Acies-OS: A Content-Centric Platform for Edge AI Twinning and Orchestration

Jinyang Li, Yizhuo Chen, Tomoyoshi Kimura,
Tianshi Wang, Ruijie Wang, Denizhan Kara, Yigong Hu
Department of Computer Science
University of Illinois Urbana-Champaign
Champaign, IL, USA

Li Wu, Walid A. Hanafy,
Abel Souza, Prashant Shenoy
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA, USA

Maggie Wigness,
Joydeep Bhattacharyya
DEVCOM Army Research Laboratory
Adelphi, MD, USA

Jae Kim, Guijun Wang, Greg Kimberly,
Josh Eckhardt, Denis Osipychev
The Boeing Company
Seattle, WA, USA

Tarek Abdelzaher
Department of Computer Science
University of Illinois Urbana-Champaign
Champaign, IL, USA
zaher@illinois.edu

2024 33rd International Conference on Computer Communications and Networks (ICCCN) | 979-8-3503-8461-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICCCN61486.2024.10637580

***Abstract*—This paper describes `Acies-OS`, a content-centric platform for edge AI twinning and orchestration that allows easy deployment, re-configuration, and control of edge AI services, augmented by a digital twin. The work is motivated by the proliferation of edge AI in a plethora of IoT applications, ranging from home automation to military defense, and the emergence of digital twins that go beyond monitoring and emulation into configuration management and optimization of edge capabilities. While past work focused on either the edge capabilities themselves or the digital twin, this work focuses on their seamless interactions, offering abstractions that enable the digital twin to manage and optimize an increasingly diverse edge AI system. Acies-OS features a structured namespace, a thin client library with flexible pub/sub-based communication, health monitoring support, and a control plane for twin-based value-added analysis and optimization. To illustrate the use of Acies-OS, we implemented a multi-node multi-modality vehicle classification application and used Acies-OS to interface it to a digital twin. We then deployed the system in the field to showcase run-time twin-based optimizations of inference latency, classification accuracy, and robustness to failures in noisy and challenging conditions.**

***Index Terms*—Digital Twin, Digital Twin Control Plane, Content-Centric Network, Internet of Things, Cyber Physical Systems, Edge AI.**

I. Introduction

Acies-OS is a novel content-centric platform for edge AI twinning and orchestration. It is motivated by two recent advances in IoT applications. The first is the *proliferation of intelligent IoT services*, or edge AI. The second is the *emergence of digital twins* [1] that go beyond emulation into run-time *orchestration* (perhaps based on running multiple emulations to arrive at a most favorable configuration of the deployed system or based on outputs of compute-intensive algorithms for twin-based functional optimization). These advances call for solutions to interface digital twins to the systems they emulate, optimize, and manage in a manner that facilitates not only state monitoring but also control.

Importantly, unlike many other application domains where the twinned system is fairly static (e.g., a vehicle that generally consists of the same types of components and interconnections), thereby allowing for a static custom interface with the twin, IoT deployments can differ dramatically from one to another. They may differ in the types of sensors used, the number of nodes deployed, the types of analytics executed, and the allocation of functions to computing resources, among other factors. Thus, general support is needed for describing the implemented system to the digital twin, as well as for describing the knobs exposed by the system for twin-based management and control. This support is the goal of AciesOS.

Acies-OS[1] assumes a microservices-based application architecture, where deployed analytics consist of multiple discrete stages executed across multiple hosts [2, 3]. The end-to-end functionality is modeled as a processing graph. This abstraction is particularly well-suited for Edge AI applications, where resource constraints call for careful distribution of functionality across multiple heterogeneous nodes, such that each node contains only one or a few steps of the pipeline. Unique to Acies-OS is the assumption of a built-in digital twin that helps configure, troubleshoot, and optimize edge performance [4]. As deployment conditions continually push for innovations that optimize operations and decision-making processes, digital twins have emerged as pivotal tools. The paper enables seamless integration of edge AI and digital twin-based orchestration to leverage their full joint potential.

Applying digital twin technology to edge AI applications presents several challenges, especially when combined with complex and dynamic deployment environments. We deployed a vehicle classification application and conducted real-world experiments to assess system performance. Evaluation results indicate that our digital twin framework helps identify an optimal accuracy-latency tradeoff [5], improve classification accuracy through model selection, and ensure uninterrupted application data flow during failures, demonstrating the flexibility, utility, and extensibility of the proposed design.

[1]The code for this project is available at https://github.com/acies-os/acies-os

II. System Design

A key challenge explored in Acies-OS is to develop a flexible interface between a digital twin and its twinned edge AI system that not only allows for state monitoring and twin-based emulation but also enables flexible control of the twinned system's workflow configuration to enable a wide range of system optimizations and adaptations to a dynamic, possibly unfriendly, environment.

Acies-OS features a middleware library that exports the abstraction of nodes, much like ROS [6], interconnected by a computation graph. The application is modeled as a set of services, where a service constitutes a node (in the computation graph) that processes incoming messages from topics, produces results, and publishes them to other topics. The services also subscribe to a special control (or configuration) topic that allows manipulation of service-specific parameters.

