more variable, latencies associated with high-fidelity model generation. To achieve this, we design the system as an asynchronous on-line/off-line loop as shown in Figure 3.2. The loop is *closed*: the sensor observations that the on-line component serves are the same observations that parameterize the next off-line iteration, whose output in turn replaces the model the on-line component is serving from. Closure describes what feeds what, not how fast; the two halves of the cycle run at their own rates.

Asynchronous On-line/Off-line Design At a high level, GROWFLOW consists of an off-line and on-line component, both of which operate continuously and communicate asynchronously. The on-line component runs at the edge, ingesting sensor data and producing low-latency inferences on-demand by parameterizing the most recently available surrogate model. In parallel, the off-line component executes computa-

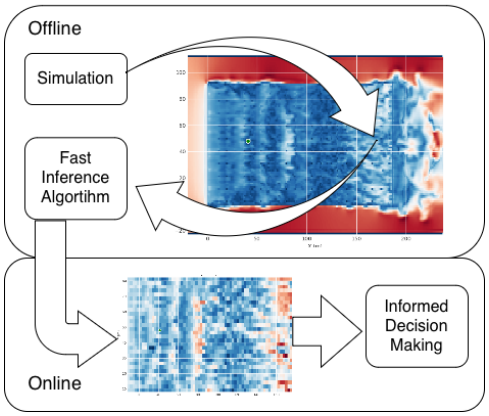

Figure 3.2: GrowFlow On-line/Off-line Architecture. Low-latency edge inference executes independently while high-fidelity simulation and model retraining executes on HPC resources and publishes "fresh" surrogate models to the edge asynchronously.

tionally intensive simulation and model retraining on HPC resources and asynchronously publishes updated surrogate models for on-line use. Rather than retraining on a fixed schedule, GrowFlow continuously executes the off-line pipeline: each iteration begins as soon as the preceding iteration completes, using the most recent sensor observations available at that time. Thus, model-refresh cadence is determined by the execution time and availability of the remote computational resources rather than by a predefined interval. This approach minimizes the time that the on-line component relies on a stale model, allowing inference to proceed independently of the off-line pipeline. It also treats model staleness as an explicit systems concern rather than assuming that model updates are instantaneous or continuously available.

Hybrid Simulation and Fast Surrogate Inference GrowFlow separates expensive high-fidelity model generation from fast surrogate inference. In our prototype deployment, the off-line path implements Computational Fluid Dynamics (CFD) to generate high-resolution training data, while the on-line path uses a lightweight surrogate model suitable for edge execution. High-fidelity simulation can capture spatially complex physical processes but is too computationally expensive for the inference

critical path. Surrogate models provide low-latency predictions but depend on timely, representative training data [36, 130]. By decoupling these two functions, GrowFlow allows on-line inference to remain responsive while the off-line path continuously refreshes model fidelity as conditions evolve. This separation makes the architecture extensible: the on-line inference model is not inherently tied to a particular surrogate modeling technique. Alternative inference approaches can be incorporated provided that they can be trained or updated using data generated by the off-line path and deployed within the resource and latency constraints of the on-line environment. Herein, we instantiate and evaluate this design using Principal Component Regression (PC-R); evaluation of alternative surrogate techniques is left for future work.

Distributed Execution Substrate GrowFlow is implemented using the xGFabric substrate described above [78, 142]. The delay-tolerant data movement, workflow coordination, and model synchronization that xGFabric provides are what allow the on-line component to continue operating while remote computation or communication is delayed. GrowFlow builds on this substrate by adding the asynchronous on-line/off-line modeling core that continuously refreshes edge inference models using simulation-generated training data.

This organization yields two important properties. First, on-line inference is non-blocking with respect to remote simulation and training: HPC queuing, network disruption, or long-running computation may delay model refresh but do not delay the current inference path. Second, model staleness becomes an observable systems property determined by the latency of the off-line loop. We evaluate this tradeoff explicitly in Section 3.3 by measuring both refresh latency and the effect of model staleness on inference accuracy.

3.1.9 Design Principles and Off-line Loop Structure

The loop described above is realized by a set of pluggable modules organized into three stages. This subsection states the application characteristics the design targets and the principles that follow from them, defines the stages, and describes the modules that implement them.

We designed GrowFlow to support cyber-physical applications with four characteristics:

- **Real-time data ingress for large-scale computation**: the system integrates streaming sensor data with remote high-performance simulation and training pipelines.
- **Heterogeneous and unreliable resources**: the architecture accommodates intermittently connected edge devices, variable network conditions, and batch-scheduled HPC resources with unpredictable latency.
- **Latency–accuracy trade-offs**: the system prioritizes low-latency inference using surrogate models while asynchronously refining those models using slower, high-fidelity simulations.
- **Dynamic environmental conditions**: the system supports adaptive sensing and data transport, responding to changing application demands and infrastructure conditions.

The architecture, shown in the schematic of Figure 3.3, embodies four design principles:

- **Module Independence**: simulation and inference algorithms are pluggable. As new algorithms emerge they can be integrated without redesigning the system,

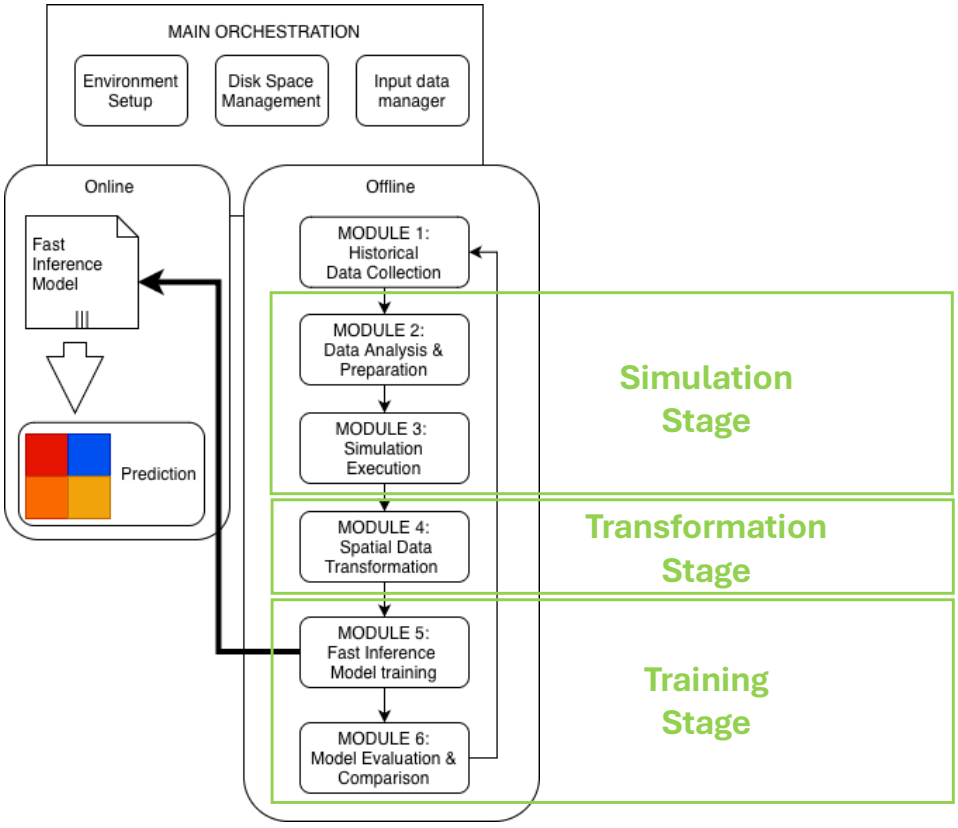

Figure 3.3: GrowFlow architecture schematic. The on-line component (left) serves inferences continuously at the edge; the off-line component (right) executes the Simulation, Transformation, and Training Stages on HPC resources and publishes updated surrogate models back to the edge.

which is critical for HPC-in-the-loop systems where algorithms and hardware evolve rapidly.

- **Parallelization**: steps that process independent data can be distributed and performed in parallel. This enables efficient use of HPC resources for expensive computations and allows partial deployment on edge devices with limited resources.

- **Asynchronous Model Update**: the on-line and off-line components operate independently, so the on-line component continues to serve low-latency inference even when off-line computation is delayed by queuing or by intermittent edge–HPC

connectivity.

- **Flexible Orchestration**: the orchestration layer can manage workflow execution across different infrastructure, accommodating diverse scheduling policies, resource availability, and network conditions.

3.1.9.1 Off-line Stages

The off-line component is organized into three stages. The *Simulation Stage* uses high-fidelity simulation to generate training sets. The *Transformation Stage* (possibly empty) reorganizes the resulting data so that it is suitable for training. The *Training Stage* fits a model capable of fast, on-demand inference. The output of the Training Stage is published, replacing any previously generated model. When a user or application requests an inference, the most recently published model is parameterized with up-to-date inputs to generate a prediction in real time.

The three stages may in principle be parallelized and overlapped as a pipeline. In this work we parallelize the Simulation Stage and the Training Stage but do not overlap the stages, and we execute the off-line path continuously in a loop so as to produce inference models as quickly as possible.

3.1.9.2 Orchestration

An orchestrator coordinates the on-line and off-line components: it initializes the execution environment for both and initiates each off-line iteration. In the first iteration the orchestration layer ensures correct software deployment and establishes connectivity to the data source; subsequent iterations verify the deployment and incorporate new input data. The orchestrator tracks the current on-line model version and ensures that the latest model is deployed and that data collection proceeds correctly.

Data collection is interrupted only if connectivity to the data source is lost.

3.1.9.3 Off-line Modules

Each off-line iteration invokes the following modules in sequence, producing one new surrogate model that serves the on-line component until the next iteration completes.

Module 1: Data Collection. Acquires data from the data source, abstracting the details of acquisition and delivering data in the required format. This includes extracting the sensor values needed as initial and boundary conditions for the simulations.

Module 2: Data Analysis and Preparation. Prepares the input data structures required by the simulations. Spatial simulations, for example, often require an initial grid of spatial cells initialized to starting values, which may require unit transformation, interpolation, relaxation, or rotation.

Module 3: Simulation. Provides the high-fidelity, domain-specific numerical solver that computes the state of the application domain from the initial conditions. This step admits parallelism in two forms: an individual simulation may execute across the cores or nodes of a distributed-memory machine, and multiple simulations, each producing a separate element of the training set, may execute concurrently. This module manages whichever form the deployment uses.

Module 4: Transformation. Post-processes simulation results through spatial reconfiguration for training. This stage is a synchronization point: it does not begin until all simulations on which it depends have completed and their outputs have been gathered.

Module 5: Training. Fits the fast inference model. This module is also parallelizable across available machines. Its output is the new surrogate that is published to the on-line component.

Module 6: Model Evaluation and Comparison. Evaluates the new model against its

predecessors, both for computational steering and for training and inference diagnostics. Errors and residuals not recorded during training are gathered here, and any external alerting — for instance on a sudden loss of accuracy — is triggered from this module.

3.2 Implementation and Execution Cost for the CUPS Deployment

We instantiate GrowFlow in the CUPS agricultural screenhouse deployment described in Chapter 2. Sensor observations collected at the edge parameterize high-fidelity CFD simulations that generate training sets for surrogate model training on large-scale compute resources in a data center. The resulting models are asynchronously returned to the edge, where GrowFlow uses the most recent model to provide low-latency wind-field inference on demand, using current sensor readings as inputs. The platform beneath it is the one described in Sections 3.1.2 through 3.1.7; what follows concerns the loop and the cost of running it at this site.

Edge Sensing and Orchestration Davis Instruments weather stations [40] deployed inside and outside CUPS report wind speed and direction every five minutes to an on-site edge cluster of Intel NUCs [71] located in an agricultural outbuilding near the CUPS. GrowFlow uses CSPOT [139] as part of the xGFabric system [78] to persist and asynchronously transfer observations to compute resources that are remote from the site and updated surrogate models back to the edge at the site. The on-line component executes at the edge using recent locally available observations and the latest published surrogate, allowing inference to continue independently of remote connectivity or off-line loop progress.

Hybrid Simulation Using the validated OpenFOAM CFD model of CUPS

described in Chapter 2 [35, 79], the off-line path generates three-dimensional wind fields from observed external wind conditions. Each simulation produces approximately 6.5 million irregularly spaced, three-dimensional wind-speed vectors over the structure. GrowFlow executes CFD simulations using newly observed sensor readings external to the CUPS as boundary conditions in parallel across available HPC resources.

Surrogate Training and Inference At the edge, GrowFlow uses Principal Component Regression (PC-R) [74] to infer the wind speed at each point of the spatial grid within the CUPS at a pre-determined spatial resolution. We select PC-R because its low training and inference cost reduces off-line loop latency while capturing temporal autocorrelation in wind observations.

Each inference uses a 12-sample (one-hour) window of external wind observations. During training, GrowFlow constructs overlapping one-hour windows from the most recent six hours of observations and associates each with CFD-generated wind values at specific spatial locations. The speed of the training (the generation of the regression coefficients for each point in the CUPS) can be tuned based on the desired resolution of the output. In this study, GrowFlow transforms CFD outputs from an irregular 6.5-million-cell mesh to a regular 12,920-point inference grid using spatial nearest-neighbor indexing via a KD-Tree [28]. This translates to a growing region of 168.2 m $\times$ 74.0 m $\times$ 6.0 m at 2m resolution within the CUPS. The four vertical levels of this region lie at $Z = 1$, 3, 5, and 7m. The lower three sit beneath the approximately 5.5m roof of the structure (Section 2.2.1); the topmost lies above it, and is included deliberately. The canopy does not grow that high, but the air at and immediately above the screen governs how sprayed inputs settle as residue on the roof, is where a tear in the mesh would first register, and characterizes airflow at the transition from the structure's roof to open air (e.g. for tracking aerosol dispersal escaping from the CUPS during spraying).

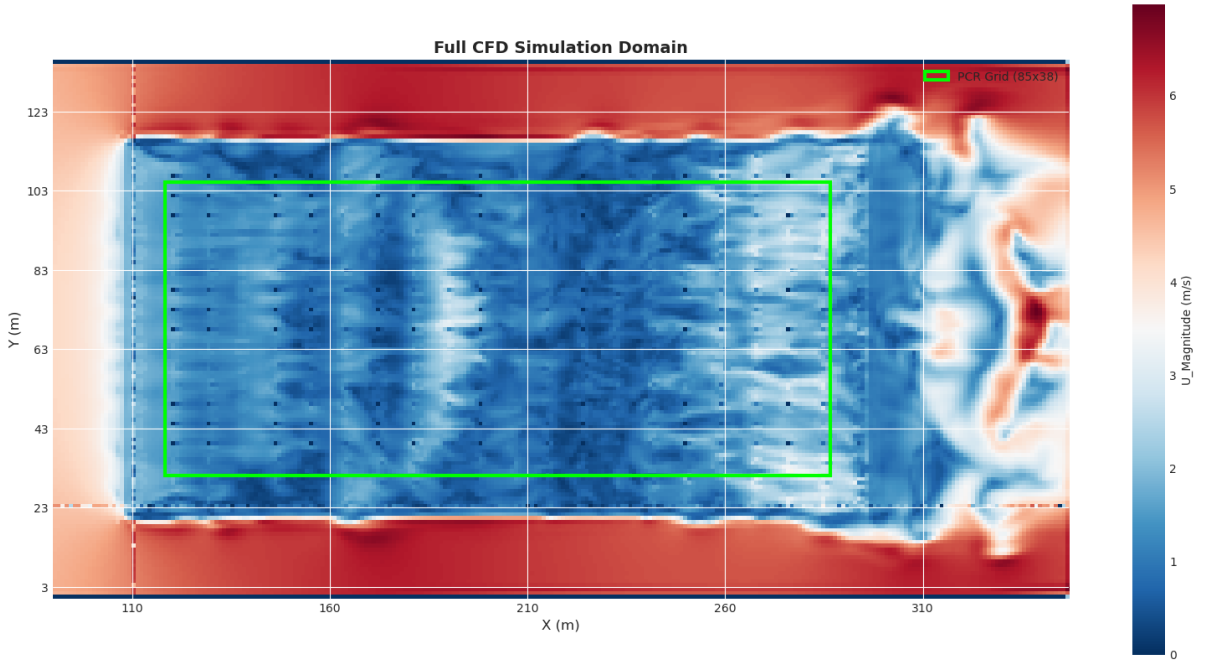

Figure 3.4: Simulation output – a view from above. The visualization shows the CUPS wind field at elevation Z = 1m. The color gradient shows the wind speeds (m/s) as calculated by a CFD simulation for a particular set of boundary conditions (the predominant wind direction is from the left). The green rectangle demarcates the growing region considered in this study.

Figure 3.4 illustrates a representative CFD-generated wind field at Z = 1m. The spatially horizontal CUPS growing region is depicted as a green rectangle. The predominant wind direction is from the left (which is North on site). The color gradient shows the wind speeds (red is high, blue is low). The simulation captures substantial wind attenuation across the screened structure and spatial variation induced by the enclosure and internal support structures, motivating the need for spatially resolved inference beyond sparse sensor measurements.

3.2.1 PC-R Validation for CUPS

Before adopting PC-R as the surrogate for this deployment, we validated it against historical sensor data. We selected approximately 170,000 wind measurements

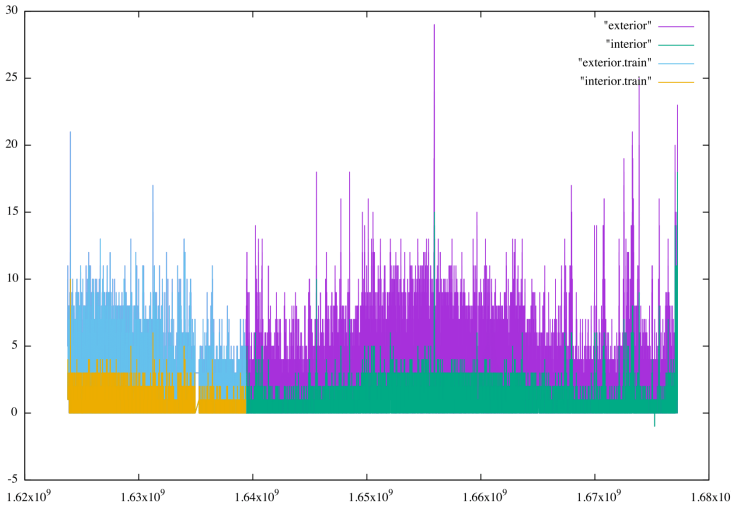

Figure 3.5: PC-R validation using CUPS external and internal weather station data. The x-axis is Linux epoch time; the y-axis is average wind speed over 10-minute periods, generated every 5 minutes. Light blue and yellow (left) are the external and internal measurements of the 50,000-sample training set; purple and green are the external and internal measurements of the 120,000-sample test set.

gathered at 5-minute intervals between June 15, 2021 and February 25, 2023 from the weather station external to the CUPS and from a station located inside the structure, and time-correlated them. We then split the time-ordered data into a 50,000-measurement training set and a 120,000-measurement test set (Figure 3.5).

Each interior measurement is matched to the exterior measurement with the nearest timestamp (typically within two seconds). The eleven exterior readings immediately preceding that measurement are prepended to form a twelve-element explanatory vector for each interior value. This yields 50,000 training vectors, from which we generate a single set of PC-R regression coefficients.

Table 3.1 summarizes the prediction error over the test set. The mean

Table 3.1: PC-R validation metrics. Prediction error when the single set of PC-R coefficients is used to predict interior measurements from exterior values in the held-out test set. Sensor measurement error is as reported by the weather station manufacturer (Davis Instruments).

MAE (m/s)	**RMSE (m/s)**	**Sensor measurement error**
0.32	0.9	0.44 to 0.88

absolute error of 0.32 m/s falls below the manufacturer-quoted measurement error range of 0.44–0.88 m/s, and the RMSE of 0.9 m/s sits slightly above it. The PC-R predictions are therefore comparable in accuracy to the measurements themselves over long time periods, which is what motivates the choice of PC-R for the CUPS prototype.