A control plane allows the twin to publish on configuration topics, thereby manipulating system configuration. All communication is content-centric, following a simple but flexible namespace that allows posting to nodes, services, and control channels. Below, we describe key elements of the namespace first, then elaborate the middleware library and control plane.

A. Namespace Design

Acies-OS features a namespace design geared to support a flexible and extensible control plane. The design aims to (i) support interoperability among heterogeneous components, (ii) enhance fault tolerance, and (iii) enable flexible twin-based control of system state. Towards that end, it adopts a content-centric approach [7], leveraging a pub/sub-based system. By combining a pub/sub-based middleware with carefully designed namespaces, different subsystems can be seamlessly integrated and interoperable through topics. Pub/sub-based communication patterns offer location-transparent communication, while the control plane provides robustness and fault tolerance. While many existing frameworks and middleware employ pub/sub-based communication [8, 9, 10], and some even offer namespace support [8, 9], the semantics and organization of the namespace are typically left to the discretion of the application developer. This variability from one application to another complicates system integration and impedes component reuse. In contrast, our well-designed simple (but extensible) namespace simplifies dataflow management and facilitates flexible and extensible development of control plane software. Next, we discuss the proposed namespace design, which consists of three main components.

Service Space: The application workload is modeled as a processing pipeline comprising multiple services running on one or multiple nodes. Each service is uniquely identified by a namespace in the format of row 1.1 as illustrated in Table I. Services may subscribe to multiple topics, process incoming messages, and publish resulting messages to output topics (row 1.2). Additionally, each service has a dedicated control topic to receive messages from the control plane (row 1.4) All services on a node fall under the node namespace (row 1). In cases where a node runs backup services for another node, the address of the backup service is the fully qualified address of the primary service placed under the `backup/` subspace (row 1.3).

Row ID	Address
1	`node_id/*`
1.1	`node_id/service_id/*`
1.2	`node_id/service_id/output`
1.3	`node_id1/backup/node_id2/service_id/*`
1.4	`node_id/service_id/ctrl`
2	`twin/*`
2.1	`twin/node_id/*`
2.2	`twin/node_id/service_id/*`
2.3	`twin/node_id/service_id/ctrl`
3	`cp/*`
3.1	`cp/heartbeat`
3.2	`cp/controller_id/*`
3.3	`cp/controller_id/ctrl`
3.4	`*/ctrl`

TABLE I: Namespace design

Twin Space: The address of a digital twin corresponds to the fully qualified address of its physical counterpart, placed under the twin namespace (row 2.2, Table I). Each twinned component, akin to its physical counterpart, possesses a dedicated control topic to receive messages from the control plane (row 2.3). Additionally, the node-service mapping is preserved within the twin space (row 2.1), facilitating the straightforward management and control of all services on a node. For instance, this allows for the simultaneous spawning of all twins of a node on similar hardware or a virtual machine. Consolidating all twinned components under the same namespace (row 2) streamlines the physical-digital twin address conversion process.

Control Space: All control topics are located within the control space (row 3, Table I). Each service transmits heartbeats and diagnostic messages to the heartbeat topic (row 3.1), which controllers may subscribe to for control functionalities such as monitoring, anomaly detection, and failover. Controllers, specialized services residing in the control space, have their own control topic to receive replies from the controlled services (row 3.3). Controllers can issue control messages to the control topics of selected targets or all services (row 3.4).

B. Twin middleware library

We implement a digital twin middleware library that adopts the proposed namespace design. The programming model and two example services implemented with the library are shown in Figure 1.

As mentioned earlier, a service is modeled as a node in a computation graph that processes incoming messages from topics, produces results, and publishes them to topics. The twin library handles control messages, including synchronization (Sync) [11], getting and setting service parameters, and maintaining heartbeats. The application handles the rest of the messages through message queues.

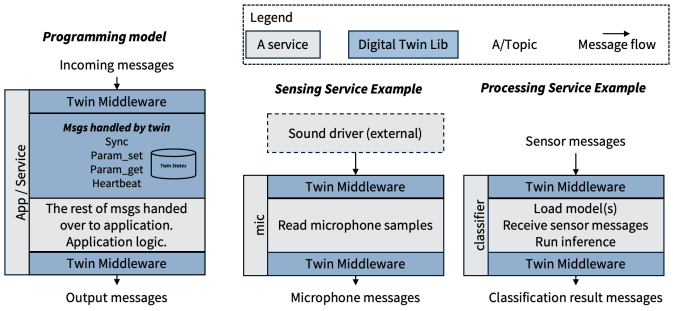


Fig. 1: Digital Twin middleware library and example services

We provide two examples in Figure 1. To implement a microphone service that records samples from a microphone and publishes the messages to the microphone topic, we only need to implement the application logic that reads samples from the sound driver and calls the `Service.send()` API with the target topic (.e.g, `node1/mic`) and a vector of samples as the payload. To implement a processing service that runs a neural network classifier on the incoming data and publishes the classification result to a topic, we need to provide the following application logics: (1) loading the neural network model on service start, (2) listening on input topics, (3) running the classifier when there are enough data, and (4) publish the classification to the downstream pipeline. The twin middleware handles message serialization and deserialization, distributes data to all subscribers, and responds to control messages.

C. Control Plane Architecture

The control plane monitors and oversees system states as shown in Figure 2a. The twin library sends out heartbeats to the `cp/heartbeat` topic, which the controller subscribes to. Each service has a dedicated control topic in its own namespace `<node>/<service>/ctrl` that receives control messages from the controller. Similarly, a controller can receive replies from the twins in its control topic `cp/controller/ctrl`.

Figure 2b shows the system architecture of a typical networked twin system. Each service in the system is communicating and managed by the control plane as illustrated in Figure 2a. On the physical twin side, there may be multiple services running on each node, forming a processing pipeline. Twins of selected components can be mirrored on physical nodes, virtual machines on edge servers, or in the cloud. Digital twins and their mirrored topics exist in the `twin/` namespace. In the example in Figure 2b, Node 1 and Node 2 run the same pipeline where Service 3 consumes messages from Service 1 and Service 2. The input topics to Service 3, `<node>/s1/topic` and `<node>/s2/topic` are mirrored to `twin/<node>/s1/topic` and `twin/<node>/s2/topic`. Then, the controller can perform analyses and optimizations on the digital twins of Service 3. If the controller finds a better service configuration or detects an anomaly, the services are reconfigured through its control topic `<node>/<service>/ctrl` by the controller.

The namespace design supports the development of various control plane analyses and optimizations tailored to the needs of the application. Next, we discuss two typical control and management functions that are commonly needed in edge AI applications and are built into the default controller. These functions are (i) inference optimization and (ii) failover (see Section III). They illustrate the utility, flexibility, and extendibility of the namespace design.

III. Control Function Examples

In IoT systems and Edge AI applications, the ability to monitor, respond to, and adapt to environmental dynamics is crucial for ensuring system robustness and reliability. However, implementing control plane software to manage both physical and digital twins, monitor their operational status, optimize AI model performance, and handle failures can be challenging, given the openness of the system and dynamic environment. In this section, we demonstrate how the proposed namespace design facilitates the implementation of such software through several examples.

A. Dynamic Model Selection

The first example of a twin-based value-added control function is a dynamic (edge AI) model selection to enhance inference accuracy and/or latency. Since edge nodes are generally resource-limited and heterogeneous, it is not always clear *a priori* which version of an AI model will offer the best latency/accuracy trade-off. Larger models may be too slow on a given edge device, whereas simpler models may be inaccurate. What is the best model to use given the current external environmental condition and internal resource availability/load? The idea is to exploit the substantial capabilities of a server that executes the digital twin to dynamically evaluate and select the most appropriate model, given latency and accuracy specifications.

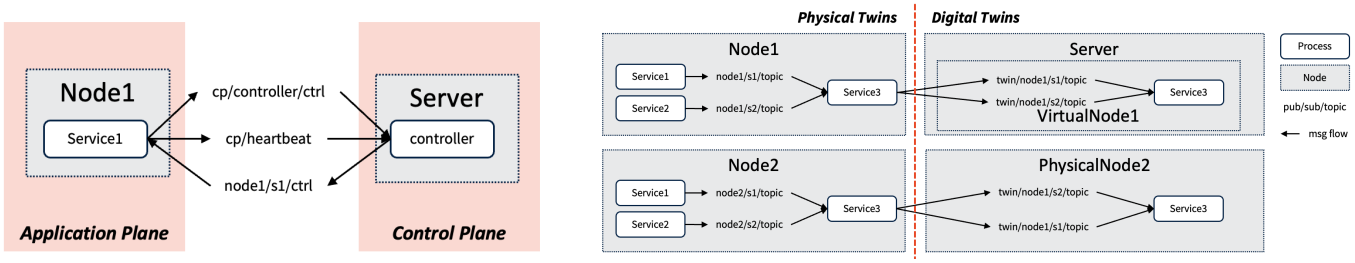
Let $\boldsymbol{x}_t$ denote the data received at time t and $\mathcal{X} = \{\boldsymbol{x}_t\}_{t=1}^{T}$ all the data received up to time T. The data synchronized to the twin space (row 2, Table I) $\mathcal{X}'$ is a subset: $\mathcal{X}' \subseteq \mathcal{X}$.

The controller can replay data point $\boldsymbol{x}_t \in \mathcal{X}'$ in the temporal order and observe the prediction of each model:

$$\hat{\boldsymbol{y}}_{t,i} = f(\boldsymbol{x}_t; \theta_i), \quad \boldsymbol{x}_t \in \mathcal{X}' \tag{1}$$

where $f(\cdot)$ is model inference, θ_i represents the i-th available model and $\hat{\boldsymbol{y}}_{t,i}$ the inference result of θ_i for $\boldsymbol{x}_t$. After a certain period of time, we may gain access to the ground truth label $\boldsymbol{y}_t$ for a certain subset of data points $\boldsymbol{x}_t \in \mathcal{X}'$, from stronger models on the cloud or human operators. To make a timely and accurate estimation of each model's trustworthiness, we evaluate each model's prediction at the latest time point with available ground truth. Formally, let $\boldsymbol{y}_s$ denote the latest available ground truth label before time point t, we calculate and compare the KL divergence between ground truth $\boldsymbol{y}_s$ and prediction $\hat{\boldsymbol{y}}_{s,i}$ made by each model, and select the model with least-KL-divergence-prediction as the *trusted* model of the system at time point t:

$$i_t^* = \arg\min_i \mathbb{KL}(\boldsymbol{y}_s || \hat{\boldsymbol{y}}_{s,i}) \tag{2}$$