3.2.2 Grid Relaxation and Spatial Transformation

To tune training speed we generate a relaxed subgrid from the full 6.5-million-cell CFD output. The CFD cells are non-uniform in shape and size whereas the subgrid is regular in X, Y, and Z. Because the CUPS sides are angled at 45 degrees so that they can be anchored against external wind shear, only a rectangular volume within the structure is suitable for growing; Table 3.2 gives its physical boundaries, point counts, and resolution. The origin for X lies north of the structure at the exterior weather station; the origin for Y lies on the west side, shifted modestly outside the CUPS boundary so as to capture flow around the structure.

The Transformation Stage aligns CFD output with this subgrid. CFD produces an irregularly gridded 3D wind field in CSV form, one row per point with coordinates and a wind vector, but training requires a target value at each subgrid point. The difficulty is volume: each simulation produces approximately 6.5 million values — cells yield values at their points and some at face centers — and each output file is roughly one gigabyte.

Table 3.2: Grid physical boundaries. Subgrid extent, number of points, and resolution in each direction.

Axis	Min (m)	Max (m)	Range (m)	Grid Points	Resolution (m)
X	118.3	286.5	168.2	85	∼2.0
Y	31.1	105.1	74.0	38	∼2.0
Z	1.0	7.0	6.0	4	∼2.0
Total Domain			$168.2 \times 74.0 \times 6$ m^3		
Total Grid Points			12,920		

To obtain a target wind speed at each subgrid point we must locate and average the nearest CFD points, a numerical relaxation from the adjacent cells. We build a K-dimensional tree (KD-Tree) [28] over the CFD result points and query it for the nearest neighbor of each subgrid point. A KD-Tree is a space-partitioning binary search tree in which each node represents a k-dimensional point and implicitly defines a splitting hyperplane; this structure supports efficient nearest-neighbor and range queries. Indexing this way reduces the computational complexity of the relaxation enough to make per-iteration processing of the CFD output tractable.

3.2.3 Execution Time Model

Figure 3.6 shows the execution flow and the points at which the off-line path parallelizes. At the start of each cycle GrowFlow examines the data that arrived during the previous cycle to determine how many simulations the next cycle requires. The CUPS prototype schedules one simulation per available node and runs the nodes in parallel, so the Simulation Stage divides the number of simulations by the number of available nodes to balance load. The orchestrator distributes simulations to machines and

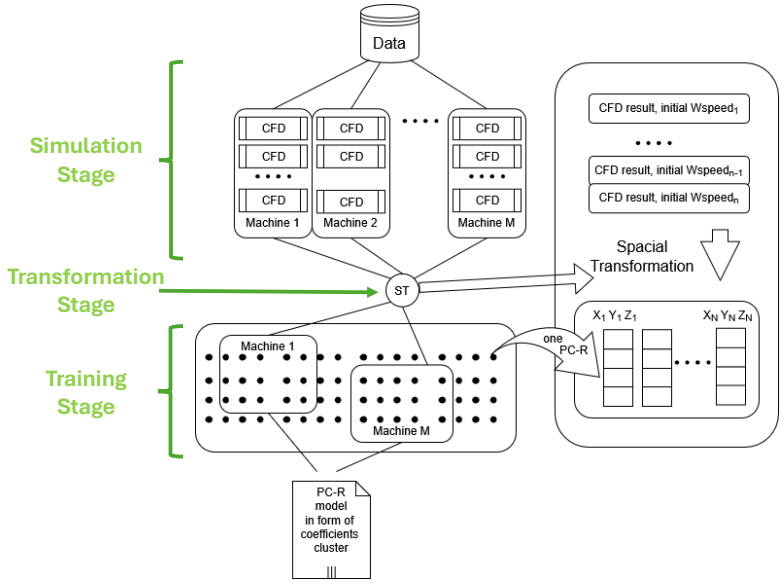

Figure 3.6: Execution flow of the GrowFlow prototype for CUPS across stages, and the opportunities for parallelization within the off-line path.

collects results on the head node, giving a total simulation time of

$$T_{sim} = T_{deploy} + \left\lceil \frac{N_{sim}}{N_{mach}} \right\rceil \cdot T_{single\_sim} + T_{collect} \tag{3.5}$$

where each term estimates the time required for the corresponding function. Because all simulations must finish before the Transformation Stage begins, the stage duration is a multiple of the individual simulation time and grows by one simulation interval for each additional batch of N_{mach} simulations.

The Transformation Stage is not parallelized in the CUPS implementation and executes on the head node. Its duration depends on subgrid size rather than on how much data accumulated since the previous cycle, so at the meter scale used here it is effectively constant; higher resolutions would cost proportionally more.

In the Training Stage each PC-R regression is independent, so parallelization proceeds by splitting subgrid points across machines. GrowFlow assigns each machine the same spatial region across all simulations produced by the Simulation Stage, so that a machine computes regressions for that region over multiple time steps. Because the explanatory vectors overlap in time, this assignment minimizes the disk I/O needed to move data between disk and memory. Total training time is

$$T_{pcr} = T_{deploy} + N_{points\_per\_mach} \cdot T_{single\_point} + T_{read\_data} + T_{collect} \tag{3.6}$$

where $N_{points\_per\_mach}$ is the number of subgrid points assigned to each machine. Despite the low per-point cost, parallelization is necessary because sequential execution requires holding large data volumes in memory for fast access.

3.2.4 Reducing Simulation Count

Beyond parallelization, a site-specific property of the deployment substantially reduces off-line loop time. The number of simulations in the Simulation Stage dominates loop duration, since that stage is both the slowest and the one that determines how much data the later stages must process. The deployed weather stations, however, report integer-valued wind speeds, and the number of distinct speed and direction combinations is bounded by the physically possible range and the direction discretization. In the evaluation of Section 3.3.4, each PC-R regression is fit over 72 training samples — one per five-minute record across the six-hour training window. A single sample pairs a

12-measurement window of consecutive exterior readings — the previous hour, sampled every five minutes — with a CFD-derived target for that window's boundary conditions, and successive samples advance the window by one measurement. Each sample therefore nominally requires a simulation of its own. Because the readings are quantized, however, the 72 windows resolve to just 12 unique wind speed and direction pairs — so for this experiment GrowFlow runs 12 CFD simulations rather than 72.

This reduction is opportunistic rather than structural. How far the 72 windows collapse depends on the wind conditions present in a given history window, so the count varies from one iteration to the next and cannot be relied upon: an iteration spanning highly variable conditions may yield close to 72 distinct pairs and realize no saving at all. GrowFlow therefore treats the deduplication as a speedup to be taken when the effective input space happens to be small, not as a guaranteed property of the loop, and systems built on this loop provision for the full 72 simulations — as RBF does in Chapter 4. Other regression parameterizations yield different counts, but the potential for reduction holds wherever sensor quantization makes the effective input space finite.

3.3 Evaluation

We evaluate the individual elements of the system — the communication latency of the CSPOT sensor network, the response time of the OpenFOAM simulation code, and the pipeline that connects them end to end — and then the on-line/off-line loop built on them. For the loop, the evaluation addresses four questions:

1. Can the off-line pipeline refresh models at a time scale suitable for continuous operation?

2. Does continuous model refresh effectively limit prediction degradation due to model staleness?

3. Can the surrogate model accurately approximate high-fidelity simulation?

4. Can the resulting surrogate provide low-latency inference suitable for edge deployment?

Taken together, these results show that GrowFlow moves the cost of high-fidelity model generation off the IoT critical path while refreshing models frequently enough to limit staleness-induced error and sustain sub-second spatial inference.

Because the components of the system are substitutable, the measurements below were not all taken against the same hardware. Sensor data originates either at the CUPS site itself or at the UNL testbed, where Raspberry Pi nodes carry CUPS observations over a private 5G network in place of the on-site equipment. Simulation and training execute either on batch-scheduled HPC reached through the Pilot interface of Section 3.1.6 — Notre Dame's CRC, Purdue's ANVIL, or TACC's Stampede3 — or on a dedicated cluster of 18 32-core virtual machines distributed across five ARM server nodes connected by 10-Gb Ethernet, of which one runs GrowFlow orchestration and 17 execute simulation and training tasks in parallel. Each virtual machine has 32 cores, 96GB of memory, and 100 GB of disk storage, and all machines are configured with Ubuntu 20.04 and include OpenFOAM 11 and Python 3; while each physical node supports 128 cores, we did not observe additional speedup from OpenFOAM CFD simulation beyond 32 cores. The architecture is identical in every case. What differs is the quality, number, and availability of the machines, and that difference is visible in the timings reported below.

The capacity of the private 5G link on which the rest of this chapter depends was characterized by members of the xGFabric team at the University

of Nebraska–Lincoln. Their evaluation covers single-user uplink throughput across bandwidths, duplexing modes, and device types; the same measurement extended to two simultaneous users; and the effect of network slicing on throughput when two user equipment nodes are assigned complementary physical resource block allocations. That work is reported in full in [78]. What matters here is the magnitude rather than the detail: every configuration they measured carries the periodic sensor telemetry and the surrogate model transfers of Section 3.3.4 with headroom to spare, so the 5G link is not the bottleneck in the on-line/off-line loop.

3.3.1 CSPOT Sensor Network

Note: The message-latency measurements reported in this subsection were taken by members of the xGFabric team at the University of Nebraska–Lincoln; the analysis and presentation given here are the work of the author and the team [78].

From a performance perspective, end-to-end, the application consists of two data paths. The first transmits telemetry data to be used to determine the initial conditions of the CFD simulation from UNL to UCSB every 5 minutes. This 5-minute interval is the reporting interval of the weather stations deployed in and around the CUPS. On the second data path, a Laminar program reads the most recent 6 telemetry values (covering the most recent 30 minutes) and compares them to the previous 30-minute period using three different tests of statistical difference. If conditions have changed in a way that is statistically measurable under the assumptions of the tests, it generates an alert indicating that a new CFD simulation is needed. The alert status is stored in a CSPOT log at UCSB and fetched to ND on a 30-minute duty cycle. The Laminar program components can be deployed either within the private 5G network or at UCSB in any combination. We execute the statistical tests and a voting algorithm to arbitrate

Table 3.3: CSPOT Message Latency for 1KB payload.

Path	Latency Avg. (ms)	Latency SD (ms)
UNL->UCSB (5G+Int.)	101	17
UNL->UCSB (Internet)	17	0.8
UCSB->ND (Internet)	92	1

between them at UCSB in this study.

Because both the telemetry data path and the Laminar data path use CSPOT as a message transport, we report the CSPOT message performance for the prototype. We measure the time to deliver a 1KB message payload, 30 times back-to-back. (The first of 30 measurements is discarded because of the initial connection start-up penalty.) Further, each message is acknowledged with a sequence number after the data has been appended to a log in persistent storage. Table 3.3 shows the average latency for delivering a 1KB message payload to persistent storage at the end of a log.

Three configurations are measured. `UNL->UCSB (5G+Int.)` measures the latency from a client at UNL carried over the 5G network and the public internet to the xGFabric node at UCSB. `UNL->UCSB (Internet)` is the same measurement, but moving the client to a wired Ethernet connection to the Internet, skipping the 5G wireless network. `UCSB->ND` measures between CSPOT nodes at UCSB and ND over the public Internet.

These latencies are an order of magnitude larger than the protocol could achieve, and for this application that does not matter. The current CSPOT internal messaging protocol (which uses ZeroMQ [66] as a transport) is optimized for reliability and not message latency. For example, to append data to a remote CSPOT log requires the client to request the size of a log element (which is fixed for each log and stored with the log as part of its header) from the site where the log is hosted before the data is actually sent from the client to the log. This size is used to determine the size of the message sent from the client carrying the element to be appended. Earlier versions of CSPOT focused on message latency used caching of the element size on the client

side to avoid fetching the element size each time. This optimization effectively halves the message latency, but causes the append to fail if the log element size is changed on the server side without a client cache update. While these and other optimizations are possible, for the prototype xGFabric application where new telemetry data is available every 300 seconds and change-detection is performed by the Laminar program every 30 minutes, further reducing the message latency by as much as an order of magnitude will not appreciably affect application performance. The effect of moving the telemetry sources from the private 5G network (latency 101ms) to the Internet directly (17ms) – an order of magnitude improvement in message latency – would be imperceptible end-to-end.

3.3.2 Portability and Simulation Latency across Compute Sites

Note: The multi-site HPC deployment described in this subsection, and the portability and simulation-latency measurements taken on it, are the work of Ryan Hartung of the xGFabric team at the University of Notre Dame; the analysis and presentation given here are the work of the author and the team [78].

We deployed and evaluated the simulation across three distinct HPC environments — Notre Dame's Center for Research Computing (CRC), Purdue's ANVIL, and the University of Texas' Advanced Computing Center's (TACC) Stampede3 — establishing that the off-line path is not tied to a single facility. Future deployments of xGFabric will make use of varying HPC sites in order to exploit the changing availability and performance of different facilities, so this multi-site deployment allowed us to assess portability challenges and performance characteristics across heterogeneous facilities.

Some practical differences between sites are easily observed, such as operating system and batch scheduler. We anticipate these and future differences by developing scripts that perform various checks, resource allocation specifications, and user

prompts for each computing environment, and by using Miniconda to capture and deploy Python components. This strategy ensures reproducible builds by maintaining explicit version specifications for all packages and libraries.

We found the primary portability challenge to be the variation in pre-installed software modules across the computing sites. Each HPC system provided different versions of OpenFOAM and ParaView with distinct dependency requirements and compilation configurations. The ParaView installations varied in their graphics library dependencies. This heterogeneity created several issues when creating display environments for rendering the VTK output files generated by the OpenFOAM simulations.

To resolve the visualization rendering challenges, we implemented a front-end SSH-based solution that requires users to establish display-forwarded connections to head nodes for offscreen rendering. Batch job submission with embedded environment variables is an alternative approach, but we found the complexity of dynamically detecting graphics library configurations and available virtual display systems across diverse HPC environments to be prohibitive. Specifically, Notre Dame and ANVIL systems utilized OpenGL-compiled ParaView with X.Org display servers supporting virtual framebuffer allocation, while Stampede3 employed Mesa-compiled ParaView. ANVIL's configuration presented additional constraints, lacking support for both virtual framebuffer and Mesa environment pass-through capabilities.

Computational performance remained relatively consistent across all three deployment sites. Figure 3.7 shows the performance of full CFD computation (including mesh generation) obtained at Notre Dame on a single node as a function of core count. The data points show the mean total execution time for each core count, and the whiskers show ± two standard deviations over 10 runs of each size, approximating a 0.95 confidence interval. With 64 cores, the average total time required to complete the simulation is 420.39 seconds with a standard deviation of 36.29 seconds. We observed comparable

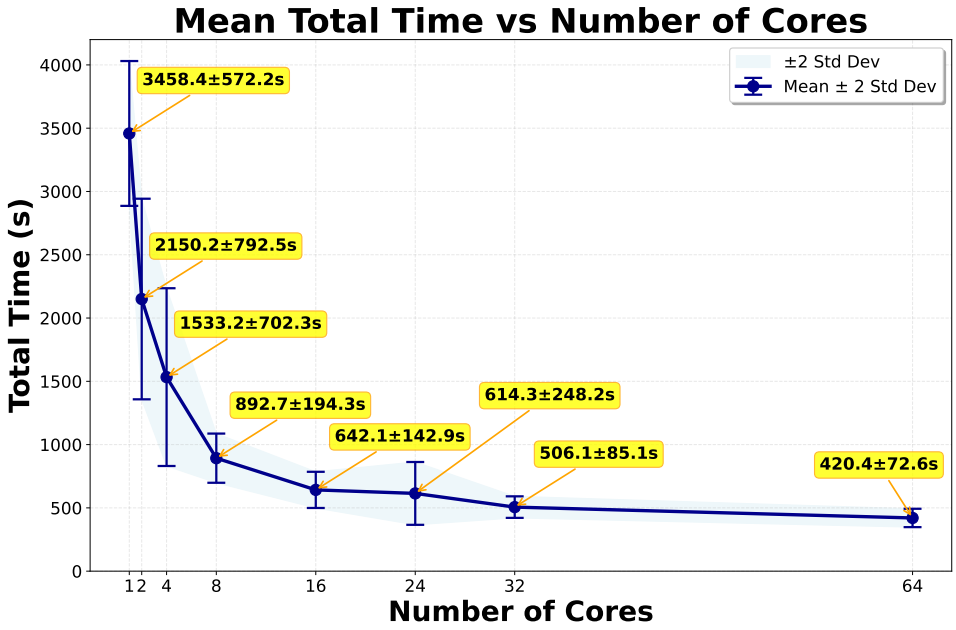

Figure 3.7: OpenFOAM Performance

Single-node speedup curve for OpenFOAM simulation on 64-core single node, 10 runs per core count, mean and 2 standard deviations shown.

performance at ANVIL and Stampede3 and report the Notre Dame measurements in detail here as representative, so the portability approach was effective across all three infrastructures.

The same simulation is substantially slower on the dedicated ARM cluster, and the gap is the clearest measure of what the choice of compute site costs. There, the twelve simulations of one Simulation Stage run concurrently across seventeen machines, so the 1136 seconds they occupy (standard deviation 72.4 seconds; Section 3.3.4) is the wall-clock time of a single run — a factor of 2.7 against the 420.39 seconds measured on 64 x86 cores at Notre Dame, with half the core count on a different processor architecture. Both figures describe the same OpenFOAM model of the same structure. Neither is the system's simulation time in the abstract: the loop cadence reported below is a property of the machines the loop happened to run on, not of the design.

3.3.3 End-to-End Pipeline Performance

The end-to-end performance of xGFabric is dominated by the time to prepare, queue, and execute the CFD simulation. The telemetry data needed to generate the input files needed by OpenFOAM is available every 300 seconds and requires approximately 200 milliseconds to transfer from the 5G network at UNL to the head node of the cluster at ND via the data repository at UCSB (cf. Table 3.3).

If a 64 core machine were dedicated to this application, it could sustain a rate of one simulation produced approximately every 7 minutes. However, these simulations are retrospective. That is, they simulate the conditions that existed at the time just before the simulation was initiated – not when it has completed. Recall that the Laminar program that can trigger a statistical change requires 30 minutes of telemetry data. Thus, with 64 cores, xGFabric is able to generate a CFD simulation that is valid for a minimum of 23 minutes (the 23 minutes remaining after the 7 minutes of simulation completes) up to the next change in wind speed.

Note that multi-node execution (using MPI) does not generate a speedup for the total application. The OpenFOAM computation, itself, runs fastest on 2 nodes, each with 64 cores. However, the total application (which includes both input-file generation and output postprocessing) slows down (despite the faster OpenFOAM time) when executed on more than one node. However, xGFabric is able to deploy the application to the best configuration possible given its performance needs, end-to-end.

Making use of a shared computing facility rather than a dedicated one introduces queueing delay as well, which varies with the offered load, the requested machines, the system capacity, and the scheduling discipline. During the course of this project, the queueing delay at Notre Dame varied from zero to 24 hours at various points, and other facilities were no different. The Pilot controller (Section 3.1.6)

is designed to sidestep this by submitting a pilot placeholder in advance, and then "activating" the pilot as needed to achieve real-time response.

3.3.4 Off-line Loop Cadence

We now turn to the on-line/off-line loop itself, beginning with the off-line iteration described in Sections 3.1.8 and 3.1.9. Timing measurements were collected across multiple executions of the complete off-line pipeline. We report the mean and standard deviation of the timings computed over 10 iterations, excluding the initial iteration, which reflects one-time system initialization effects. The evaluation periods spanned varying wind conditions (which can affect CFD convergence times and surrogate training times).

The total time for an off-line loop iteration on the dedicated ARM cluster is 31.23 minutes on average (2.78 stdev). Because each iteration begins immediately upon completion of the previous one, this execution time also determines the model-refresh cadence under continuous operation. CFD simulation accounts for the majority of this time with an average of 22.73 minutes (0.49 stdev). PC-R model training requires 3.8 minutes on average (0.38 stdev). The remaining time is that required for data analysis and preparation (0.41 minutes on average (0.04 stdev)), spatial transformation of the simulation output (4.27 minutes on average (2.25 stdev)), and general system overhead (approximately 1 second). Thus, although high-fidelity model generation requires approximately 31 minutes, this latency is removed from the on-line inference critical path by GrowFlow's asynchronous architecture.

Table 3.4 gives the same breakdown in seconds, with per-stage standard deviations. The first iteration is excluded because its Data Analysis and Preparation stage takes approximately 350 s, dominated by file transfers that subsequent iterations

Table 3.4: Average duration of GrowFlow pipeline stages. Mean and standard deviation in seconds for one loop iteration, computed over 10 iterations and excluding the initial iteration, which reflects one-time initialization effects.