(a) Control plane

(b) System architecture

Fig. 2: Control plane and system architecture

where i_t^* denotes the index of the *trusted* model at time point t. Then, the controller reconfigures the corresponding physical twin through the node's control topic (row 1.4, Table I) to use i_t^* so that the physical twin will run the best model.

B. Inference Latency Profiling and Reconfiguration

Inference latency is another critical aspect of many edge AI applications, directly influencing their effectiveness and responsiveness. To address this, our control plane includes a latency profiling functionality that measures and analyzes the time it takes for the neural network models to process data.

The controller replays data from $\mathcal{X}'$ to digital twins (row 2.2, Table I) configured with different parameters θ_i to record the resulting latency τ_i and inference performance A_i. This data establishes a relationship between τ_i and A_i, providing a basis for making informed trade-offs between accuracy and latency based on specific application needs.

The latency profiling and reconfiguration feature helps identify configurations that achieve decent accuracy with sufficiently fast response times, balancing performance with speed to maintain effective operation. We include an experiment in Section IV-C to showcase this feature.

Apart from edge inference optimizations, the control plane enables failure detection and can perform appropriate failover mechanisms to maintain the system's correctness.

C. Anomaly Detection

To continuously monitor the status of the twined system, the control plane collects heartbeats and synchronization messages (row 3.1, Table I). Each time a heartbeat or a state synchronization message is received, the *time to live* (TTL) value of the health status record for the corresponding physical twin is updated. System states, such as memory utilization, CPU utilization, CPU temperature, and core component voltages, are periodically piggybacked on the heartbeat messages. The system states, together with twined parameters registered by the application, are stored in a database on the controller, which is available for the anomaly detection algorithms.

The default controller implements a simple model-based anomaly detection algorithm [12, 13]. The controller interprets an *execution plan* supplied by the application, which details system models that delineate both healthy and failure states. The controller periodically evaluates the current system states by comparing them with the predefined models in the execution plan. If it detects any deviations that indicate a failure, the controller will try to initiate failover procedures, ensuring continuous system operation and minimizing downtime. The controller is designed with flexibility in mind and can be extended with more complex algorithms (e.g., knowledge-based and data-driven approaches). We broadly categorized commonly encountered failures into service failures and node failures, described below in detail.

D. Service Failures and Failover

We distinguish in our terminology between *service failures* and *node failures*. *Service failures* are defined as failures of certain components on a node that can be partially recovered by reconfiguring the system without resorting to backup nodes. This is as opposed to *node failures*, where some backup node(s) must be called upon. If a service failure is detected, the controller initiates a failover procedure as follows:

1) The controller first identifies the components in the processing pipeline that remain functional and healthy (row 3.1, Table I).
2) It then explores the configuration space, adjusting the parameters of the involved digital twins (row 2, Table I) to identify a configuration that achieves a system state recognized as healthy by the models defined in the execution plan.
3) Upon identifying a suitable configuration, the controller implements the necessary adjustments to the physical twins, including changing physical twin parameters, activating backups and heterogeneous replicas [3] (row 1.3, Table I), to facilitate recovery.

If no viable failover can be found, the controller will warn of the failure and report the incident. In Section IV-D, an experiment study with multiple failures is presented to demonstrate the effectiveness of the service failover procedure.

E. Node Failures and Failover

A node failure is defined as the failure of an entire node that can not be recovered solely within the existing system and must be addressed by activating a backup node. If a node failure is detected, the controller initiates a node failover as follows:

1) The controller will first locate any available backup nodes in the system.
2) Then, the controller will apply the last-known configuration of the digital twins from the failed node to

the digital twins of the backup nodes, testing whether this setup can achieve a system state deemed healthy according to the models defined in the execution plan.
3) If the backup nodes cannot accommodate a full replica of the failed node, the controller will seek a node capable of running a heterogeneous backup to restore a healthy system state [3, 14].
4) Upon identifying a suitable configuration, the controller implements the necessary adjustments and activates the backup node.

If no viable failover can be found, the controller will report the failure. The effectiveness of node failure failover is demonstrated in Section IV-D.

These control plane functionalities apply to other applications that adhere to the same namespace structure. Additionally, other types of controllers can be developed as plug-and-play solutions for applications following the same convention.

IV. Evaluation

The above has been a description of how certain useful services can be implemented where changes in load, environmental conditions, or health status trigger the twin to consider the space of possible reconfigurations, ultimately producing a new system configuration that is optimized for the new condition. Below, we evaluate the efficacy of these mechanisms.

For the sake of the experiments below, we implemented a multi-node, multi-modality vehicle classification application as a case study to evaluate the effectiveness of our design from various perspectives.

A. Experiment Setup

Extending the experimental setup in [15, 16, 17, 18], we build our vehicle classification system with multiple Raspberry Pi nodes, each of which is equipped with a microphone, a geophone and a GPS sensor to record acoustic signals, seismic signals and GPS locations simultaneously. Raspberry Pis are connected to portable batteries as power supply. We place the nodes on the sides of an experimental field. The nodes are connected with each other and to an edge server using a standard 802.11 router. Our experiments include four commonly available civilian vehicles with different weights, powers, and torques. The detailed specifications of the vehicles are described in Table II.

Model	Type	Weight (lbs)	Engine (all 4cyl)	Power (hp)	Torque (lb-ft)
2018 Mustang	Sports Car	3858	2.3L T	310	350
2022 MX-5	Roadster	2745	2.0L NA	181	151
2023 CX-30	Compact SUV	4345	2.5L NA	186	186
2017 GLE-350	Mid-size SUV	6217	2.0L T	255	273

TABLE II: The specifications of the four target vehicles used in our experiment. The vehicles span a variety of types, weights, powers, and torques to ensure

We choose neural networks [16, 17, 18] as vehicle classifiers in our experiments for their state-of-the-art performance. They take in sensor signals as input and output a categorical probability distribution as the prediction.

We conducted comprehensive experiments with the same vehicle classification system and procedures in two different locations (denoted as A and B, respectively). The spatial layout of our system in location A and an example vehicle trace are shown in Figure 3a and Figure 3b, respectively. The system layout and vehicle trace in location B are determined similarly.

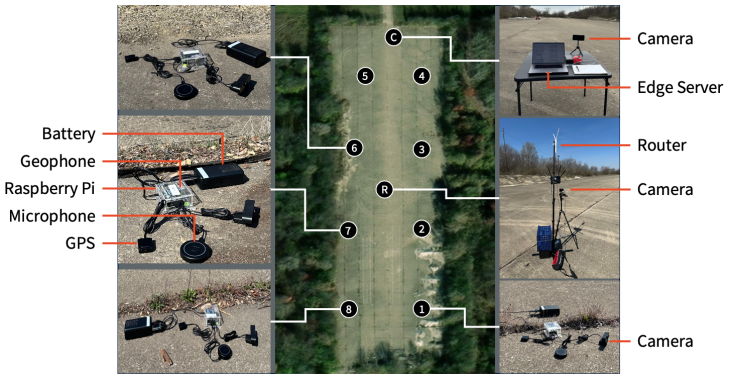
B. Evaluation of Dynamic Model Selection

We first experiment with the model selection algorithm using our vehicle classification system on location B. In this experiment, we adopt one acoustic signal based model and one seismic signal based model to make predictions. The ground truth label for each data point is revealed to the twin after the prediction is made for this data point. In this experiment, we reveal ground truth to the twin manually. In a real deployment, ground truth might come from a calibration service (e.g., reliable camera-based classifier) that is available (for calibration purposes) to the twin but not otherwise available for the deployed system. As described in Section III-A, the digital twin dynamically selects which one of the models to trust based on the latest available ground truth label and the models' corresponding predictions. The calibration service is deployed for nearly 600 seconds in total. To further examine the effectiveness of our model selection algorithm under severe environmental conditions, we introduced additional simulated wind noise from 150 seconds to 370 seconds.

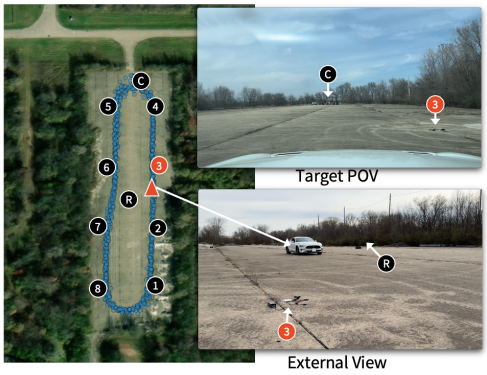
The experiment results are shown in Figure 4, we can see that the dynamically selected *trusted* model achieves steadily high accuracy and outperforms both of the individual models most of the time. In contrast, the accuracy of the seismic model is unstable throughout the experiment, while the accuracy of the acoustic model is steadily high in the quiet environment but drastically decreases in the presence of wind noise. The results show that our model selection algorithm can stably identify the more trustworthy model dynamically, even in severe environmental conditions, and consequently improve the overall prediction accuracy of the system.

C. Evaluation on Inference Latency Profiling

In the following experiment, we explore the latency-accuracy trade-off across various models within our vehicle classification system, employing the identical experimental setup, location, and the two neural network models outlined in Section IV-B. Throughout the experiment, simulated wind noise was introduced to assess our system under severe environmental conditions. For comprehensiveness, we profiled the two neural network models and the selected *trusted* model described in Section IV-B. All models originally operated on a latency of 1000 ms. Models with latencies larger than 1000 ms are created by performing temporally ensemble on original model predictions of recent seconds.



(a) Experimental setup and spatial layout of our vehicle classification system. 8 Raspberry Pi nodes with sensors and batteries are deployed uniformly on the sides of the field and are connected by an 802.11 router.