Stage	Duration Mean (s)	Duration Stdev (s)
Data Analysis and Preparation	25	2.2
Simulation Execution	1364	29.6
Spatial Data Transformation	256	135.2
Fast Inference Model Training	228	22.6
Rest (system overhead)	1	0.5
Total Pipeline	**1874**	**167.8**

Table 3.5: Performance breakdown of the Fast Inference Model Training stage. Mean and standard deviation in seconds for a single iteration of the stage.

Step	Duration Mean (s)	Duration Stdev (s)
Deployment to machines	77	7.9
Per machine regression (parallel)	43	0.9
Single point regression	0.057	0.003
Data collection	23	0.9
Merge & finalize	85	8.7
Total PC-R Phase	**228**	**22.6**

do not repeat. Of the 1364 s spent in Simulation Execution, the simulations themselves account for 1136 s (stdev 72.4 s); the remainder is the data synchronization GrowFlow performs at the end of the stage. When the number of required simulations exceeds the number of available machines, GrowFlow queues them until machines free up.

Table 3.5 decomposes the Training Stage. GrowFlow distributes the 12,920 subgrid points ($85 \times 38 \times 4$, cf. Table 3.2) across the 17 worker machines, giving each approximately 760 points. For each point it fits a regression over 72 samples, each pairing 12 consecutive exterior wind measurements — the previous hour, sampled every five minutes — with the corresponding CFD-derived target. Each machine completes its 760 regressions in roughly 43 s at 0.057 s per point. Executed sequentially — one point at a time, with each point's inputs read from disk rather than held in memory — training

the full grid requires approximately five hours; parallel execution completes in 228 s, an approximately 79× speedup. The bulk of that reduction comes from in-memory processing of the reduced intermediate datasets rather than from the worker count alone, which by itself bounds the gain at 17×.

3.3.5 Model Staleness

We define *model staleness* as the elapsed time between the most recent sensor observation represented in a surrogate's training data — the cutoff — and the time at which that surrogate is queried. This is the cutoff-relative measure, $staleness_c$ in the terminology of the Glossary; every staleness figure reported in this chapter is cutoff-relative. To quantify its effect on prediction accuracy, we measure PC-R mean absolute error (MAE) as a function of staleness across multiple stable, non-zero wind periods using a training window representative of typical operation (Figure 3.8).

During our experiments, each off-line iteration completes in approximately 31 minutes on average, at which point the next iteration begins immediately using the newest available sensor observations. Thus, under continuous operation, a newly published model is already approximately 0.5 h stale when it becomes available, and is replaced approximately 31 minutes later by the model produced by the subsequent iteration. GrowFlow therefore typically operates with model staleness between approximately 0.5 and 1 h.

GrowFlow does not guarantee this cadence, however. It is a consequence of the available resources and the scheduling regime. A batch scheduler that implements queuing, for example, can lengthen this interval and make it far more irregular. That is, the off-line pipeline runs as compute resources and new sensor data permit, so an individual refresh cycle can occasionally be delayed. Figure 3.8 depicts the relationship between this

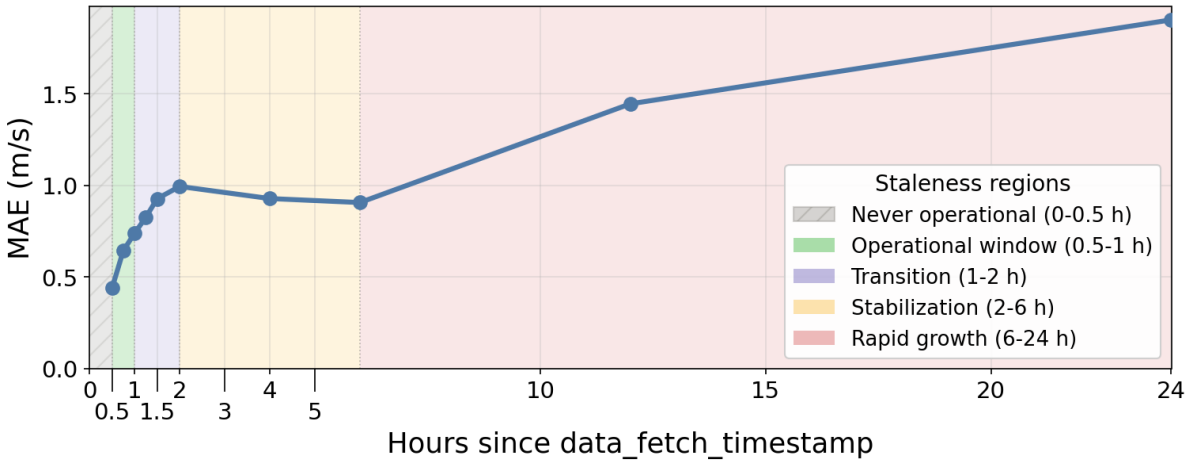

Figure 3.8: PC-R MAE vs. hours elapsed since training data fetch timestamp (6-hour training window). The x-axis origin (t=0) is the most recent sensor record used in training; one off-line loop iteration (≈31min) must complete before the model is available, placing the operational staleness window at approximately 0.5h. Without continuous refresh, MAE rises to ≈1.9m/s by 24h.

delay and MAE. For example, even if model staleness extends to 2 h, approximately twice the upper end of GrowFlow's typical operational window, MAE remains comparable in magnitude to the 0.44–0.88 m/s measurement uncertainty of the deployed sensors reported by the manufacturer. The PC-R surrogate model accuracy decays rapidly when the model staleness exceeds 6h. The staleness curve therefore characterizes not only normal operation but GrowFlow's ability to tolerate interruption in compute or data availability.

Figure 3.8 marks five staleness regions. The 0–0.5 h region is not reached operationally because one off-line iteration is required before a model becomes available. We have observed for this application that GrowFlow typically operates between 0.5 and 1 h of staleness, where MAE remains approximately 0.65–1.0 m/s. Error remains relatively stable through approximately 6 h, partly because recurring wind conditions can temporarily return toward those represented in training. Beyond 6 h, however, staleness-induced error increases substantially, reaching approximately 1.9 m/s by 24 h. Because

Table 3.6: Period 1 PC-R training fit. Per-height mean and standard deviation of R^2 and RMSE across XY grid points, when the surrogate is fit to Period 1.

Z (m)	Points	R^2 Mean	R^2 Stdev	RMSE Mean (m/s)	RMSE Stdev (m/s)
1.0	3230	0.8829	0.1231	0.1426	0.0805
3.0	3230	0.8830	0.1155	0.1510	0.0943
5.0	3230	0.9468	0.0904	0.1341	0.1321
7.0	3230	0.9528	0.0935	0.2218	0.2593
All	12920	0.9164	0.1117	0.1624	0.1619

GrowFlow's refresh cadence is roughly an order of magnitude shorter than this 6h onset of rapid error growth, even a delayed cycle would need to persist for hours before accuracy meaningfully degrades – motivating GrowFlow's continuous, best-effort model refresh via the asynchronous on-line/off-line loop rather than a hard real-time refresh guarantee.

3.3.6 Surrogate Accuracy

We next evaluate surrogate accuracy using two contiguous four-hour periods, Period 1 and Period 2, separated at 10:18 (cf. Figure 3.9). Both periods contain an extended stable high-wind regime reaching comparable peak magnitudes (≈13m/s), but differing in timing and duration, bounded by lower and more variable conditions. We train the PC-R surrogate using simulation-generated data corresponding to Period 1 and evaluate both its fit to Period 1 and its ability to predict the unseen conditions in Period 2. Table 3.6 reports the training fit across the four modeled heights: averaged over all 12,920 points the fit is strong, with mean $R^2 = 0.9164$ and mean $RMSE = 0.1624$m/s. By R^2, fit quality is highest near the roof ($Z = 5$ and 7m) and lowest at ground level near the structure walls.

Figure 3.10 compares PC-R predictions against CFD ground truth for Period 2 at a single, randomly chosen interior grid point representative of typical (non-extreme)

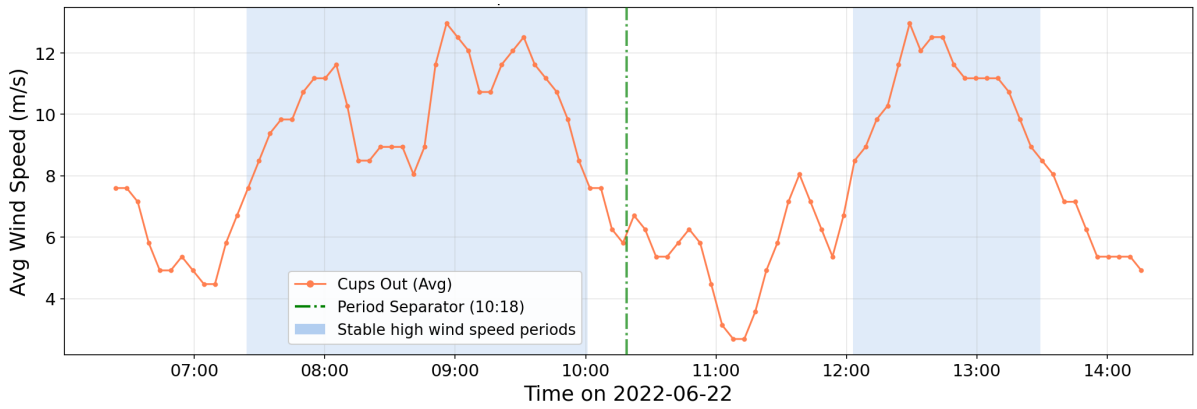

Figure 3.9: Historical wind measurements outside the CUPS spanning Period 1 and Period 2, separated at 10:18 (green line). Each period contains an extended stable high-wind regime (shaded), reaching comparable peak magnitudes (≈13m/s), bounded by lower, more variable conditions.

behavior.

Table 3.7 shows the model accuracy statistics when we use PC-R to predict unseen conditions in Period 2 over a significant future time period (4 hours). The table reports data for multiple structure heights (Z column) and presents the MAE and RMSE average and standard deviation in meters per second across points (Pts column). The worst case is located close to the roof ($Z = 5m$) of the structure. The predictions in Figure 3.10 span the same ≈4-hour duration as Period 2 itself, so the right-hand side of the figure reflects a model that is as stale as the model-staleness analysis above characterizes at the end of the evaluation window. That analysis, however, reports MAE averaged across many points and periods at each staleness level, whereas Figure 3.10 shows a single grid point under one, comparatively calm wind trajectory (2–4m/s, well below the site's ≈13m/s peaks) – its instantaneous error is not on its own representative of that average trend. Still, this point's MAE (0.37m/s) is well below the Z=5m average in Table 3.7 (1.19m/s), showing that individual points can deviate substantially, in either direction, from the aggregate staleness-driven error.

To contextualize the average MAE of 0.80m/s, we note that external wind

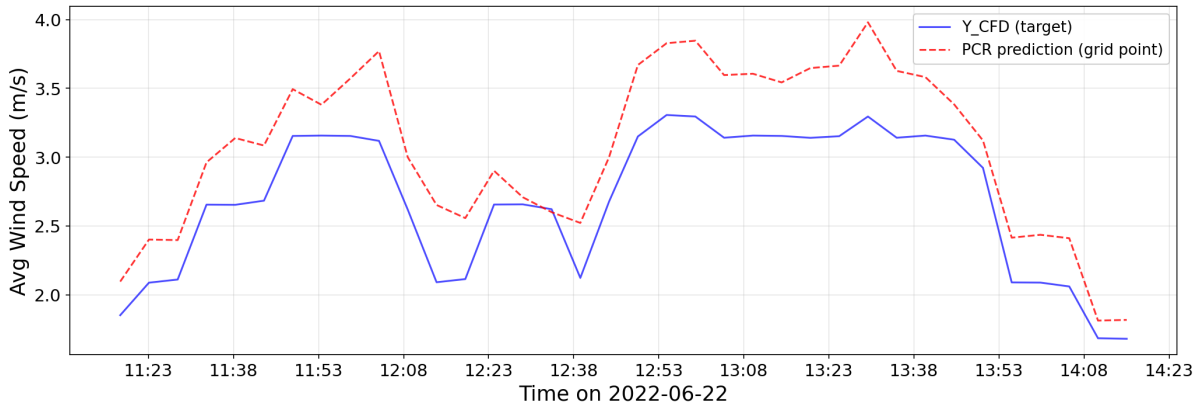

Figure 3.10: Comparison of CFD ground truth (solid) and PC-R predictions (dashed) at a representative interior grid point (x=40.3m, y=48.1m, z=5.0m, relative to the growing-region origin) over the full Period 2 duration in Figure 3.9. The model is trained exclusively on Period 1; at this point, prediction accuracy is MAE=0.37m/s, R^2=0.39, $RMSE$=0.40m/s.

speeds at the CUPS site range from 0 to 17m/s and average approximately 5m/s; the MAE therefore represents roughly 16% of the mean wind speed. For the operational decisions this system supports—spray timing, frost mitigation, and environmental monitoring—a useful reference point for interpreting this error is the accuracy of the sensors themselves. GrowFlow's overall MAE is comparable in magnitude to the manufacturer's reported sensor error range (0.44–0.88m/s), indicating that surrogate error is commensurate with measurement uncertainty at operational scales.

3.3.7 Spatial Visualization

Figures 3.11 and 3.12 show the visualizations growers require from a decision support tool: wind field predictions spanning the entire CUPS structure, produced from a single off-line loop iteration. Figures 3.13 and 3.14 show the corresponding high-resolution CFD fields that GrowFlow generates midway through the same iteration, at the end of the Simulation Stage.

Placing the two side by side illustrates the fidelity–responsiveness trade-off

Table 3.7: PC-R Model Prediction Accuracy for Future Time Period. Wind speed statistics across XY grid points at each height considered (Z) when we train on Period 1 and evaluate on Period 2 (Period-2 data held out from training). The average MAE across points and heights is 0.80 m/s.

Z (m)	Pts	RMSE Mean	RMSE Stdev	MAE Mean (m/s)	MAE Stdev (m/s)
1.0	3230	0.5553	0.3248	0.4685	0.3008
3.0	3230	0.5003	0.2776	0.4160	0.2477
5.0	3230	1.2954	0.8962	1.1884	0.8650
7.0	3230	1.2460	1.0359	1.1247	0.9599
All	12920	0.8993	0.8082	**0.7994**	0.7640

that underpins the architecture. CFD resolves fine-grained flow structures and boundary effects but is far too expensive for the inference critical path. PC-R operates on a coarser spatial representation, enabling prediction over the whole structure in less than a second.

The reduced resolution is a deliberate choice: it trades fine-scale detail for low-latency inference, and PC-R predictions may smooth or omit localized phenomena that CFD resolves. Because the surrogate is periodically retrained against updated CFD output, however, this loss of resolution does not accumulate unbounded error. The system instead continuously realigns the fast, low-resolution surrogate with the evolving high-fidelity physics, bounding drift through simulation-driven updates.

3.3.8 On-line Inference Performance

A critical advantage of the PC-R approach is the computational efficiency of on-line inference. Once trained, each per-point PC-R model evaluates a linear regression requiring only 12 multiplications — one per input wind measurement — and 12 additions, summing the 12 intermediate results and the bias term. The cost of an inference is therefore set by the form of the model rather than by the machine that

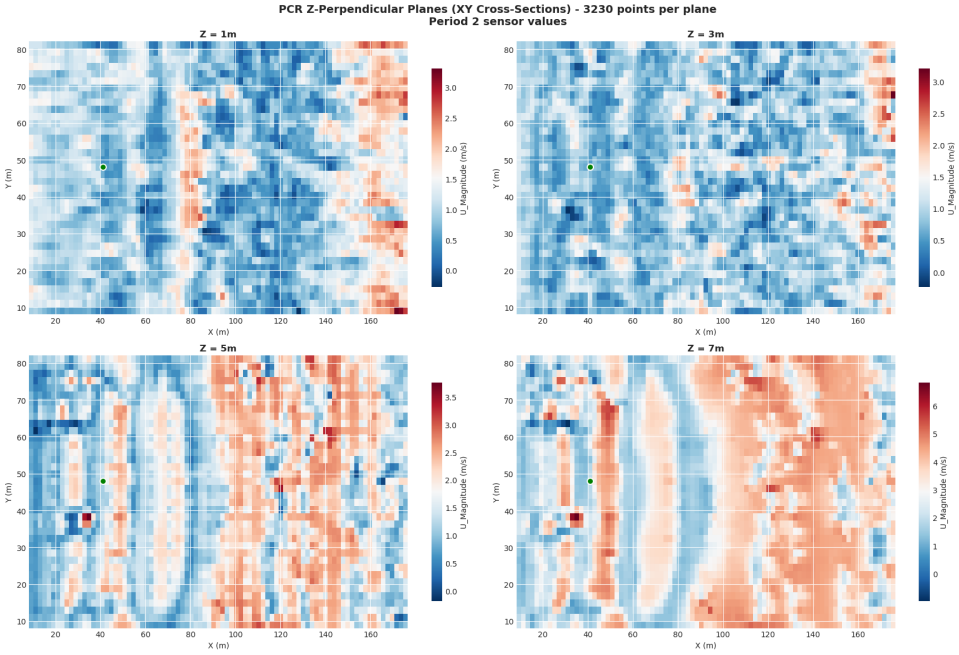

Figure 3.11: Top view (XY plane) of PC-R wind speed predictions at four height levels ($Z = 1, 3, 5, 7$m).

evaluates it. PC-R computes predictions for all 12,920 spatial locations in less than one second, compared with approximately 22.73 minutes for the corresponding Simulation Stage — a latency reduction of at least 1000×. This margin is what enables the on-line path to provide responsive spatial inference while high-fidelity model generation proceeds asynchronously.

The low computational requirements of PC-R inference make the approach particularly suitable for edge deployment scenarios, where predictions can

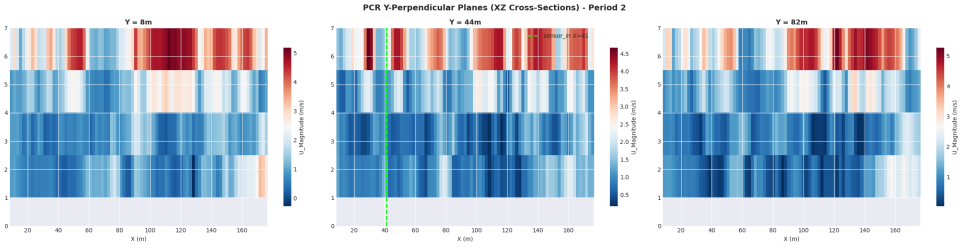

Figure 3.12: Side view (XZ plane) of PC-R wind speed predictions at three lateral positions 8, 45, and 82 meters from the west boundary of the CUPS.

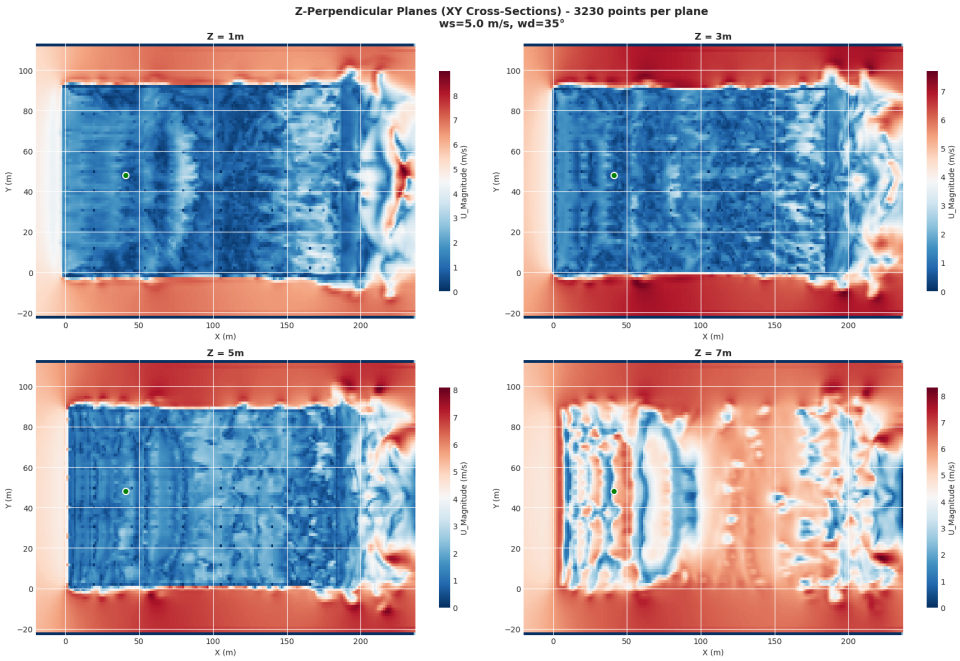

Figure 3.13: Top view (XY plane) of CFD wind speed results at four height levels ($Z = 1, 3, 5, 7$m).

be generated directly on resource-constrained hardware at the farm site. Generating a complete horizontal plane visualization requires querying approximately 3,230 grid points and takes less than one second, providing growers with interactive/low-latency spatial visualization and immediate access to actionable wind field information. Both figures are upper bounds measured on the same cluster that executes the off-line loop, using an untuned evaluation path that reads each point's coefficients individually.

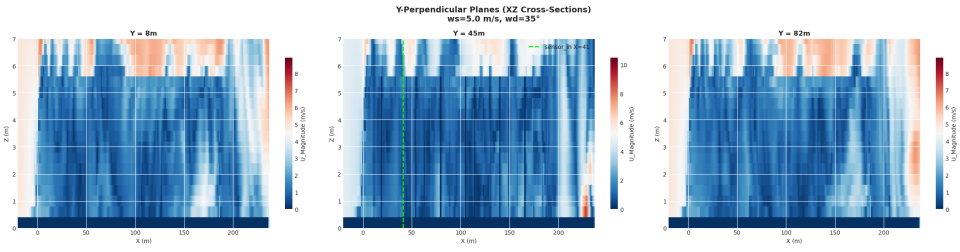

Figure 3.14: Side view (XZ plane) of CFD wind speed results at three lateral positions 8, 45, and 82 meters from the west boundary of the CUPS.

They establish that inference at this scale is feasible at all rather than characterizing the tier it is deployed to — the on-site Intel NUC cluster and the Raspberry Pi 4 units of Section 3.1.2. Chapter 4 reports the measurement on that hardware directly, with the deployed inference path and the model in its served form; the figure there is smaller by more than two orders of magnitude despite the weaker hardware, because the cost measured here is dominated by how the coefficients are read rather than by the arithmetic (Section 4.3.4.3).

3.4 Discussion

The results above validate the on-line/off-line architecture for real-time environmental inference in controlled-environment agriculture. They confirm the central architectural decision — decoupling computationally intensive physics-based modeling from lightweight, low-latency inference — by showing continuous operation at meter scale with best-effort refresh of model fidelity. Two properties carry the argument. First, the off-line loop completes in approximately 31 minutes (Table 3.4), and the comparatively short training phase means updated models are deployed promptly once new simulation output is available, minimizing the window in which the system serves a stale model. Second, on-line inference across all 12,920 points completes in sub-second time, more than three orders of magnitude faster than the CFD simulation it approximates.

Several limitations qualify these results. They concern the origin of the training data, the breadth of what was evaluated, and the conditions under which the evaluation was conducted. None of them contradicts the architectural claim above, but each bounds how widely it should be read.

The most significant is the reliance on CFD as the source of high-fidelity training data. Although the CFD model is validated against in situ measurements in

Chapter 2, it remains an approximation of the physical processes inside the structure. Modeling assumptions and inaccuracies in the simulation propagate into the surrogate, which is trained directly on simulation output rather than on measurements.

The evaluation also covers a single surrogate technique and a single predicted quantity. PC-R offers interpretability and low training and inference cost, and it is well matched to the strong temporal autocorrelation of wind observations, but a linear model cannot capture fine-scale nonlinear dynamics in highly turbulent regions. The observed worst-case errors, on the order of 2 m/s, are concentrated in the Z=5 and 7m planes rather than in the lower canopy where frost mitigation and spray timing decisions are made; at Z=1 and 3m error remains within sensor measurement uncertainty, so the practical effect on decision quality is small. Chapter 4 revisits this limitation by evaluating richer surrogate families alongside PC-R.

Finally, although the architecture is designed for distributed edge deployment, this evaluation emphasizes off-line loop performance. It does not fully characterize inference behavior under severely constrained edge resources or prolonged network disruption. Chapter 4 takes up the first of these, measuring inference cost on the Raspberry Pi hardware deployed at the site.

3.4.1 Cost of Sustaining the Cadence

The loop demonstrated above runs continuously, but its cadence is purchased with dedicated hardware. The 31-minute refresh interval of Section 3.3.4 was measured on eighteen virtual machines held for the duration of the experiment, and the Simulation Stage that dominates that interval consumes essentially all of them. A deployment that owns capacity of that size outright can sustain the cadence indefinitely; a deployment that does not must obtain it from somewhere else.

Enlarging the dedicated allocation is the obvious remedy, and it is the one that scales worst. Capacity provisioned for the worst case sits idle in the common case, and the cadence bought is linear in the hardware while the conditions that make a refresh worth performing arrive at a rate no amount of hardware influences. The alternative is no simpler: a shared facility supplies capacity at no fixed cost, but its queueing delay ranged from zero to twenty-four hours over the course of this work (Section 3.3.3) — far longer than the staleness horizon the loop exists to stay inside.

The approach Chapter 4 takes instead is to use the shared facility without depending on it. Dedicated resources continue to supply a floor on the refresh rate while jobs sent to a shared supercomputer raise it whenever they happen to run, so a long queue delay lengthens the interval between model updates but never stops them. Because the capacity those jobs consume is shared with other users and charged against a finite allocation, Chapter 4 also asks which of them are worth submitting at all.

3.5 Related Work

3.5.1 Distributed Edge–HPC Frameworks

Supporting real-time inference alongside computationally intensive modeling requires coordinated use of edge, cloud, and HPC resources. Santillan and Abad [114] analyze hundreds of real-world cloud architectures to characterize how HPC and edge services are integrated, highlighting common patterns in storage, orchestration, and machine learning workflows. Knebel et al. [76] survey digital twin adoption in the Oil and Gas industry, identifying cloud and edge computing as critical enablers while noting that security, data privacy, and operational complexity continue to limit large-scale deployment. Recent work on environmental and agricultural digital twins shares

similar goals, but often assumes centralized execution or static model updates, limiting responsiveness in operational settings.

A parallel line of work builds large-scale sensor deployment infrastructure. The xGFabric software stack described above is designed to run natively (on microcontrollers), as a runtime using Linux, or as a containerized guest, which makes it complementary to infrastructure such as Sage [27, 111], which includes Waggle [137] as a software stack, or the Array of Things [34]. It differs from these approaches in that it is full-stack (including a network-transparent dataflow programming environment), it includes support for managing HPC workloads, and it targets 5G/6G wireless infrastructure at the edge. Interfacing xGFabric to Sage via Waggle is the subject of our future research efforts.

Complementary work has focused on programming models and runtime systems that simplify development across heterogeneous resources. Ekaireb et al. [45] introduce a distributed dataflow framework for edge–cloud continuum computing, and Wolski et al. [140] explore dataflow as an intermediate representation for portable edge deployments. While these efforts provide essential building blocks for distributed execution, they do not directly address adaptive retraining, model staleness, or closed-loop coordination between inference and simulation.

In summary, prior work has advanced physics-based simulation, surrogate modeling, data-driven prediction, and distributed computing largely in isolation. The first three strands are surveyed in Section 2.5; the fourth is the subject of this section. The system described in this chapter integrates these strands into a self-adaptive, closed-loop edge–HPC architecture that explicitly manages model staleness, retraining latency, and distributed execution under realistic connectivity and scheduling constraints. To our knowledge, this work represents the first end-to-end deployment and empirical evaluation of such a system in a production-scale controlled-environment agriculture

setting.

Chapter 4

Optimization: Reverse Backfill for Asynchronous Edge–HPC Model Improvement

Note: Text and figures in this chapter are adapted in part from: Kurafeeva et al., "Asynchronous Scientific Workflows for Simulation-Driven Inference Across the Edge–HPC Computing Continuum," Future Generation Computer Systems (Elsevier), special issue, 2026, in submission. The decision study of Section 4.3.5 is reported here over a longer deployment window than in that publication, which covers a smaller subset of it.

This chapter presents RBF (Reverse Backfill), an asynchronous scientific workflow architecture that coordinates simulation, training, and inference across edge, cluster, and shared HPC resources. The on-line/off-line loop established in Chapter 3 delivers updated surrogate models to the edge as HPC jobs complete, but raises a cost-effectiveness question: each HPC execution consumes thousands of core-hours, and an individual run does not always improve the currently deployed model. RBF addresses this by treating HPC as an eventually consistent model-improvement engine,

decoupling edge inference from simulation timing through a distributed Workflow Artifact Manager (WAM). This chapter characterizes surrogate model accuracy as a function of staleness and derives decision-time heuristics that predict at the head of the batch queue whether a queued job should proceed — eliminating all degrading releases while avoiding roughly 89% of the HPC allocation that unconditional release would consume.

4.1 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 4.1 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 4.1 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 4.2. The edge tier (left) generates sensor data,

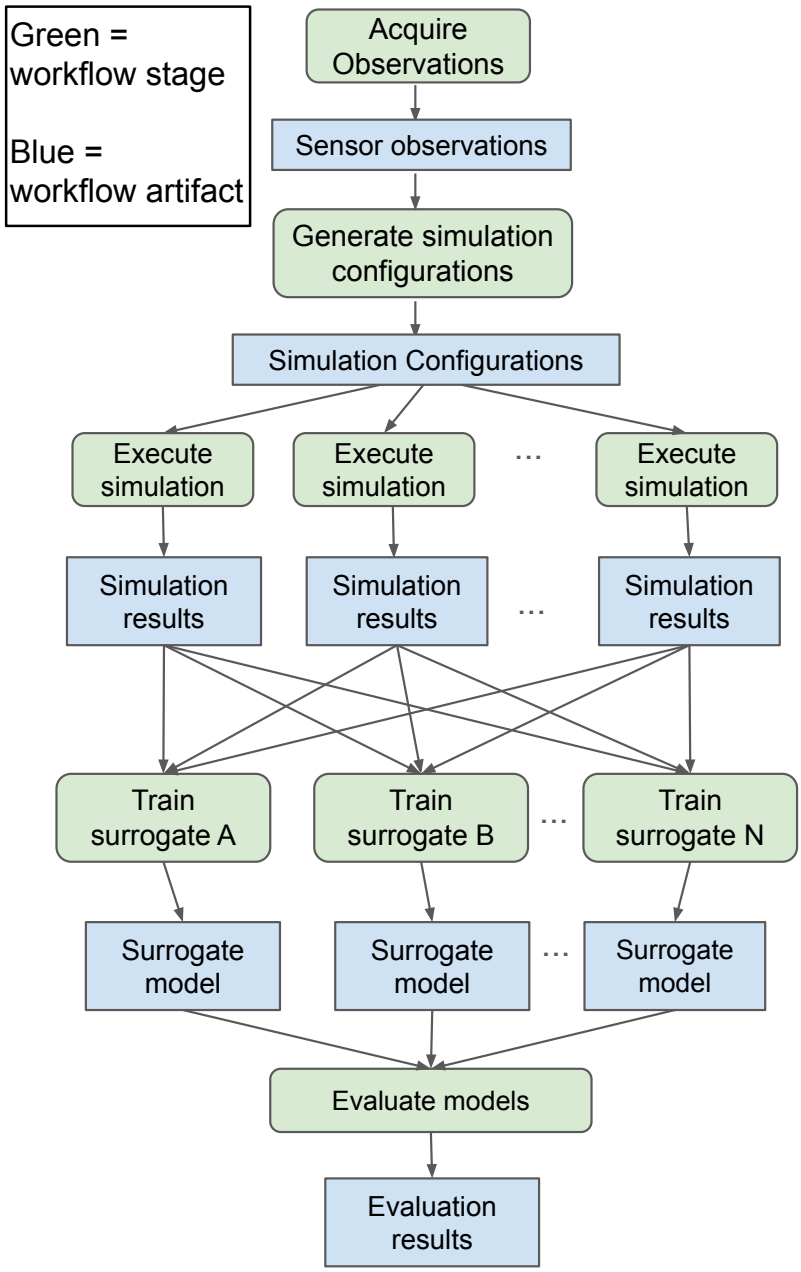

Figure 4.1: RBF Workflow. One iteration of the continuously executing simulation-driven workflow. Workflow stages (green) consume and produce persistent workflow artifacts (blue). Parallel simulation stages generate simulation artifacts that are used to train multiple surrogate models, which are subsequently 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.

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.

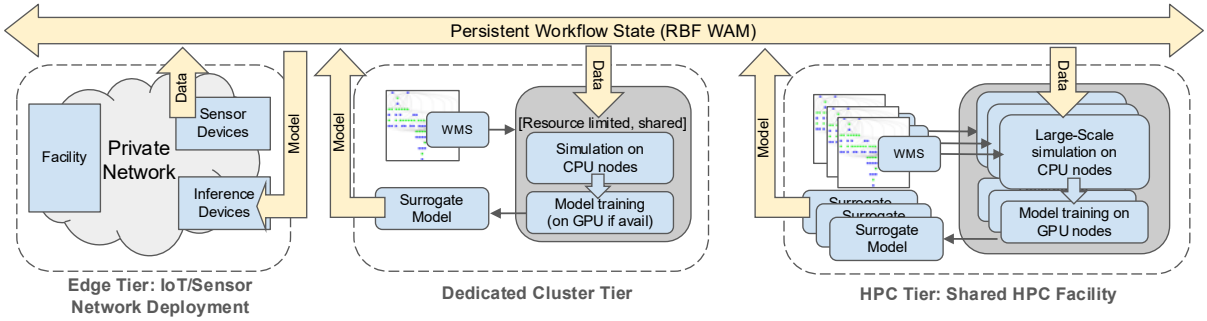

Figure 4.2: RBF System Architecture. The Reverse Backfill architecture spans three tiers connected through the Workflow Artifact Manager (WAM), which provides persistent workflow state and versioned workflow artifacts across the computing continuum. (Left) At the remote facility, sensor observations 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.

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.

Two properties follow from this arrangement, and together they are what separate RBF from continuous learning systems: inference never blocks, and model fidelity improves over time. In those systems models are updated directly from streaming observations, whereas in RBF model updates are mediated by simulation, making update latency and variability a fundamental design consideration. 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.

4.1.1 Edge Inference Tier

Note: The edge inference tier described here was built and instrumented in part by Benjamin C. Carter of the UCSB RACELab, under the guidance of the author; it is evaluated in Section 4.3.4 [77].

The edge inference tier ingests sensor data continuously and answers queries from the most recent model it has, without waiting on anything upstream. 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.

The age of the model an edge service is serving — its *staleness* — is the quantity the remainder of this chapter is organized around, and RBF measures it from two reference points. The two are defined in the Glossary: $staleness_p$, the time elapsed since the model was published to the edge, which each new deployment resets to zero; and $staleness_c$, the time elapsed since the most recent sensor record used to train it, which a new deployment reduces only to the compute time of the iteration that produced the model. The two differ by that compute time. Because RBF draws models from tiers whose compute times differ and vary — a dedicated cluster and an opportunistically scheduled shared HPC facility — the two measures can diverge substantially, and each result in this chapter states which of them it reports. The decay characterization of Section 4.3.2 is expressed in $staleness_p$, measured from the moment a model reaches the edge; every run in that characterization incurs the same pipeline compute time, so the two measures differ there only by a constant. The opportunistic-execution heuristics built on those curves are expressed in $staleness_c$, measured from the training cutoff, so that

models arriving from tiers with different compute times are placed on a common clock. Under either measure, the accuracy of a served model degrades as its 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 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.

4.1.2 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.

Because the modeling choice is not fixed at design time, the accuracy–latency–cost tradeoff becomes something the deployment can measure rather than assume — Section 4.3.1 trains a physics-informed network, an operator-learning method, and a linear regression in the same pipeline. 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.

4.1.3 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 whenever 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.

Nothing in this arrangement requires a change on the HPC side: no scheduler modification, no priority exemption, and no real-time deadline imposed on a batch facility. 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.

4.1.4 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.

Persistence is what makes the arrangement survivable: because every artifact outlives the stage that produced it, a failed or canceled stage costs its own work and nothing else. 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.

4.2 System Implementation

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. The paragraphs that follow describe how one pipeline instance is timed and how instances overlap; the subsections then give the facility itself, the artifact layer through which its stages communicate, and the wireless networks that carry data to and from it.

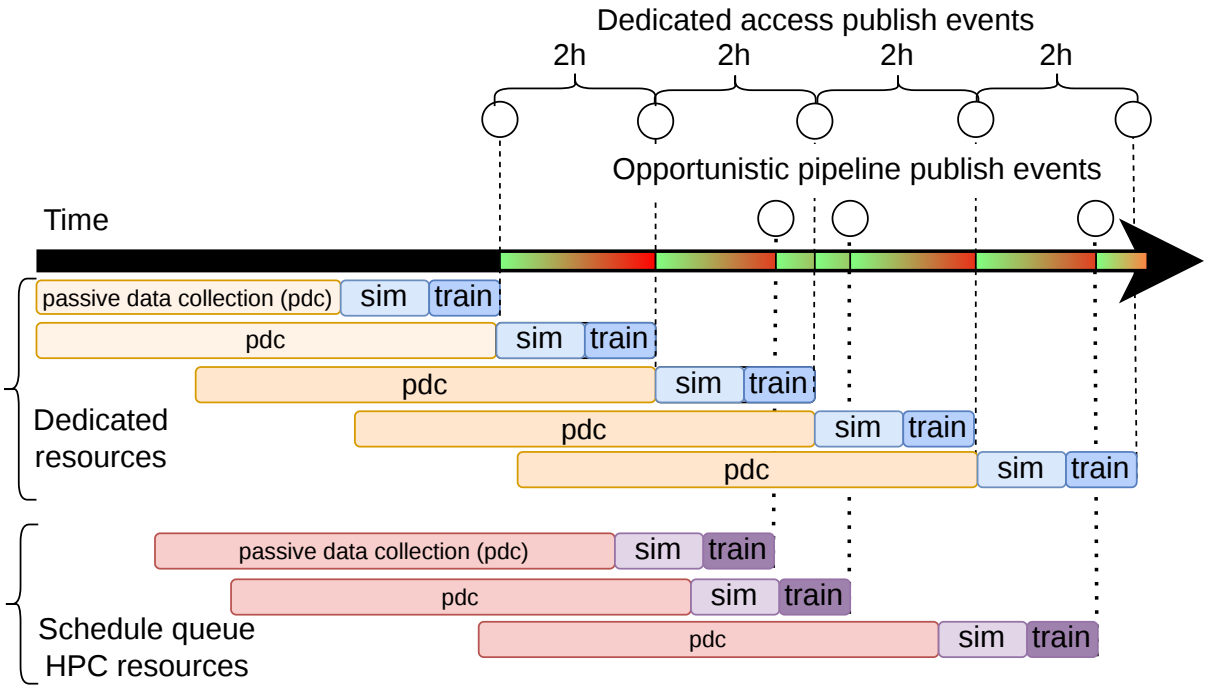

Figure 4.3: Timeline of the RBF instantiation illustrating asynchronous, simulation-driven model updates. Passive data collection (pdc), simulation (sim), and training (train) stages overlap across multiple pipeline instances, while model updates are published opportunistically upon completion. This approach enables continuous inference despite irregular and delayed HPC execution.

As shown in Figure 4.3, 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, low-latency inference while mitigating model staleness under irregular and delayed HPC-driven model updates.

A set of simulations (*sim*) is launched in parallel on dedicated resources using data collected up to the time of execution (denoted *pdc* (for passive data collection) in Figure 4.3). 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 72-machine 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 at that scale. The bound is set by the width of the dedicated tier rather than by the workflow: provisioned with 18 machines instead of 72 — one head node and 17 workers — each worker executes up to five simulations in sequence rather than one, and the same pipeline yields models three to five hours old, depending on the complexity of the wind conditions the simulations must resolve.

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 mitigates 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.