(b) Exemplar vehicle trace. The vehicle is passing by Node 3 (marked as red), which can be seen from both target point of view and external view.

Fig. 3: Experimental setup of the vehicle classification system and an exemplar vehicle trace.

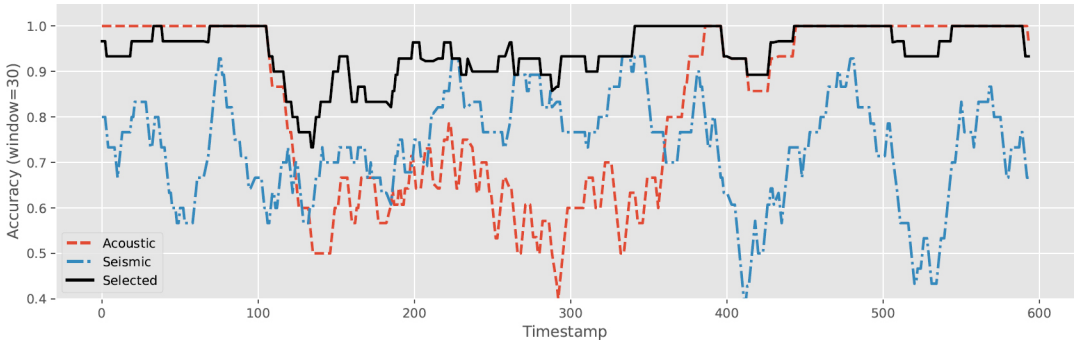


Fig. 4: Comparison of the classification accuracy of different methods. Classification accuracy at each time point is calculated over the past 30 seconds. We introduced simulated wind noise from 150 seconds to 370 seconds.

The results, displayed in Figure 5, reveal that models with higher latency tend to achieve higher accuracy for all three model families, which is an expected outcome given the broader temporal information base for the models with larger latency. Notably, the selected model consistently outperforms the individual models at all latency levels, validating the efficacy of our model selection algorithm. However, the accuracy of the selected model plateaus at higher latencies, underscoring the possibility and importance of selecting a model with an optimal latency level. For instance, one may choose the selected model with a latency of around 2000 ms for a real-time vehicle classification application to deliver satisfactorily accurate predictions at the cost of a tolerable time delay.

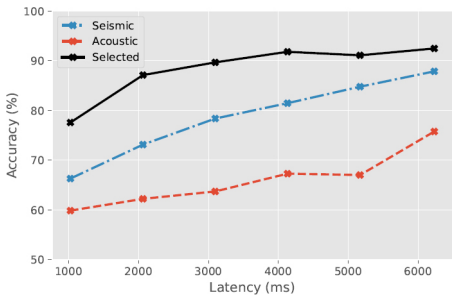


Fig. 5: The trade-off between classification accuracy and inference latency for different methods.

D. Evaluation on Failure Detection and Failover

In the following experiments, we implemented our failure detection and failover mechanisms described in Section III-C within our vehicle classification system in location A. These experiments involved three neural network models serving as classifiers: 1) *classifier-geo*, which requires only seismic signals; 2) *classifier-mic*, which requires only acoustic signals; and 3) *classifier-both*, which utilizes both acoustic and seismic signals and yields higher accuracy than the former two models. In healthy states, each node operates one of these classifiers based on the available sensor data. *classifier-both* should be selected if both sensors are operational; otherwise, either *classifier-geo* or *classifier-mic* should be selected depending on which sensor remains operational.

In this particular application, we abstract the most common issues encountered during vehicle classification system operations into two categories: service failure and node failure. Service failure is defined as any discrepancy between available sensor signals and the running classifier, which may be caused by a change in sensor availability or a failure of the classifier process. An example of service failure is shown in Figure 6.

Based on Section III, we specifically designed two failover mechanisms for service failure: 1) for service failure caused by the change of sensor availability, we replace the current classifier on the corresponding node with the appropriate classifier, as illustrated in Figure 7; 2) for service failure caused by the failure of the classifier process, we identify a node with extra computational power to run a backup classifier.

The node failure in our application is defined as the failure of the entire Raspberry Pi or all equipped sensors, which may be caused by a power outage. The corresponding failover mechanism is to activate a standby node to replace the failed one. The failure and failover process is depicted in detail in Figure 8.

We conducted two experiments to evaluate the empirical performance of our failover system. In the first experiment, we inject four service failures on different nodes at different time points, as detailed in Table III. The result is shown in Figure 9 (left). We can observe that, after the first injected failure, the number of correct classifications of the system with failover is consistently higher than the system without failover. Besides, the number of correct classifications of the system with failover grows linearly throughout the experiment, showing that our failure detection and failover mechanism successfully addressed all injected failures and consistently brought the system back to a healthy state.

In the other experiment, we injected 2 node failures on different nodes, which are also listed in Table IV. We can observe from the result shown in Figure 9 (right) that the system with failover also achieves a consistently higher number of correct classifications than the system without failover. Similarly, the number of correct classifications for the system with failover also grows linearly a period after the injection of the failures, which also indicates the effectiveness of our failure detection and failover mechanism.