4.2.1 RBF Evaluation Facility

We refer to this deployment — the CUPS screenhouse of Chapter 2, instrumented and running the full workflow — as the *Evaluation Facility.* It makes continuous meteorological inferences for the 4-acre structure, using a network of sensors that collect wind speed, wind direction, temperature, and humidity inside and outside the screenhouse. Measurements parameterize high-fidelity CFD simulations that model airflow within the screenhouse; the resulting simulation outputs train surrogate models for low-latency inference.

The *sim* stage runs the validated OpenFOAM CFD model of Chapter 2, unchanged. That model, its mesh, its solver configuration, and its validation against in situ measurements are described in Section 2.2; RBF supplies it with boundary conditions derived from the sensor data and consumes its output. The prototype implements the stage as 72 such simulations executing in parallel, one per machine of the dedicated cluster

tier.

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 4.3.2). 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 over the full CFD mesh, which is then resampled onto 9690 inference points inside the CUPS. For this experiment we restrict the inference grid to the three canopy-level heights ($Z = 1$, 3, and 5m) of the subgrid defined in Section 3.2.2, omitting the above-roof plane, which gives $85 \times 38 \times 3 = 9690$ points.

Simulation outputs are used to train multiple surrogate models, including a physics-informed neural network (PINN) [32, 107], a Fourier neural operator (FNO) [84], and a principal component regression (PC-R) [74] 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 [96]. RBF runs *sim* and *train* stages using a RADICAL-Pilot [91] 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 [109]), 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.

4.2.2 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 [139] 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 [22]). RBF relies directly on the append-only structure of these logs: a job canceled before it computes leaves the state it would have consumed unchanged, so cancellation does not impede workflow progress.

4.2.3 Private 5G Network

Note: The private 5G networks described here are the work of our collaborators at the University of Nebraska–Lincoln, and their edge-side deployment and the slicing evaluation of Section 4.3.4.2 were carried out with Benjamin C. Carter of the UCSB RACELab [77]; model dissemination to the edge traverses these links.

The first of the two private 5G networks on which we deploy RBF is a commercial cell sited at the agricultural facility, and it carries the empirical evaluation reported in this chapter. It uses a Celona AP25-48 cell [11], an off-the-shelf commercial solution for 5G private cellular networking, backhauled over Starlink [13] satellite Internet service because the site is rural. This network is an example of commercially available 5G technology designed for outdoor deployment in remote areas and long-term unattended operation.

The second is a laboratory-sited private 5G network built from two open-source components: srsRAN Project [126] for the radio access network (RAN) and Open5GS [101] for the 5G core, both deployed as Docker containers on a dedicated edge host. The base station is driven by an Ettus Research USRP B210 software-defined radio operating on 3GPP Band n78 with a 40 MHz channel in TDD mode, and the core implements a full 5G Standalone architecture. This cell captures forward-looking capabilities—network slicing in particular—that are not yet available commercially in a weatherproof package for unattended outdoor deployment.

Two Raspberry Pi 4 Model B single board computers serve as user equipment, each fitted with a Quectel 5G modem and a programmable SIM registered in the Open5GS subscriber database. The two 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 latency. The *general data path* carries all other network data from other users or applications. Because the two share the same radio link, contention between them can degrade model transfer latency, which motivates the use of network slicing to isolate the flows; we evaluate this in Section 4.3.4.2.

4.3 Results

We next use the Evaluation Facility and CUPS application to empirically evaluate RBF end-to-end. The instantiation just described is what is under test: whether RBF transparently tolerates delayed and irregular model-update timings while sustaining continuous inference through asynchronous and opportunistic model improvement. 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 evaluate the deployment of the inference models onto edge devices, showing that they can be delivered and executed with minimal overhead. Finally, we examine the value of a *single* opportunistic run and develop the decision-time heuristics that predict, at the head of the batch queue, whether each one should proceed.

4.3.1 Dedicated-Access Pipeline Performance

The dedicated cluster and GPUs of the Dedicated Cluster Tier — which we call the "dedicated-access pipeline" — comprise 72 machines, each executing 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, PC-R 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 37.4 ± 7.6 minutes of CFD computation across the 72 nodes — longer than the 29.6 ± 5.5 minutes a single simulation case requires, because the stage completes only when the last node does — and 14.2 ± 4.1 minutes for data transformation of simulation outputs (for use in

model training). The *train* stage executes PINN, FNO, and PC-R 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 PC-R 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.

4.3.2 Inference Model Accuracy Decay

The 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

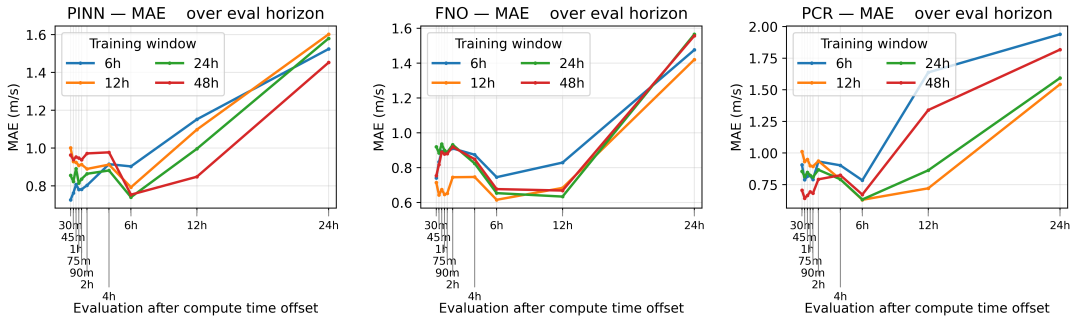

Figure 4.4: Model accuracy decay over time using different history windows for all three models (PINN, FNO, PC-R). The x-axis shows elapsed time after a model becomes available at the edge ($staleness_p$; see the Glossary), ranging from 30 minutes to 24 hours. The y-axis shows *cumulative* MAE (m/s): 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.

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?

The first is read off the decay curves presented in this subsection; the second is taken up in Section 4.3.3, where additional execution tiers raise the publication rate. Figure 4.4 shows the decay of model accuracy over time for all three models (PINN, FNO, PC-R). 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 4.3.1; a model is never fresher than the one in service. In the terminology of the Glossary these curves are plotted against $staleness_p$. Every run characterized here incurs the same pipeline compute time, so the corresponding $staleness_c$ is obtained by adding that fixed offset of approximately 134.8 minutes. The analyses that build on these curves later in this chapter are expressed in $staleness_c$, which places models arriving from tiers with different compute times on a common clock. 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 4.4 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 PC-R 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 4.3.1, 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 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.

4.3.3 Opportunistic Model Regeneration to Improve Delivery Cadence

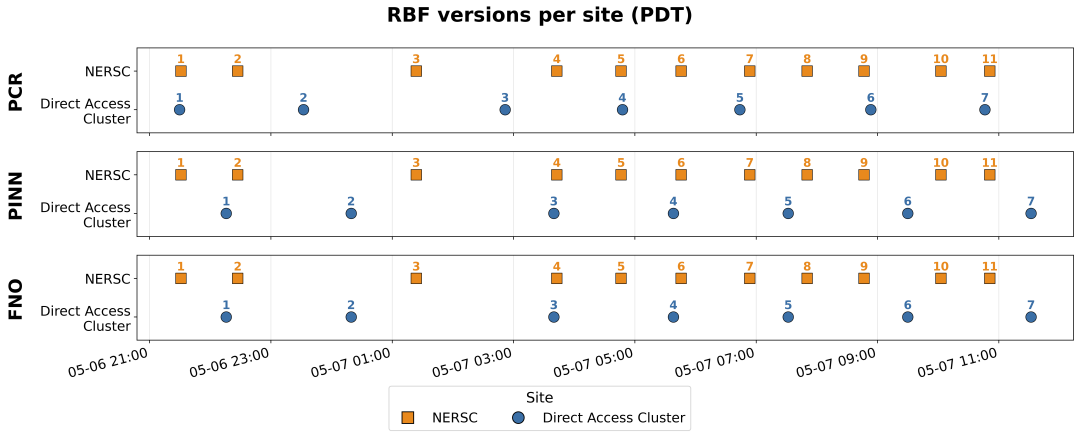

Figure 4.5: Timeline of model publish events for all three model types (PINN, FNO, PC-R) during a simultaneous live experiment on two resources: the dedicated-access cluster and the NERSC batch-queue HPC.

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 4.4, 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 end-to-end pipeline time reported in Section 4.3.1 for the pipeline of Figure 4.3). 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. This holds for the operating mode described here, in which one queued job runs back-to-back iterations for the duration of its allocation: the job is committed or canceled as a whole, so the allocation is the unit whose benefit is at issue, and the staleness it removes across that span is the measure of that benefit. Section 4.3.5 treats the separate case in which each job produces a single iteration, where the benefit of any one job is not assured.

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 4.3.5.

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.

The physical setting bounds how large that benefit can be: 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 [40], 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 measure the resulting cadence in a live multi-site experiment. Figure 4.5 shows model publish events for PC-R, 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; PC-R events are offset from PINN and FNO because PC-R 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 wait for a new allocation is approximately 17 hours and varies little from one allocation to the next, creating gaps of at least 18 hours during 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-node training jobs [97]. We execute the training stage on CPU nodes rather than on the GPU partition. GPU training is faster in isolation, but the CPU partition is far less contended, which both simplifies scheduling —the training stage does not have to wait on a second, heavily oversubscribed queue—and reduces the effect RBF has on other users of the shared facility. However, as shown in Figure 4.5, 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).

Read together, the two event streams of Figure 4.5 keep the effective model age — $staleness_p$, measured from publication — below 2 hours in the large majority of cases. Doing so keeps model accuracy, across all three model types, below the upper end of the sensor measurement error range, 0.88 m/s, over the interval a model is served. 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 4.1: Minutes between FNO model publish events, measured while a NERSC allocation is executing; queue waits between allocations are excluded

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

Table 4.1 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. These intervals characterize the cadence RBF sustains during an active allocation, which is the operating mode described above: once a queued job begins executing, it produces model generations back-to-back for the duration of its allocation. The 17-hour wait for the *next* allocation falls outside these intervals; during such a wait the combined stream reverts to the dedicated pipeline's 134.8-minute cadence, so the dedicated cadence is a floor that opportunistic execution raises rather than replaces. 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 4.4, increasing the publication rate therefore directly improves the expected accuracy of the deployed surrogate models.

4.3.4 Edge Tier Performance

Note: The experiments in this subsection were run, and their data collected, by Benjamin C. Carter of the UCSB RACELab, with the laboratory private 5G network and its slicing configuration provided by our collaborators at the University of Nebraska–Lincoln. This work was carried out under the guidance of the author, who performed the analysis and prepared the presentation given here [77]. The results are reported because they bound how quickly a published model reaches the edge.

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 form the delivery stage of the end-to-end workflow, bounding how quickly improvements generated by upstream simulation and training become available for operational inference.

This subsection evaluates those 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 [126] radio access network and Open5GS [101]. 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.

4.3.4.1 Satellite Long-Haul and Cellular 5G at CUPS

When a new model is published via the WAM, edge-inference devices fetch it and update the locally available model. Each surrogate model traverses a

Starlink [13] 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 [11], 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.

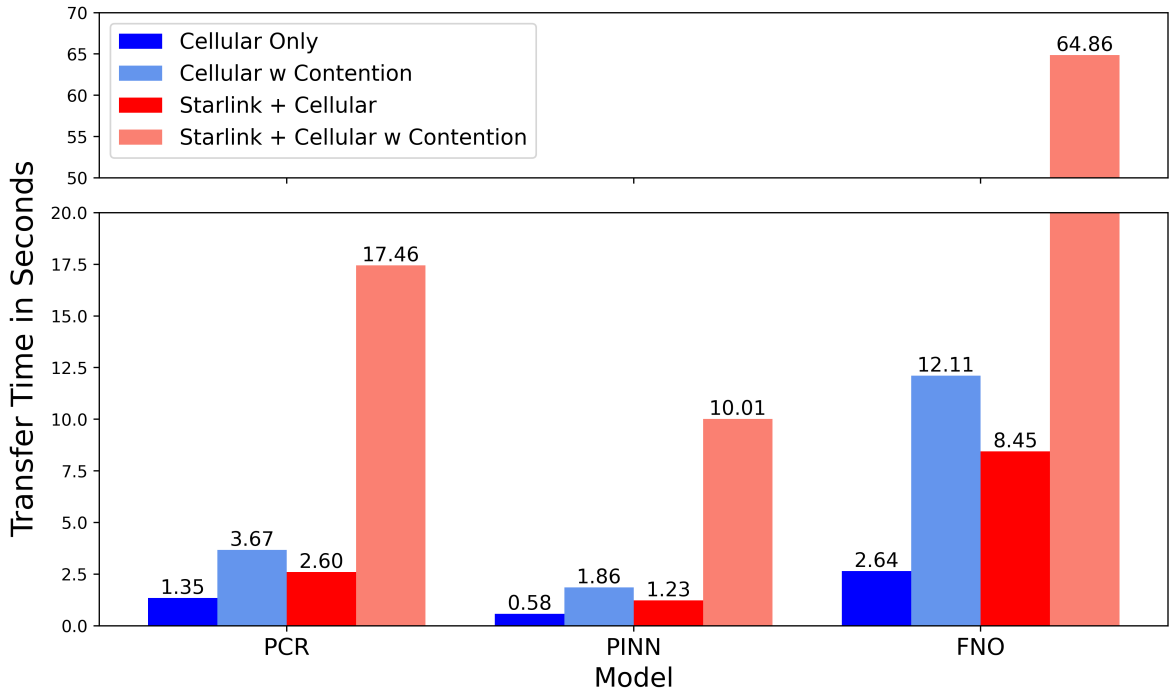

Figure 4.6: P-95 Model Transfer Time of 100 runs. Transfer time is in seconds. Each bar represents a different network path (cell only or Starlink and cellular) for model delivery. Note the gap between 20 and 50 seconds only contains the FNO bar.

Figure 4.6 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. PC-R 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 125 seconds later—60 seconds to detect the publication and 65 seconds to transfer it—with confidence 0.95. As described in Section 4.3.3, sensor data updates every 5 minutes so model delivery of a "fresher" model falls well within the data sampling interval. The sampling interval also sets the fastest publication cadence the workflow could usefully sustain, since a model trained on data that has not changed carries no new information. Even at that limit — one publication every 300 seconds — detection and transfer together occupy at most 125 of those seconds, so a retrieval always completes before the next model is published and the edge is never more than one publication behind. 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.

4.3.4.2 Laboratory 5G Network Slicing Evaluation

The laboratory cell bounds the worst case for model dissemination under contention. On that cell, which implements the OpenRAN [12] standard, two slices are configured: an enhanced mobile broadband (eMBB) slice spanning the entire radio spectrum, and a second slice capped at 10% of available Physical Resource Blocks (PRBs). One Raspberry Pi 4 on the eMBB slice repeatedly fetches the PC-R, PINN, and FNO models from a remote WAM host, while a second on the capped slice saturates the RAN with UDP traffic to generate contention.

Table 4.2 shows the mean throughput over 100 runs. Without slicing, con-

tention cuts throughput by 20–23%; with slicing, that cut falls to 6% for PC-R and 2% for FNO and disappears entirely for PINN, and the cost of slicing when uncontended is only 0.4–6.6%. The residual 6% for PC-R is likely noise rather than a slicing effect, since both a larger model (FNO) and a smaller one (PINN) stayed within 2%. The practical conclusion is that commercial off-the-shelf 5G infrastructure is already sufficient for edge model retrieval given the small size of these models, and that open-source private 5G networks additionally offer isolation that lowers the worst-case transfer latency.

Table 4.2: 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.**
PC-R	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%

4.3.4.3 Cost of Inference on Edge

Table 4.3: Execution time (in seconds) for an inference on all points within CUPS using a Raspberry Pi 4. We report the P95 across 100 trials.

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

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).

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 4.3. We show that initialization for all models takes longer than a single inference, sometimes orders of magnitude longer (8518× for PC-R). In PC-R'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 (PC-R) 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 PC-R initialization, RBF inference time is still well below our target 5 minute threshold.

4.3.5 Predicting the Benefit-Cost Tradeoff

Figure 4.5 and Table 4.1 of Section 4.3.3 illustrate the potential benefit of back-to-back executions on the faster NERSC resource once it begins executing an 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.

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 accepts some "false positive" predictions in exchange for publishing more often and capturing more of the improvement on offer. 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. We refer to this moment throughout as the *head of the batch queue*: strictly, the scheduler has already selected the job and it has begun to execute, but because no simulation or training compute has yet been spent, the allocation is still intact and is returned in full if the heuristic cancels.

More specifically, RBF predicts the value of each *isolated* opportunistic run. Within the dedicated cluster's constant publication cadence (Section 4.3.1), 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. Back-testing here names the evaluation method for the study as a whole, and is distinct from the direction in which an individual model's error is measured within it — a distinction the axes of Section 4.3.5.1 turn on. Unlike the results shown in Section 4.3.3, 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.

Over June and July of 2026 the batch scheduler selected 108 RBF submissions for execution, and we evaluate the decisions the heuristics would have made for each of them. Six carry no scored decision — the first has no predecessor to form a prediction from, two follow an incumbent whose error is numerically zero, and three fall in windows that recorded no sensor readings — leaving 102 submissions that carry both a decision-time prediction and a measured outcome. The deployment continues to run, so the study grows with it. The isolated runs in this window divide almost exactly evenly between those that helped and those that did not, which makes it a demanding test of whether the useful runs can be distinguished from the rest.

As a baseline, allowing each of the HPC jobs to proceed unconditionally (i.e. without a decision by the heuristic) averages −20.1% improvement per model published by each HPC job when compared to the models produced by the dedicated pipeline. Fifty-one of the 102 models were more accurate and fifty-one were less accurate. We also observe severe losses for a small number of runs which overwhelm the many modest gains: across the window, +1,682.6% of accuracy improvement was available in total, against −3,731.2% of avoidable loss. 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.

4.3.5.1 Scoring an Individual Decision

To predict the value of an individual submission, RBF implements an automatic model selection heuristic. Each pipeline trains all three model types (PINN, FNO, PC-R) 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. Both are computed the same way at the decision point, by *front-testing*: each candidate model is evaluated on all sensor observations recorded between the last dedicated publication and the moment the decision is made. For the dedicated side these are the models currently in service; for the opportunistic side they are the models published by the previous opportunistic iteration, since the candidate this job would produce does not exist yet. 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, $[t_{\mathbf{opp}}, t_{\mathbf{ded,\ next}})$:

$$\%\text{improve} = 100 \times \frac{\text{MAE}_{\text{ded}} - \text{MAE}_{\text{opp}}}{\text{MAE}_{\text{ded}}} \tag{4.1}$$

Positive means the opportunistic model outperformed the dedicated-pipeline winner; negative means the run made the model published to the edge worse. The same expression, applied to each opportunistic model type individually rather than to its winner, yields a *per-type* predicted %improvement, and both heuristics below are built from those three per-type estimates.

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. The assumption is justified by the model's *decay curve* (Section 4.3.2) — its error as a function of staleness — which is typically flat over a window this short (under either staleness measure, since the two differ only by a constant). Comparing the two winners in terms of recent error therefore yields a predicted %improvement.

The percentage-improvement quantity appears twice in the analysis—once as a *prediction* formed at the head of the queue, and once as a *realized outcome*

Table 4.4: The four choices that make the percentage-improvement quantity concrete, and the options considered on each. Axes A and B feed the prediction formed at the head of the queue; axes C and D feed the realized outcome used to grade it. The setting reported throughout this section is set in bold. Axis B carries two bold entries because it belongs to the rule rather than to the grading: both inferences are computable at the head of the queue, and the two heuristics of Section 4.3.5.4 make different choices on it. Statistics quoted for the prediction itself use B3.

Axis	**Quantity fixed**	**Options considered**
A	Incumbent winner	(A1) backtest over the incumbent's own service period so far; (A2) backtest over a fixed two-hour trailing window; (A3) **front-test to the decision point**
B	Predicted opportunistic winner	(B1) backtest the predecessor against the preceding dedicated publication; (B2) **the predecessor's own realized-outcome winner**; (B3) **front-test the predecessor to the decision point**
C	Outcome window	(C1) **to the next dedicated publication**; (C2) through the second following dedicated publication
D	Realized opportunistic winner	(D1) the type axis B predicted, fixed at decision time; (D2) the job's own backtest winner; (D3) **the type that performed best over the outcome window**

used to grade that prediction after the fact. Both use the expression above and differ only in which MAE values feed it. Making them concrete requires four independent choices, which we treat as orthogonal axes and evaluate exhaustively; Table 4.4 lists them together with the options considered on each. Axis A fixes which of the three dedicated-pipeline model types is taken to be the model in service, and feeds both sides of the comparison. Axis B fixes which type the queued job is expected to produce; because the candidate models do not exist yet, this can only be inferred from the previous opportunistic iteration, and it consumes no information unavailable at the head of the queue. Axis C fixes the interval over which the realized outcome is graded, and axis D fixes which of the three models the completed job actually publishes is credited with that outcome. The four axes yield $3 \times 3 \times 2 \times 3 = 54$ combinations, each of which we score over the same replay.

One property of the reported setting deserves to be stated explicitly. Crediting the completed job with whichever of its three models performed best over the outcome window—the axis D setting used here—requires hindsight, and a deployed system cannot make that selection. It is the right choice for measuring the *value that was available* from a given opportunistic run, and it is never consumed by any decision rule: no rule reads the realized winner, the realized improvement, or the correctness of its own prediction. It does mean the realized improvements are an upper envelope on what a fielded system would capture. The two implementable alternatives on axis D leave the sign accuracy of the prediction unchanged; what they change is the scale of the realized losses, which grow several-fold when the predicted type is fixed at decision time or when the job's own backtest winner is used instead. The opportunistic resource looks considerably worse when the edge cannot pick the best of the three published models, which is an argument for the winner-selection mechanism above rather than against the analysis.

4.3.5.2 Measured Outcomes

Figure 4.7 shows how those 102 outcomes are distributed. Modest gains and occasional severe losses recur throughout the window rather than clustering in any one stretch of it, so the asymmetry noted above — +1,682.6% of improvement available against −3,731.2% of avoidable loss — is a property of the deployment rather than of one bad stretch.

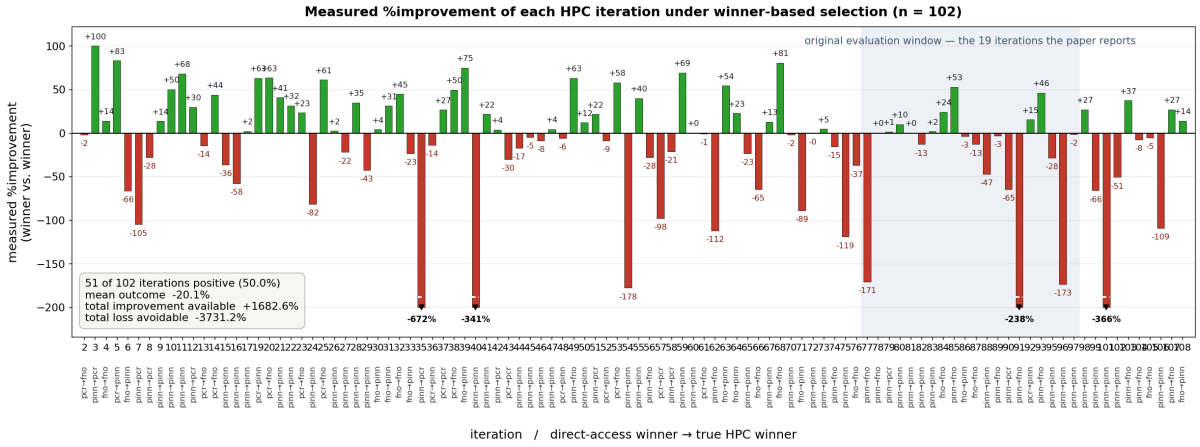

Figure 4.7: Realized %improvement of each opportunistic publication over the study window, under the winner-based selection described above. Green bars improved the deployed model and red bars degraded it; four bars are truncated at −200% and annotated with their true values. The shaded region marks the 19 decisions of the window reported in the associated publication. The axis label beneath each bar gives the incumbent winner type and the realized opportunistic winner type for that decision.

The pattern holds within each model type as well (Figure 4.8). Scored individually rather than through the winner, FNO is positive on 48 of the 102 decisions with a mean of −47.3%, PINN on 53 at −30.5%, and PC-R on 44 at −28.0%. No type is reliably the one worth publishing, which is what makes the winner-selection mechanism necessary and is also why the three per-type predictions carry independent information for the agreement-based rules below.

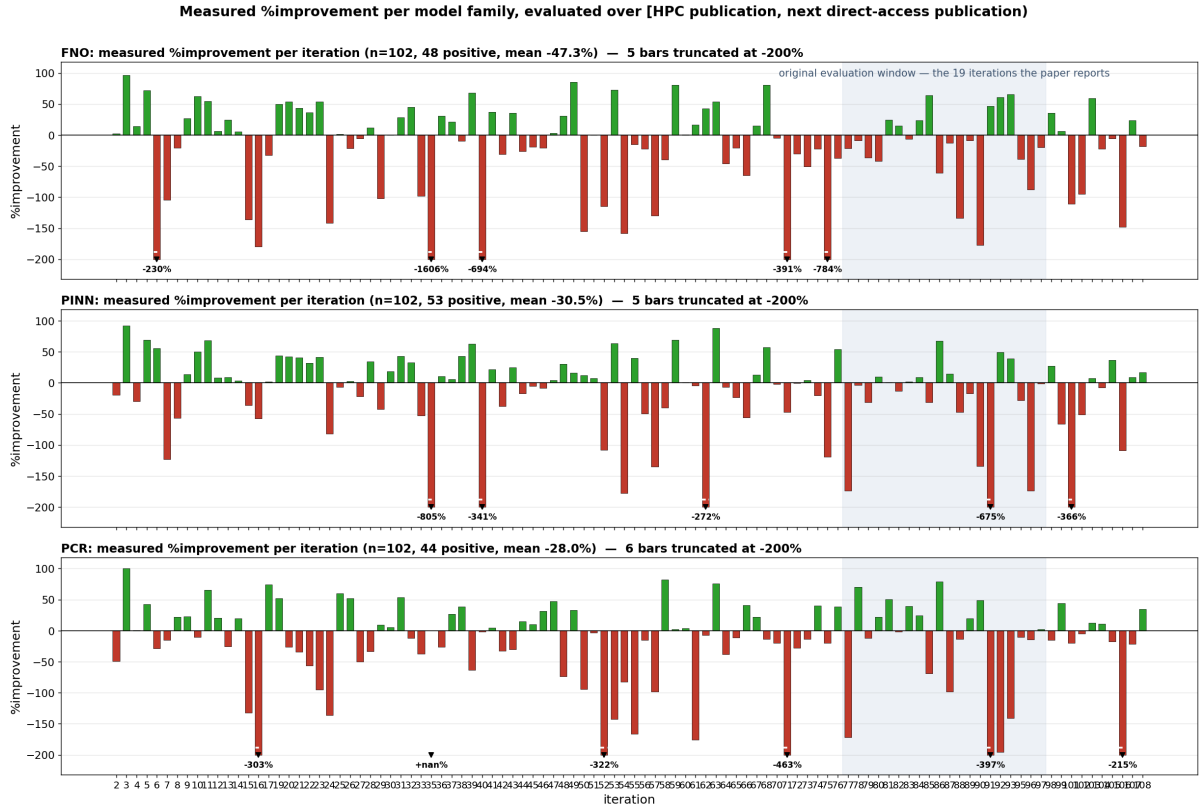

Figure 4.8: Realized %improvement per model type over the study window, evaluated over the same outcome window as Figure 4.7. Each type is scored against the incumbent independently, without winner selection. Bars beyond −200% are truncated and annotated. The winner-based mean of −20.1% is better than that of any single type (−47.3% to −28.0%), and the difference is the value contributed by selecting a winner at all.

4.3.5.3 Candidate Decision Rules

Every rule rests on a single estimate: the sign of $\hat{g}$, the predicted %improvement computed from recent per-type error as described above. Treated as a binary classifier over the 102 decisions, that estimate predicts the sign of the realized improvement correctly 63.7% of the time. The outcomes are evenly divided—51 of the 102 decisions improved the deployed model—so a trivial predictor that always answers with one class is only 50% accurate, leaving the estimate an edge of 13.7 percentage points over that floor. Under the other decision-time inference on axis B—taking the predecessor's own realized-outcome winner rather than front-testing it—the same estimate is 61.8% accurate, an edge of 11.8 points. An edge in this range is real but modest, and it is the reason the system applies structured heuristics rather than acting on the prediction directly. It is also why the weaker of the two estimates is not automatically the weaker basis for a rule: what a heuristic captures depends on which decisions it acts on, not on its accuracy over all of them.

Table 4.5 reports the candidate rules we evaluated, scored over all 102 decisions. All are built exclusively from decision-time signals: the predicted %improvement, the predicted and incumbent winner types, and the three per-type predictions individually. The realized outcome is used only to grade them. Two families appear. The *agreement* family counts how many of the three model types independently predict a positive improvement. The *shift* family keys on whether the predicted opportunistic winner is a different model type than the incumbent, optionally combined with the spread between the highest and lowest per-type estimate. Rules are grouped by their axis B setting, since the same rule scores differently depending on how the predicted opportunistic winner is inferred.

The table separates the rules worth deploying from the rest. Releasing on

Table 4.5: Decision rules scored over all 102 decisions of the study. "Rel." is the number of jobs released, "Prec." the fraction of released jobs whose realized improvement was positive, "Avg" the mean realized %improvement per released job, and "Total" the sum of those realized improvements over every job the rule releases. "Core-h saved" is approximate savings at ≈6,150 core-hours per canceled run, against the roughly 628,000 core-hours that releasing every job would consume. The oracle is unreachable in practice: it releases exactly the jobs that helped and so captures the most improvement available in total. That total, not the per-run average, is the ceiling it sets — a rule that releases fewer but higher-value jobs can exceed its Avg while capturing less overall. The two rules RBF exposes are set in bold.

Rule	**Rel./102**	**Prec.**	**Avg**	**Total**	**Core-h saved**
Unconditional	102	50%	−20.1%	−2,048.6%	—
Oracle (ceiling)	51	100%	+33.0%	**+1,682.6%**	∼314,000
Axis B: front-test to the decision point (read by No-Regret)					
No-Regret: unanimous, 3 of 3	11	**100%**	**+35.5%**	**+390.4%**	∼560,000
$\hat{g} > \theta$ or unanimous	17	82%	+22.9%	+389.5%	∼523,000
Majority: ≥ 2 of 3	30	70%	+8.1%	+243.9%	∼443,000
Naive: $\hat{g} > 0$	42	67%	+7.4%	+311.0%	∼369,000
Shift, or spread > 125 pp	56	52%	−30.0%	−1,682.6%	∼283,000
Axis B: predecessor's realized-outcome winner (read by High-Yield)					
High-Yield: $\hat{g} > \theta$ or unanimous	29	72%	+15.2%	**+441.5%**	∼449,000
Naive: $\hat{g} > 0$	46	63%	−3.4%	−158.0%	∼344,000
Unanimous: 3 of 3	16	81%	+18.1%	+289.3%	∼529,000
$\hat{g} > 0$ and shift	21	71%	+10.0%	+209.8%	∼498,000
Shift, or spread > 125 pp	53	55%	−23.4%	−1,241.8%	∼301,000

the raw prediction alone—the naive rule—fires on roughly half the decisions and is the weakest filter in the table that pays at all: it averages +7.4% under the front-test inference and −3.4% under the realized-outcome inference, so it is not reliably better than declining the opportunistic resource altogether. The severe outliers of Figure 4.7 are exactly what a classifier with a 14-point edge cannot be relied upon to catch. The shift family is negative under both axis B settings, despite having the more appealing story: a change in winning type ought to indicate that conditions have moved since the job was submitted, and the 125-point spread threshold in particular fares worst of any rule in the table that filters at all. Agreement is the signal that pays, and it pays in proportion to how much agreement is demanded: under the front-test inference, releasing on any single positive prediction averages +7.4%, on a majority +8.1%, on the threshold-or-unanimity disjunction +22.9%, and on unanimity alone +35.5%, with precision rising from 67% to 100% along the same ladder. It is therefore consensus rather than weight of evidence that carries the signal, and the monotonicity of that ladder is itself evidence that the agreement count is informative rather than incidental.

Within the agreement family, the intermediate rules are the natural fallbacks if unanimity proves too rare in a longer deployment. Requiring a majority rather than all three releases 30 jobs instead of 11 and stays positive at +8.1%, though at 70% precision it forfeits the guarantee entirely; it is the setting to reach for when refresh rate matters more than either. The unanimity condition itself is worth reading across the two axis B inferences: the same three-way agreement releases 16 jobs at 81% precision when the predecessor's realized winner is used, against 11 at 100% when it is front-tested. The perfect precision therefore belongs to the pairing of unanimity with the front-test estimate, not to unanimity alone—which is why the axis B choice is reported as part of each heuristic rather than as a property of the study.

The oracle row bounds what is available. With perfect foresight, 51 of the 102

decisions were worth releasing at a +33.0% average, and the best implementable rule in the table collects 26% of the improvement the oracle captures in total, so substantial headroom remains. That best rule is not one of the front-test agreement rules but the threshold-or-unanimity disjunction read against the predecessor's realized outcome—the strongest rule and the strongest guarantee therefore sit under different axis B inferences, which is why the two heuristics below do not share one.

4.3.5.4 The RBF Heuristics

Of the rules evaluated, two fit this system best and bracket the operating range. Both take agreement among the three model types as their primary signal, both act only on information available at the head of the queue, and RBF exposes both as configuration options. They differ in their release condition and in which decision-time estimate of the predecessor they read—the axis B choice of Table 4.4.

The No-Regret heuristic, given in Algorithm 1, allows a job that is ready for execution on the HPC resource to continue only when *all three* model types independently predict a positive %improvement over the current dedicated winner. The intuition behind the success of this heuristic is that agreement across three models with different inductive biases—a physics-informed network, an operator-learning method, and a linear regression on principal components—is difficult to produce by coincidence. When all three agree that the opportunistic side has moved ahead, the signal is far more likely to reflect a real change in conditions since the job was submitted than the reading of any single model.

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 execution, it always yielded an improvement in accuracy (not just in aggregate). In this study, the No-Regret heuristic decided to continue 11 of

Algorithm 1 The No-Regret heuristic, computing the head-of-queue decision for opportunistic iteration n. Both error vectors are measured by front-testing over the same elapsed window W—no quantity depends on the candidate model, which does not yet exist.

Require: $\mathcal{T} = \{\text{FNO}, \text{PINN}, \text{PCR}\}$: model types
Require: $\text{ded}_n[\tau]$: published dedicated model of type τ
Require: $\text{opp}_{n-1}[\tau]$: type-τ model published by the last completed opportunistic iteration
Require: W: sensor data observed between the last dedicated publication and the decision point
Ensure: RELEASE or CANCEL
for $\tau \in \mathcal{T}$ **do**
  $u[\tau] \leftarrow \text{MAE}(\text{ded}_n[\tau], W)$
  $p[\tau] \leftarrow \text{MAE}(\text{opp}_{n-1}[\tau], W)$ {front-test to the decision point}
end for
$w_{\text{ded}} \leftarrow \arg\min_{\tau \in \mathcal{T}} u[\tau]$
$U \leftarrow u[w_{\text{ded}}]$ {incumbent error}
for $\tau \in \mathcal{T}$ **do**
  $\hat{g}[\tau] \leftarrow 100\,(U - p[\tau])\,/\,U$ {per-type predicted %improve}
end for
if $\hat{g}[\tau] > 0$ **for all** $\tau \in \mathcal{T}$ **then**
  return RELEASE
else
  return CANCEL
end if

the 102 total submissions, which generated an average improvement of +35.5% in terms of MAE—higher than the +33.0% average available to a rule with perfect foresight, because No-Regret selects only high-value releases and declines the marginal ones. 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.

No-Regret's guarantee is bought with rarity, and rarity costs more than staleness alone. Because it requires unanimity, it fires seldom, and each opportunistic publication it declines is a model refresh the edge does not receive—which, by the staleness argument of Section 4.3.3, has its own cost. The second heuristic relaxes the guarantee in exchange for publishing more often and for capturing more of the improvement that was available.

The High-Yield heuristic, given in Algorithm 2, differs from No-Regret in

Algorithm 2 The High-Yield heuristic. It reuses the incumbent quantities of Algorithm 1, but estimates the predecessor's per-type error from the window that iteration actually served rather than by front-testing it, and releases on unanimity or on a single sufficiently large predicted improvement. Both estimates are complete before the decision is made.

Require: $\mathcal{T}$ and the incumbent error U, from Algorithm 1
Require: $\text{opp}_{n-1}[\tau]$: type-τ model published by the last completed opportunistic iteration
Require: S_{n-1}: the window that iteration served, closed at the most recent dedicated publication
Require: $\theta = 50$: release threshold, in percentage points
Ensure: RELEASE or CANCEL
for $\tau \in \mathcal{T}$ **do**
  $p[\tau] \leftarrow \text{MAE}(\text{opp}_{n-1}[\tau], S_{n-1})$ {predecessor's realized error}
  $\hat{g}[\tau] \leftarrow 100\,(U - p[\tau])\,/\,U$
end for
$\hat{g} \leftarrow \max_{\tau \in \mathcal{T}} \hat{g}[\tau]$ {the predicted winner's %improve}
if $\hat{g}[\tau] > 0$ **for all** $\tau \in \mathcal{T}$ **or** $\hat{g} > \theta$ **then**
  return RELEASE
else
  return CANCEL
end if

two respects: it releases on unanimity *or* on any single model type predicting an improvement larger than a configuration-time threshold (set to 50 percentage points in this study), and it estimates the predecessor's per-type error from the window that iteration served rather than by front-testing it to the decision point. The rationale for the first is that a sufficiently large predicted gain is worth acting on even without consensus. The rationale for the second is empirical: the realized-outcome estimate is the weaker of the two classifiers, but paired with this release condition it captures more improvement than any other rule evaluated. Because the two heuristics read the predecessor differently, High-Yield's release set is not a superset of No-Regret's; the two overlap heavily, but each fires on decisions the other declines.

High-Yield releases 29 of the 102 submissions rather than No-Regret's 11, and captures more of the improvement that was available—+441.5% in total against No-Regret's +390.4%, or 26% of the oracle's ceiling against 23%. It also recovers 21 of the 51 opportunistic runs that genuinely helped, where No-Regret recovers 11. What it gives up is precision rather than value: at 72% precision, 8 of its releases were

ultimately degrading, so High-Yield does not carry No-Regret's guarantee, and its average improvement per released run is +15.2% against No-Regret's +35.5%. The two rules therefore bracket an operating range rather than ranking against one another. No-Regret maximizes per-run value and eliminates regret entirely; High-Yield maximizes the total improvement captured and the rate at which the edge is refreshed, which reduces staleness by the mechanism of Section 4.3.3. Table 4.6 summarizes both against the unconditional baseline.

Table 4.6: Performance of No-Regret and High-Yield heuristics over the 102 isolated opportunistic publications—each executed at the time the batch scheduler selected it, and so falling at an arbitrary point within the dedicated cluster's constant cadence—against an unconditional-execution baseline. Averages are per individual execution; for the unconditional executions all 102 HPC jobs are allowed to proceed. "Total %improve captured" sums the realized improvement over the jobs a heuristic releases, so it reads against the total available; "helpful runs captured" counts how many of the 51 submissions that would have improved the deployed model the heuristic actually released; "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. Neither heuristic dominates the other: the better figure in each row is set in bold.

	Uncond.	**No-Regret**	**High-Yield**
Released / 102	102	11	29
Helpful runs captured / 51	51	11	**21**
Avg %improve / released run	−20.1%	**+35.5%**	+15.2%
Total %improve captured	−2,048.6%	+390.4%	**+441.5%**
Degrading releases	51	**0**	8
Compute avoided (core-h)	—	**∼560,000**	∼449,000
Compute avoided (%)	—	**89%**	72%

Each canceled submission avoids the cost of an entire opportunistic HPC pipeline execution. We price that cost at the production configuration of the *sim* stage — 72 compute nodes of 32 cores each, or 2,304 cores (Section 4.2) — held for the median 2.67 hours an opportunistic iteration is measured to occupy the resource, giving roughly 6,150 core-hours per run. Launching all 102 opportunistic submissions unconditionally therefore requires nearly 628,000 core-hours. The replayed deployment itself requested a

narrower 8-machine allocation for these iterations: at the full 72-node width the queue wait was long enough that too few decisions would have been observed to score them meaningfully. The savings below are therefore stated at the production width the system is designed to run at, and scale down in proportion for the narrower request used to gather the sample.