When	Service Failure Causes	Failover
342s	node7/mic disabled	node7/classifier-both → node7/classifier-geo
458s	node3/geo disabled	node3/classifier-both → node3/classifier-mic
462s	node6/classifier-both failed	node6/classifier-both → node4/backup/node6/classifier-both
522s	node5/mic disabled	node5/classifier-both → node5/classifier-geo

TABLE III: Service failures injected during the experiment. The experiment lasts for 1800 seconds.

When	Failed Node	Failover
152s	node5/*	node5/* → node6/*
212s	node7/*	node7/* → node4/*

TABLE IV: Node failures injected during the experiment. The experiment lasts for 1800 seconds.

V. Related Work

The concept of the digital twin originated from NASA's early simulations of spacecraft systems, where mirroring physical systems in a digital framework allowed for effective management of complex operations [1]. Digital twin technology

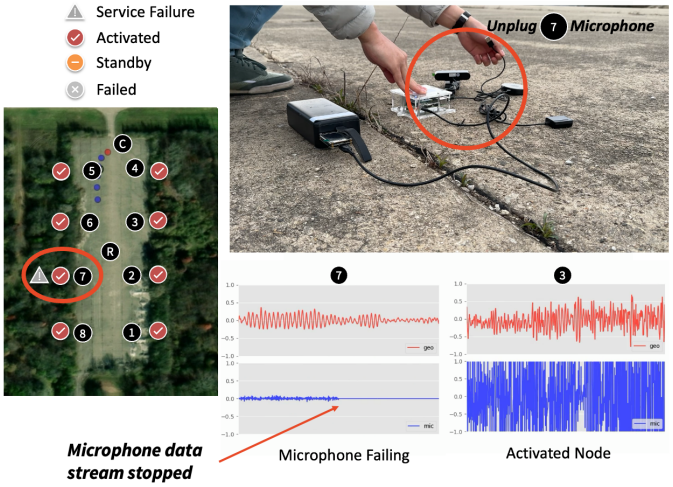


Fig. 6: Illustration of an example of service failure in our vehicle classification system. The microphone attached to node 7 is unplugged, making the acoustic signals inaccessible. The currently running *classifier-both* can no longer process the data, arriving at a service failure.

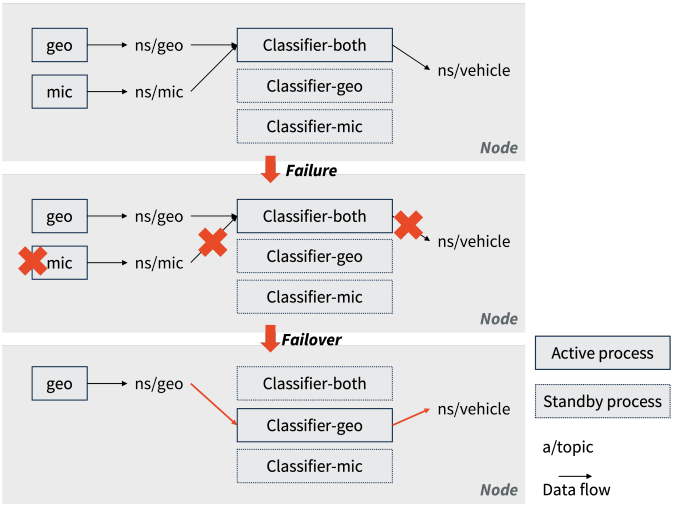


Fig. 7: Illustration of a service failure and corresponding failover mechanism. The microphone is disabled, rendering the current classifier *classifier-both* unusable and consequently causing a service failure. The service failure is then addressed by switching the *classifier-both* to *classifier-geo*.

bridges the physical and digital worlds by collecting real-time data from embedded sensors and other sources, which are then used to simulate the physical counterpart in a virtual environment. Examples include OpenTwins [8], a versatile framework that supports real-time data streams and machine learning predictions via its Kafka-ML integration; Mobility Digital Twin (MDT) [19] for transportation and mobility services; uDiT [10], a digital twin architecture designed to enhance the dependability of Cyber-Physical Systems (CPS) through distributed cooperation and data-centric communication middleware; a digital twin middleware [20] based on the YANG data model, designed to enhance communication