The No-Regret heuristic launches only 11 of the 102 candidate jobs, canceling 91 and avoiding approximately *560,000 core-hours* of HPC computation—89% of the allocation unconditional release would consume—while never degrading the accuracy of the deployed model. The High-Yield heuristic launches 29 jobs and cancels 73, reducing HPC consumption by approximately *449,000 core-hours*, or 72%. Because the dedicated execution pipeline continues to publish throughout, every cancellation still leaves the edge on a model whose staleness is reset at the dedicated cadence, holding served accuracy within the band established in Section 4.3.3. The heuristics therefore need not capture every opportunistic run that would have helped; their purpose is to eliminate the runs that would have degraded the deployed model, and to commit allocation only where the evidence available at the decision point supports it.

Both heuristics hold up on the decisions they were not chosen on. The release conditions and the threshold θ were fixed on the earlier subset of this window reported in the associated publication, and have not been changed since. Scored on the iterations added afterward alone, No-Regret releases 9 jobs and every one of them improves the deployed model, preserving the perfect precision it showed on the earlier subset; High-Yield releases 25 at 72% precision, against 75% on the earlier subset. Neither rule's behavior was fitted to the decisions it was selected on, which is the main reason we report them as operating points rather than as the best of a swept family.

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.

4.4 Discussion

Taken together, the four stages establish an end-to-end account of the system. The dedicated-access pipeline publishes a new inference model every 134.8 minutes, which sets the baseline cadence and, through the decay curves of Section 4.3.2, the accuracy the edge can sustain without additional resources. Augmenting that pipeline with opportunistic execution on shared HPC reduces the interval between publications to 50.0 minutes during an active allocation—a 2.7× improvement—while keeping model error within the sensor measurement bound. Dissemination and inference at the edge then cost seconds against a five-minute observation interval. Finally, because an individual opportunistic run is not guaranteed to help, the decision heuristics gate each one at the head of the queue, eliminating degrading releases while avoiding the large majority of the allocation that unconditional release would consume, with agreement among independently trained surrogates as the signal they act on.

The edge measurements are the reason the others matter in the order presented: system performance is dominated by simulation and training latency, and 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 heuristic results rest on 102 scored decisions drawn from one deployment window, and No-Regret attains its guarantee on the 11 decisions it releases. They are evidence that the decision is worth making and that agreement among surrogates is the signal to make it on, 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.

4.5 Related Work

Early versions of the RBF concept, including the on-line/off-line learning and inference loop, were explored in prior workshop work [78], presented in Chapter 3. This chapter 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 [41], Parsl [21], Swift/T [127], FireWorks [72], Balsam [112], and RADICAL-Pilot [91] 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 [76, 114]. Related systems such as Ray [94], InferLine [39], Jellyfish [100], EC5 [131], Neurosurgeon [75], Edgent [81], JointDNN [48], and Loconte et al. [85] 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 [84,86,107], and has been applied extensively to computational fluid dynamics and related scientific domains [30, 89]. Recent work in scientific machine learning and HPC-integrated training [50,136] has significantly improved surrogate accuracy and expressiveness, while continual and streaming learning methods assume that models can be updated directly from incoming data [60, 138]. 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. [23] demonstrate asynchronous in-situ surrogate training coupled to CFD

simulations, while VESTEC [54] submits urgent HPC simulations in response to live observations. Brewer et al. [31] classify AI–HPC execution motifs, and ROSE [19] 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 [57, 105, 115, 116, 144].

In summary, these four strands settle respectively where a workflow executes, where inference is placed, how surrogates are learned, and how AI couples to HPC — but each takes the timing of model improvement as given. RBF takes that timing as the object of study: it treats an opportunistically scheduled HPC job as a unit whose value is uncertain, and decides at the head of the queue whether committing the allocation is warranted by the evidence available at that moment.

Chapter 5

Conclusion

Cyber-physical applications increasingly require low-latency spatial inference while continuously adapting to evolving environmental conditions. High-fidelity physics simulations and model retraining workflows are computationally expensive and typically execute on remote high-performance computing systems under batch scheduling, creating a fundamental mismatch between the responsiveness required at the edge and the cost and latency of simulation-driven model updates. A common response is to precompute models off-line and deploy them statically at the edge. However, static models degrade as conditions drift, while continuously recomputing high-fidelity simulations in response to every sensor update is infeasible due to latency, cost, and queue variability.

In this dissertation, we investigate whether a new distributed systems architecture — combining high-fidelity physics-based simulation with fast surrogate inference computed asynchronously — can deliver real-time, simulation-quality decision support at the edge. We chose an agricultural deployment that enabled us to investigate this question empirically in the field, and we validated each component using real-world measurement data. With careful design and after extensive empirical evaluation across a working deployment, our answer is a resounding yes.

We first develop and validate a computational fluid dynamics model for predicting spatial airflow inside a Citrus Under Protective Screens facility using OpenFOAM and a Darcy–Forchheimer porous-media model (Chapter 2). Validation at two independent scales — a controlled laboratory replica and a commercial 4-acre screenhouse in California's Central Valley — shows that modeled values predominantly fall within sensor measurement confidence intervals in both settings. This establishes the science driver and the concrete prediction problem that motivates all subsequent system work.

We then design and build xGFabric, a distributed system that connects field IoT sensors through a private 5G wireless network to campus and national HPC facilities (Chapter 3). The platform provides delay-tolerant data movement and workflow coordination across a continuum spanning battery-powered edge nodes, intermittent wireless links, and batch-scheduled supercomputers. We characterize the end-to-end network and pipeline performance, establishing that the platform meets the latency and reliability requirements for near-real-time edge-to-HPC coupling, and that the private 5G link carries the offered load with headroom to spare.

Building on this infrastructure, we present GrowFlow and RBF, a closed-loop modeling system that couples edge-based surrogate inference with continuous simulation-driven model updating — the on-line/off-line loop of Chapter 3, extended in Chapter 4. Surrogate models run at the edge, answering a query spanning the whole structure in 3 milliseconds to 1.1 seconds depending on model complexity, while HPC executes high-fidelity simulations and retrains updated models asynchronously. The system tolerates HPC batch-scheduling delays without interrupting real-time service. Combining the dedicated cluster with opportunistic access to NERSC Perlmutter achieves a 2.7× improvement in model publication cadence during an active allocation — reducing the average interval between updates from 134.8 to 50.0 minutes — while keeping inference

accuracy comparable to the sensor measurement error itself.

Finally, we characterize surrogate model accuracy decay as a function of staleness since the training cutoff and introduce decision-time heuristics for opportunistic HPC execution (Chapter 4). Each heuristic estimates the percentage improvement a queued job would deliver over the currently deployed model from observations already available when the job reaches the head of the batch queue, and gates release primarily on the *agreement* among three independently trained surrogates rather than on the confidence of any one of them. The No-Regret heuristic eliminates all degrading releases across the 102 decisions of the study while avoiding roughly 89% of the HPC allocation that unconditional release would consume—some 560,000 core-hours—and the complementary High-Yield heuristic captures more of the improvement that was available, 26% of what perfect foresight would collect, while more than doubling the number of opportunistic model updates delivered to the edge and still avoiding 72%, demonstrating that selective, prediction-guided opportunistic execution substantially outperforms unconditional release.

Together, these contributions demonstrate that heavy-duty scientific simulation — batch-scheduled, designed for throughput rather than latency, and subject to queue variability that cannot be controlled by the end user — can be integrated into operational real-time inference systems through careful system design. The architecture, methodology, and decision framework developed here are domain-agnostic: any cyber-physical system with a simulator capable of producing surrogate training data can adopt the on-line/off-line loop, apply the decay characterization methodology, and use heuristics derived from the same decision framework. This dissertation demonstrates that the tradeoff between edge responsiveness and simulation fidelity is not binary — and provides a deployed, empirically validated system as proof.

The software developed for this dissertation — the CFD case setup, the

surrogate training and transformation pipeline, the on-line/off-line loop, and the decision heuristics — will be released as open source as part of the xGFabric project once that codebase is finalized. The architecture described here can then be reproduced and extended in other deployments.

5.1 Future Directions

The system presented in this dissertation establishes that simulation-driven inference can be delivered at the edge in real time. It does not, however, close the loop between prediction and action, nor does it package the approach for use by anyone other than its authors. The directions below follow from the limitations identified in each chapter, ordered from the most immediate extensions of the deployed system to the broadest.

1. **Closing the Loop: Real-Time Intervention and Actuation** This dissertation delivers spatial wind-field predictions at the edge, but stops at the point where those predictions would be acted upon. The natural next step is to exploit the simulation results to perform real-time interventions in the CUPS facility. Accurate predictions of airflow are key for informing application practices of sprays, such as pesticide, and of water, for irrigation and frost mitigation, inside CUPS. Building an actuation tier on top of the on-line inference path raises questions this work does not address: what confidence threshold should gate an automated spray or frost-fan event, how the system should behave when the deployed surrogate is known to be stale, and how a grower should be able to override or audit an automated decision. Answering them requires a control layer with its own safety and accountability properties, not merely a better model.

2. **Extending the Model to Additional Target Parameters** One of the most important advantages of the model we chose is that it is relatively easy to add other target parameters. Adding another parameter, such as temperature, requires only temperature input and temperature interactions with the existing 3D model rigid bodies, since thermal spread equations can be part of the current solver. Heat transfer, which the current configuration does not model, will be important to applications such as frost prevention and irrigation scheduling. Validation of model accuracy for these additional target parameters will likely require additional sensing infrastructure. However, the results of this study indicate that good model accuracy based on the physics of fluid flow is possible for these parameters as well. Each new parameter also propagates through the rest of the system: the surrogate must learn it, the off-line loop must carry it, and the decay characterization of Chapter 4 must be repeated for it, since there is no reason to expect temperature and wind to become stale at the same rate.

3. **Reducing the Cost of the High-Fidelity Simulation** This dissertation treats the CFD simulation as a fixed, expensive black box and works around its latency rather than reducing it. Making the simulation itself cheaper would shorten every off-line iteration and widen the range of conditions the system can afford to cover. A distributed system with more powerful computing elements, including GPUs, is an appropriate platform, but OpenFOAM does not support that deployment scenario as part of its freely available, open-source distribution; commercially available CFD solvers do make these deployment options available, which raises an open question about the dollar cost, in hardware and software, of achieving real-time or near-real-time computing times. A second avenue follows from the observation that the main contribution to computation time comes from detail-oriented computational

meshing with relatively small cells. Improving the meshing balance—small cells in the areas of high importance and large cells in the less significant areas—would lower the computational cost without significant loss of accuracy.

4. **On-line and Incremental Surrogate Re-Training** The off-line loop currently retrains each surrogate from scratch on every iteration. Replacing periodic full re-training with incremental updates as new sensor data arrives would allow the system to adapt to gradual drift without full CFD re-runs, extending the practical lifetime of deployed surrogates and reducing HPC re-training overhead for long-lived installations. This matters most for changes that accumulate slowly. The research reported in Chapter 2 began before trees were introduced into the structure, so the data, model, and validation all reflect the pre-tree period of the study; validating the model further with young and mature trees remains future work, and a canopy that grows over successive seasons is precisely the kind of slow geometric drift that incremental re-training would need to track. Transfer learning across screenhouse geometries is a related opportunity, and would reduce the cold-start cost of standing up the loop at a new site.

5. **Tuning the Retraining Trigger** The system supports two policies for starting an off-line iteration — change-triggered and continuous (Section 3.1.7) — but the choice between them, and the settings that govern the first, are made by hand. Running continuously maximizes freshness and spends compute regardless of whether conditions have changed; triggering on detected change does the reverse. What remains open is how that detection should be performed and parameterized. Candidate mechanisms include statistical change-point detection on the sensor streams and residual-based drift monitoring that compares on-line predictions against ground-truth readings as they arrive, each of which would suppress unnecessary

retraining under stable conditions and prioritize updates when conditions shift most rapidly. Setting detection thresholds and evaluating their effect on staleness and retraining cost are open questions, and they interact directly with the benefit-cost heuristics of Chapter 4: a change detector is, in effect, a second predictor of whether an execution will be worth its cost.

6. **Precomputed Simulation Cache for Latency Reduction** A complementary way to shorten the off-line loop is to avoid running it at all. The system could maintain a cache of CFD results and trained surrogates for previously observed boundary-condition regimes, and on a retraining trigger match the new sensor conditions against cached regimes by nearest-neighbor lookup over wind speed and direction. On a hit, the cached model would be published immediately, bypassing the full off-line loop and the approximately 134.8 minutes it takes to complete one simulation-and-training iteration on the dedicated-access pipeline. This is particularly applicable at CUPS, where sensor quantization has already been observed to collapse a nominal 72-simulation iteration to 12 distinct ones (Section 3.2.4): the effective input space is finite, and repeated regimes are correspondingly likely over seasonal timescales. Quantifying the hit rate, defining the similarity threshold, and measuring the realized latency reduction remain open.

7. **Multi-Modal Sensor Fusion and Sparse Placement Optimization** Sensor density in real agricultural structures is always limited by cost and installation complexity. The weather stations used in this study were already part of another IoT research project in the CUPS, and we leveraged those sensors and that deployment infrastructure without introducing changes that might affect other projects or increase costs. Working within that constraint suggests treating placement as an optimization problem in its own right: jointly optimizing sensor placement and

surrogate accuracy under a fixed budget, and integrating heterogeneous sensors—anemometers, temperature, humidity, CO_2, and imaging—into a single inference path. The question to answer is a practical one for a grower: where should the next sensor go for maximum information gain?

8. **Digital Twin Toolkit for Controlled Environment Agriculture** The preceding directions all point toward the same unifying artifact: a composable, open-source toolkit that lets any CEA operator deploy a digital twin for their screenhouse without requiring CFD expertise. Such a toolkit would package a parameterized CFD scenario generator that takes a screenhouse geometry to a mesh and a simulation, a surrogate training pipeline that turns a simulation library into a deployed model, an edge deployment runtime with its xGFabric configuration, and a decay monitor with automated re-training triggers. Building it would also put the domain-agnostic claim of Chapter 4 to use rather than leaving it as an argument: the loop, the decay characterization, and the decision heuristics have so far been exercised against a single simulator in a single physical setting, and standing them up over a second geometry —or a second domain entirely—is what would turn generality from a property of the design into a demonstrated one. Realizing it in practice means lowering the barrier to entry for the people who operate these structures. The goal of this line of research is to expedite adoption of CUPS and to encourage the development of smart agriculture solutions for CUPS structures. A toolkit that an agronomist can operate without a fluid dynamicist is what that ultimately requires.

Bibliography

[1] "AcuRite Iris Personal Weather Stations - Personal Weather Stations - Shop for Weather - Shop All — acurite.com." https://www.acurite.com/shop-all/weather-instruments/weather-stations/iris.html?srsltid=AfmBOoqH_TRDRqAjbuMtQCQNoWvkgo7ppGCLVi24djVjcpl0-DUMOfSu. [Accessed 23-08-2024].

[2] "How to Predict Darcy and Forchheimer Coefficients for Perforated Plates Using an Analytical Approach — simscale.com." https://www.simscale.com/knowledge-base/predict-darcy-and-forchheimer-coefficients-for-perforated-plates-using-analytical-approach/. [Accessed 06-08-2026].

[3] "OpenFOAM — openfoam.com." https://www.openfoam.com/. [Accessed 26-08-2024].

[4] "Openfoam: API guide: applications/utilities/surface/surfaceFeatureExtract/surfaceFeatureExtract.C File Reference — openfoam.com." https://www.openfoam.com/documentation/guides/latest/api/surfaceFeatureExtract_8C.html. [Accessed 28-08-2024].

[5] "OpenFOAM Documentation - Reynolds Averaged Simulation (RAS) — doc.openfoam.com." https://doc.openfoam.com/2306/tools/processing/models/turbulence/ras/. [Accessed 26-08-2024].

[6] "OpenFOAM: Manual Pages: porousSimpleFoam(1) — openfoam.com." https://www.openfoam.com/documentation/guides/latest/man/porousSimpleFoam.html. [Accessed 28-08-2024].

[7] "ParaView - Open-source, multi-platform data analysis and visualization application — paraview.org." https://www.paraview.org/. [Accessed 28-08-2024].

[8] "SnappyHexMesh - OpenFOAMWiki — openfoamwiki.net." https://openfoamwiki.net/index.php/SnappyHexMesh. [Accessed 26-08-2024].

[9] "UT363/UT363BT Mini Anemometers - UNI-T Meters | Test & Measurement Tools and Solutions — meters.uni-trend.com."

https://meters.uni-trend.com/product/ut363-ut363bt/. [Accessed 23-08-2024].

[10] "Vantage Pro2 — davisinstruments.com." https://www.davisinstruments.com/collections/vantage-pro2?srsltid=AfmBOoqaY0U6K2F-X3xuWTlRZ-3yE6LfvYtnmDXRyaIeQimtSL6bPQqQ. [Accessed 23-08-2024].

[11] *Celona Orchestrator*, 2026.

[12] *O-RAN Alliance*, 2026.

[13] *Starlink: Reliable High-Speed Internet from Space.*, 2026.

[14] G. Adamides, N. Kalatzis, A. Stylianou, N. Marianos, F. Chatzipapadopoulos, M. Giannakopoulou, G. Papadavid, V. Vassiliou, and D. Neocleous, *Smart farming techniques for climate change adaptation in cyprus*, *Atmosphere* **11** (2020), no. 6 557.

[15] S. Agehara, G. Vallad, and E. Torres-Quezada, *Protected culture for vegetable and small fruit crops: Types of structures*, *Electronic Data Information Source (EDIS) HS1224* (12, 2012).

[16] U. Agriculture and N. Resources, *Distribution of ACP in California*, 2024. [Accessed 14-09-2024].

[17] B. Alazmi and K. Vafai, *Analysis of variants within the porous media transport models*, *J. Heat Transfer* **122** (2000), no. 2 303–326.

[18] O. F. Alrebi, B. Obeidat, I. A. Abdallah, E. F. Darwish, and A. Amhamed, *Airflow dynamics in an emergency department: A cfd simulation study to analyse covid-19 dispersion*, *Alexandria Engineering Journal* **61** (2022), no. 5 3435–3445.

[19] A. Alsaadi, T. Wang, A. Park, P. Bajracharya, L. Wang, F. Sun, S. Seal, V. Jadhao, G. Fox, and 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, (New York, NY, USA), p. 61–70, Association for Computing Machinery, 2025.

[20] I. Ascough, JC, J. Hanson, M. Shaffer, G. McMaster, and L. Deer-Ascough, *The great plains framework for agricultural resource management (gpfarm): a decision support system for whole farm/ranch strategic planning.*, *Sustaining the Global Farm (2001): Selected papers from the 1999 International Soil Conservation Organization Meeting* (1999).

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

[22] F. Bakir, C. Krintz, and R. Wolski, *CAPLets: Resource Aware, Capability-Based Access Control for IoT*, in *ACM/IEEE Symposium on Edge Computing*, 2021.

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

[24] D. L. Banks and S. E. Fienberg, *Data mining, statistics*, in *Encyclopedia of Physical Science and Technology (Third Edition)* (R. A. Meyers, ed.), pp. 247–261. Academic Press, New York, third edition ed., 2003.

[25] D. Batur, J. Ryan, and M. C. Vuran, *Dynamic resource sharing in private 5G networks with slicing*, in *INFORMS Annual Meeting*, (Phoenix, AZ), 2023.

[26] J. Bear, *Dynamics of fluids in porous media*. Courier Corporation, 2013.

[27] P. H. Beckman, *Sage grande: An open artificial intelligence testbed for edge computing and intelligent sensing*, *NSF Award Number 2436842. Directorate for Computer and Information Science and Engineering* **24** (2025), no. 2436842 36842.

[28] J. L. Bentley, *Multidimensional binary search trees used for associative searching*, *Communications of the ACM* **18** (1975), no. 9 509–517.

[29] S. Bhattacharyya, K. Dey, A. R. Paul, and R. Biswas, *A novel cfd analysis to minimize the spread of covid-19 virus in hospital isolation room*, *Chaos, Solitons & Fractals* **139** (2020) 110294.

[30] T. Botarelli, M. Fanfani, P. Nesi, and L. Pinelli, *Using physics-informed neural networks for solving navier-stokes equations in fluid dynamic complex scenarios*, *Engineering Applications of Artificial Intelligence* **148** (2025).

[31] W. Brewer, A. Gainaru, V. R. Suter, F. Wang, M. Emani, and S. Jha, *AI-coupled HPC workflow applications, middleware and performance*, 2024.

[32] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, *Physics-informed neural networks (pinns) for fluid mechanics: A review*, *Acta Mechanica Sinica* **37** (2021), no. 12 1727–1738.

[33] G. Cao, H. Awbi, R. Yao, Y. Fan, K. Sirén, R. Kosonen, and J. J. Zhang, *A review of the performance of different ventilation and airflow distribution systems in buildings*, *Building and environment* **73** (2014) 171–186.

[34] C. Catlett, P. Beckman, N. Ferrier, M. E. Papka, R. Sankaran, J. Solin, V. Taylor, D. Pancoast, and D. Reed, *Hands-on computer science: the array of things experimental urban instrument*, *Computing in Science & Engineering* **24** (2022), no. 1 57–63.

[35] G. Chen, Q. Xiong, P. J. Morris, E. G. Paterson, A. Sergeev, and Y. Wang, *Openfoam for computational fluid dynamics*, *Notices of the AMS* **61** (2014), no. 4 354–363.

[36] T. Chen, R. Li, X. Hu, B. Zhang, Y. Liu, L. L. Wang, and N. Gao, *Machine learning as cfd surrogate models for rapid prediction of building-related physical fields: A review of methods and state-of-the-art*, *Building and Environment* **285** (2025) 113667.

[37] I. Ciampitti, G. Mandrini, F. Gomez, P. Cisdeli, G. N. Santiago, P. Cano, J. Lacasa, F. Palmero, D. Cammarano, and P. Kyveryga, *Digital agriculture to transform decision support systems: State of the art and prospects*, *Advances in Agronomy* **195** (2026) 67–116.

[38] O. Coulaud, P. Morel, and J. Caltagirone, *Numerical modelling of nonlinear effects in laminar flow through a porous medium*, *Journal of Fluid Mechanics* **190** (1988) 393–407.

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

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

[41] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, B. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz, *Pegasus: A framework for mapping complex scientific workflows onto distributed systems*, *Scientific Programming Journal* **13** (2005), no. 3.

[42] A. Devis, N. P. Van Lipzig, and M. Demuzere, *Should future wind speed changes be taken into account in wind farm development?*, *Environmental Research Letters* **13** (2018), no. 6 064012.

[43] Docker. `https://www.docker.com/`.

[44] T. Ebert, A. Schumann, L. Waldo, D. Stanton, and P. Rolshausen, *A Close-up Look at Screens for Excluding Asian Citrus Psyllids*, 2021.

[45] T. Ekaireb, L. Brand, N. Avaraddy, M. Mock, C. Krintz, and R. Wolski, *Distributed dataflow across the edge-cloud continuum*, in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pp. 316–327, IEEE, 2024.

[46] C. El Hachimi, S. Belaqziz, S. Khabba, H. Karjoun, M. H. Kharrou, B. A. Hssaine, S. Er-Raki, A. Daccache, A. Chehbouni, and Q. Weng, *Towards collective intelligence in agriculture: Deep reinforcement learning and digital twins for efficient management of collective irrigation water distribution systems*, *Energy Nexus* **20** (2025) 100599.

[47] A. M. Endalew, M. Hertog, M. Delele, K. Baetens, T. Persoons, M. Baelmans, H. Ramon, B. Nicolaï, and P. Verboven, *Cfd modelling and wind tunnel validation of airflow through plant canopies using 3d canopy architecture*, *International Journal of Heat and Fluid Flow* **30** (2009), no. 2 356–368.

[48] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, *Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services*, *IEEE Transactions on Mobile Computing* **20** (2021), no. 2.

[49] Ettus. `https://www.ettus.com/`.

[50] *ExaLearn: US Department of Energy (DOE) Exascale Computing Project (ECP) center*, 2020.

[51] 2024. `https://farm-ng.com` [Online; accessed 5-Apr-2024].

[52] I. Farrance and R. Frenkel, *Uncertainty of measurement: a review of the rules for calculating uncertainty components through functional relationships*, *The Clinical Biochemist Reviews* **33** (2012), no. 2 49.

[53] R. S. Ferrarezi, J. A. Qureshi, A. L. Wright, M. A. Ritenour, and N. P. F. Macan, *Citrus production under screen as a strategy to protect grapefruit trees from huanglongbing disease*, *Frontiers in Plant Science* **10** (2019).

[54] 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).

[55] B. Foundation, "blender.org - Home of the Blender project - Free and Open 3D Creation Software — blender.org." `https://www.blender.org/`. [Accessed 28-08-2024].

[56] S. Frandsen, R. Barthelmie, S. Pryor, O. Rathmann, S. Larsen, J. Højstrup, and M. Thøgersen, *Analytical modelling of wind speed deficit in large offshore wind farms*, *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology* **9** (2006), no. 1-2 39–53.

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

[58] J. J. Glen, *Feature article—mathematical models in farm planning: A survey*, *Operations Research* **35** (Oct, 1987) 641–666.

[59] N. Golubovic, C. Krintz, R. Wolski, B. Sethuramasamyraja, and B. Liu, *A Scalable System for Executing and Scoring K-Means Clustering Techniques and Its Impact on Applications in Agriculture*, *International Journal of Big Data Intelligence* (July, 2018).

[60] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama, *Machine learning for streaming data: state of the art, challenges, and opportunities*, *ACM SIGKDD Explorations Newsletter* **21** (2019), no. 2 6–22.

[61] P. Gonzalez-De-Santos, R. Fernández, D. Sepúlveda, E. Navas, and M. Armada, *Unmanned ground vehicles for smart farms*, *Agron.-Clim. Chang. Food Secur* **6** (2020) 73.

[62] A. Hakimi, P. Ghafarian, and H. Farjami, *A novel approach to wind speed modeling: A fast and robust model with high generalizability*, *Engineering Applications of Artificial Intelligence* **157** (2025) 111316.

[63] D. G. Hall, M. L. Richardson, E.-D. Ammar, and S. E. Halbert, *Asian citrus psyllid, d iaphorina citri, vector of citrus huanglongbing disease*, *Entomologia Experimentalis et Applicata* **146** (2013), no. 2 207–223.

[64] H. Hematpur, S. M. Mahmood, N. H. Nasr, and K. A. Elraies, *Foam flow in porous media: Concepts, models and challenges*, *Journal of Natural Gas Science and Engineering* **53** (2018) 163–180.

[65] J. Herath, T. Yuba, and N. Saito, *Dataflow computing*, in *Parallel Algorithms and Architectures: International Workshop Suhl, GDR, May 25–30, 1987 Proceedings*, pp. 25–36, Springer, 1987.

[66] P. Hintjens, *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[67] A. Hodges and T. Spreen, *Economic impacts of citrus greening (hlb) in florida, 2006/07-2010/11*, *Electronic Data Information Source (EDIS) FE903* (01, 2012).

[68] J. Huang, T. Hao, Y. Wang, and P. Jones, *A street-scale simulation model for the cooling performance of urban greenery: Evidence from a high-density city*, *Sustainable Cities and Society* **82** (2022) 103908.

[69] N. T. T. Huyen, T. T. Thanh, and L. T. Ha, *Developing sustainable and resilient agriculture system of european countries in the rise of digital governance*, *Sustainable Futures* **10** (2025) 101292.

[70] I. E. Idelchik, *Handbook of Hydraulic Resistance*. Begell House, Redding, CT, USA, 4 ed., 2007.

[71] "Intel NUC." `http://www.intel.com/content/www/us/en/nuc/overview.html` [Online; accessed 14-Feb-2015].

[72] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, and K. A. Persson, *FireWorks: a dynamic workflow system designed for high-throughput applications*, *Concurrency and Computation: Practice and Experience* **27** (2015), no. 17 5037–5059.

[73] P. P. Jayaraman, A. Yavari, D. Georgakopoulos, A. Morshed, and A. Zaslavsky, *Internet of things platform for smart farming: Experiences and lessons learnt*, *Sensors* **16** (2016), no. 11 1884.

[74] I. T. Jolliffe, *A note on the use of principal components in regression*, *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **31** (1982), no. 3 300–303.

[75] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and 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).

[76] F. P. Knebel, R. Trevisan, G. S. do Nascimento, M. Abel, and 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.

[77] L. Kurafeeva, R. Hartung, B. C. Carter, A. Subedi, A. Biswas, M. Fay, S. Jha, C. Krintz, A. Merzky, D. Thain, M. C. Vuran, and R. Wolski, "Asynchronous Scientific Workflows for Simulation-Driven Inference Across the Edge–HPC Computing Continuum." Submitted to *Future Generation Computer Systems* (Elsevier), special issue, 2026.

[78] L. Kurafeeva, A. Subedi, R. Hartung, M. Fay, A. Biswas, S. Jha, O. Kilic, C. Krintz, A. Merzky, D. Thain, M. C. Vuran, and R. Wolski, *xGFabric: Coupling sensor networks and HPC facilities with private 5G wireless networks for real-time*

digital agriculture, in *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC Workshops '25, (New York, NY, USA), pp. 2317–2327, Association for Computing Machinery, 2025.

[79] L. Kurafeeva, R. Wolski, C. Krintz, and T. Smyth, *Airflow modeling for citrus under protective screens*, *Sensors* **24** (2024), no. 19 6200.

[80] S. Lawan, W. Abidin, W. Chai, A. Baharun, and T. Masri, *Different models of wind speed prediction; a comprehensive review*, *International Journal of Scientific & Engineering Research* **5** (2014), no. 1 1760–1768.

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

[82] G. Li and J. Shi, *On comparing three artificial neural networks for wind speed forecasting*, *Applied Energy* **87** (2010), no. 7 2313–2320.

[83] R. Li, Y. Zhao, L. L. Wang, J. Niu, X. Shi, and N. Gao, *Numerical investigation of the blockage effect of trees on airflow distributions in a wind tunnel*, *Building and Environment* **263** (2024) 111848.

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

[85] D. Loconte, F. Ieva, L. Pinto, G. Loseto, F. Scioscia, and M. Ruta, *Expanding the cloud-to-edge continuum to the IoT in serverless federated learning*, *Future Generation Computer Systems* **155** (2024).

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

[87] D. Lucor, A. Agrawal, and A. Sergent, *Physics-aware deep neural networks for surrogate modeling of turbulent natural convection*, *arXiv preprint arXiv:2103.03565* (2021).

[88] P. K. R. Maddikunta, S. Hakak, M. Alazab, S. Bhattacharya, T. R. Gadekallu, W. Z. Khan, and Q.-V. Pham, *Unmanned aerial vehicles in smart agriculture: Applications, requirements, and challenges*, *IEEE Sensors Journal* **21** (2021), no. 16 17608–17619.

[89] A. Marcato, G. Boccardo, and D. Marchisio, *A computational workflow to study particle transport and filtration in porous media: Coupling cfd and deep learning*, *Chemical Engineering Journal* **417** (2021).

[90] R. Martin-Clouaire, *Modelling Operational Decision-Making in Agriculture*, *Agricultural Sciences* **08** (2017), no. 07 527–544.

[91] A. Merzky, M. Turilli, M. Titov, A. Al-Saadi, and S. Jha, *Design and performance characterization of radical-pilot on leadership-class platforms*, *IEEE Transactions on Parallel and Distributed Systems* **33** (2022), no. 4.

[92] M. Mirzaie, E. Lakzian, A. Khan, M. E. Warkiani, O. Mahian, and G. Ahmadi, *Covid-19 spread in a classroom equipped with partition–a cfd approach*, *Journal of Hazardous Materials* **420** (2021) 126587.

[93] F. Mohamadi and A. Fazeli, *A review on applications of cfd modeling in covid-19 pandemic*, *Archives of Computational Methods in Engineering* **29** (2022), no. 6 3567–3586.

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

[95] D. Munch, *U.s. citrus production – an uphill battle to survive*, *Farm Bureau Market Intel* (2013).

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

[97] NERSC, *Perlmutter HPC Queue Wait Times*, 2026.

[98] D. A. Nield, A. Bejan, *et. al.*, *Convection in porous media*, vol. 3. Springer, 2006.

[99] P. V. Nielsen, *Computational fluid dynamics and room air movement*, *Indoor air* **14** (2004), no. Supplement 7 134–143.

[100] V. Nigade, P. Bauszat, H. Bal, and L. Wang, *Inference serving with end-to-end latency slos over dynamic edge networks*, *Real-Time Systems* **60** (2024).

[101] Open5GS. `https://open5gs.org/`.

[102] J. Park, A. Moon, E. Lee, and S. Kim, *Understanding iot climate data based predictive model for outdoor smart farm*, in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1892–1894, IEEE, 2021.

[103] K. Paul, S. S. Chatterjee, P. Pai, A. Varshney, S. Juikar, V. Prasad, B. Bhadra, and S. Dasgupta, *Viable smart sensors and their application in data driven agriculture*, *Computers and Electronics in Agriculture* **198** (2022) 107096.

[104] C. Pest and D. P. Program, *Huanglongbing Quarantine*, 2024. [Accessed 14-09-2024].

[105] M. Polese, M. Dohler, A. Dressler, M. Erol-Kantarci, R. Jana, R. Knopp, and T. Melodia, *Empowering the 6G cellular architecture with open RAN*, *IEEE Journal on Selected Areas in Communications* **42** (2024), no. 2 245–262.

[106] pysim. `https://github.com/osmocom/pysim`.

[107] M. Raissi, P. Perdikaris, and 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.

[108] T. A. Rajaperumal and C. Christopher Columbus, *Enhanced wind power forecasting using machine learning, deep learning models and ensemble integration*, *Sci. Rep.* **15** (jul, 2025) 20572.

[109] "Raspberry Pi." `https://www.raspberrypi.org`, 2025. [Online; accessed 15-Nov-2025].

[110] RM530N-GL. `https://www.waveshare.com/wiki/RM530N-GL`.

[111] *Sage Grande Testbed*, 2025. `https://sagecontinuum.org/docs/about/overview`[Online; accessed 3-Sep-2025].

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

[113] G. Santamaría-Bonfil, A. Reyes-Ballesteros, and C. Gershenson, *Wind speed forecasting for wind farms: A method based on support vector regression*, *Renewable Energy* **85** (2016) 790–809.

[114] S. Santillan and C. L. Abad, *An analysis of hpc and edge architectures in the cloud*, 2025.

[115] Z. Sarah, G. Nencioni, and M. A. Khan, *Resource allocation in multi-access edge computing for 5G-and-beyond networks*, *Computer Networks* **227** (2023).

[116] Z. Satka, M. Ashjaei, H. Fotouhi, M. Daneshtalab, M. Sjödin, and S. Mubeen, *A comprehensive systematic review of integration of time sensitive networking and 5G communication*, *Journal of Systems Architecture* **138** (2023).

[117] D. Schindler, J. Bauhus, and H. Mayer, *Wind effects on trees*, 2012.

[118] A. Schumann, L. Waldo, N. Mariner, and T. Ebert, *Five years of fresh fruit production in cups*, *Citrus Industry Magazine* **101** (2020) 14–17.

[119] A. W. Schumann, A. Singerman, M. Ritenour, J. Qureshi, and F. Alferez, *2024–2025 florida citrus production guide: Citrus under protective screen (cups) production systems*, 2025.

[120] D. Sellier, Y. Brunet, and T. Fourcaud, *A numerical model of tree aerodynamic response to a turbulent airflow*, *Forestry* **81** (2008), no. 3 279–297.

[121] U. N. A. S. Service, *Citrus Fruits 2023 Summary*, 2023. White Paper [Accessed 14-09-2024].

[122] A. Singerman and A. Schumann, *Return-on-Investment Potential of Citrus Under Protective Screens (CUPS)*, 2023. Citrus Industry White Paper [Accessed 14-09-2024].

[123] J. Slingo and T. Palmer, *Uncertainty in weather and climate prediction*, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **369** (2011), no. 1956 4751–4767.

[124] J. M. Snow, B. E. Stuckman, and J. S. Usher, *Accuracy requirements for measurement systems with uniform and normal errors*, *Naval Engineers Journal* **105** (1993), no. 1 66–69.

[125] E. Spertus and W. J. Dally, *Experiments with Dataflow on a General-Purpose Parallel Computer*. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1991.

[126] srsRAN Project. `https://www.srsran.com`.

[127] "Swift/T High Performance Dataflow Computing." `http://swift-lang.org/Swift-T/`, 2016. [Online; accessed 15-Nov-2016].

[128] sysmoISIM SJA5. `https://sysmocom.de/`.

[129] T. Talaviya, D. Shah, N. Patel, H. Yagnik, and M. Shah, *Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides*, *Artificial Intelligence in Agriculture* **4** (2020) 58–73.

[130] A. Tallet, C. Allery, and F. Allard, *Pod approach to determine in real-time the temperature distribution in a cavity*, *Building and Environment* **93** (2015) 34–49.

[131] J. Tan, F. Liu, B. Wang, Q. Wu, and C. P. Chen, *Ec5: Edge–cloud collaborative computing framework with compressive communication*, *Future Generation Computer Systems* **166** (2025).

[132] M. Turilli, M. Santcroos, and S. Jha, *A comprehensive perspective on pilot-job systems*, *ACM Computing Surveys (CSUR)* **51** (2018), no. 2 1–32.

[133] D. o. A. University of California and N. Resources, "Welcome — lrec.ucanr.edu." `https://lrec.ucanr.edu/`. [Accessed 24-08-2024].

[134] S. M. Valdivia-Bautista, J. A. Domínguez-Navarro, M. Pérez-Cisneros, C. J. Vega-Gómez, and B. Castillo-Téllez, *Artificial intelligence in wind speed forecasting: A review*, *Energies* **16** (2023), no. 5.

[135] T. van Druenen, T. Van Hooff, H. Montazeri, and B. Blocken, *Cfd evaluation of building geometry modifications to reduce pedestrian-level wind speed*, *Building and Environment* **163** (2019) 106293.

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

[137] *Waggle Software Stack*, 2025. `https://github.com/waggle-sensor`[Online; accessed 3-Sep-2025].

[138] L. Wang, X. Zhang, H. Su, and J. Zhu, *A comprehensive survey of continual learning: Theory, method and application*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **46** (2024), no. 8 5362–5383.

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

[140] R. Wolski, C. Krintz, and M. Mock, *Leveraging Dataflow as an Intermediate Representation for Portable Edge Deployments*, in *IEEE International Conference on Fog and Mobile Edge Computing*, pp. 1–8, 2025.

[141] R. Wolski, C. Krintz, and M. Mock, *Leveraging dataflow as an intermediate representation for portable edge deployments*, in *Proceedings of 10th International Conference on Fog and Mobile Edge Computing (FMEC 2025) – to appear*, 2025.

[142] "xGFabric: Coupling Sensor Networks and HPC Facilities with Advanced Wireless Networks for Near-Real-Time Simulation of Digital Agriculture." `https://sites.google.com/view/xgfabric/home` [Online; accessed 2-Sep-2025], 2025. `https://spark.apache.org/` [Online; accessed 14-Feb-2015].

[143] Q. Xiong, T. G. Baychev, and A. P. Jivkov, *Review of pore network modelling of porous media: Experimental characterisations, network constructions and applications to reactive transport*, *Journal of contaminant hydrology* **192** (2016) 101–117.

[144] F. Zhang, J. Wang, J. Xue, R. Wang, M. Nixon, and Y. Han, *Time-sensitive networking (TSN) for industrial automation: Current advances and future directions*, *ACM Computing Surveys* **57** (2024), no. 2.

[145] R. Zhao, S. Liu, J. Liu, N. Jiang, and Q. Chen, *A two-stage cfd-gnn approach for efficient steady-state prediction of urban airflow and airborne contaminant dispersion*, *Sustainable Cities and Society* **112** (2024) 105607.