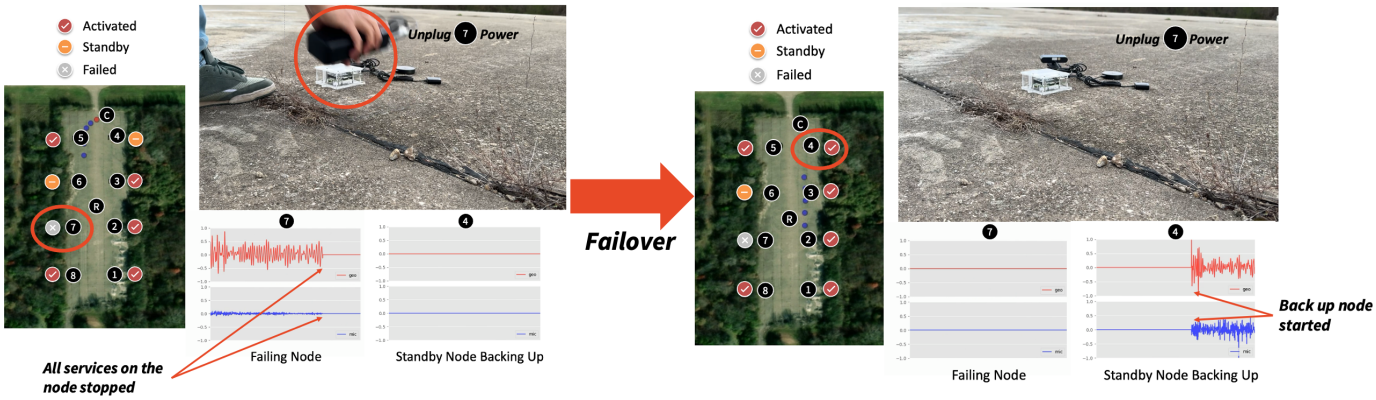


Fig. 8: Illustration of a node failure and its corresponding failover process. The battery is unplugged from node 7, which deactivates the entire node, causing a node failure. Standby node four is then activated to replace node 7.

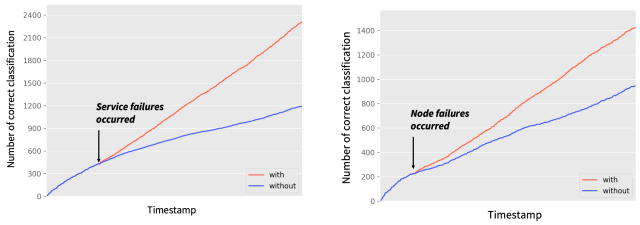


Fig. 9: Number of correct classifications over time in the presence of service failures (left) and node failure (right) with and without digital twin controller.

efficiency between IoT devices and their digital twins in smart farming; Eclipse Ditto [9] an open-source framework that facilitates the digital representation of physical objects on the Internet of Things (IoT), and others. These frameworks primarily emphasize efficient communication and modeling, with some incorporating data-centric pub/sub systems [8, 10] or hierarchical namespace support [8, 9]. In contrast, we focus on the integration of twin control and management functions with a microservices-based system abstraction, allowing flexible twin-based orchestration of system behavior.

Existing work explores the use of digital twins for optimizing deep neural network workloads and for anomaly detection, but not within a general and integrated control plane setting. For example, prior work [21] investigates the use of digital twins to offload deep neural network (DNN) inference workload to servers in the Industrial Internet of Things (IIoT). Another framework [13] leverages the Digital Twin concept to enable real-time health monitoring and anomaly prediction by merging historical and real-time data processing at the network edge. Some efforts [22] propose a flexible framework that leverages both digital twins and data-driven techniques to detect and classify anomalous behaviors, which can occur due to modeling errors or faults in the physical system. In contrast to these specialized functions, our approach centers on a general design for system control and optimization. The optimization examples in the paper entail model selection and latency optimization rather than offloading. Additionally, the paper illustrates a failover example that showcases the flexibility and extensibility of the proposed control plane design.

VI. Conclusion

In conclusion, this paper introduces a novel content-centric control plane design tailored for digital twin systems in edge AI applications. By meticulously crafting the namespace, our approach tackles key challenges in control plane development, including interoperability, robustness, and coordination. The proposed digital twin system boasts a structured namespace alongside a lightweight client library equipped with versatile pub/sub-based communication middleware. Our control plane implementation excels in monitoring system states and executing a variety of value-added analyses and optimizations.

To demonstrate the practicality and effectiveness of our proposed system, we implemented a multi-node multi-modality vehicle classification application. Through field deployment, we showcase how our digital twin system enhances inference latency, classification accuracy, and robustness in the face of environmental dynamics and system failures, particularly in noisy and challenging conditions. This study underscores the potential of content-centric control plane designs in advancing edge AI applications and highlights the tangible benefits they offer in real-world scenarios.

Acknowledgment

Research reported in this paper was sponsored in part by DEVCOM ARL under Cooperative Agreement W911NF- 17-2-0196, NSF CNS 20-38817, and the Boeing Company. It was also supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, DARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray, and D. Devine, "Digital Twin: Origin to Future," *Applied System Innovation*, vol. 4, no. 2, p. 36, Jun. 2021.

[2] A. Souza, N. Ng, T. Abdelzaher, D. Towsley, and P. Shenoy, "Unlocking Efficiency: Understanding End-to-End Performance in Distributed Analytics Pipelines," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*. Boston, MA, USA: IEEE, Oct. 2023, pp. 150–155.

[3] W. A. Hanafy, L. Wu, T. Abdelzaher, S. Diggavi, and P. Shenoy, "Failure-Resilient ML Inference at the Edge through Graceful Service Degradation," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, Oct. 2023, pp. 144–149.

[4] A. Rasheed, O. San, and T. Kvamsdal, "Digital Twin: Values, Challenges and Enablers From a Modeling Perspective," *IEEE Access*, vol. 8, pp. 21 980–22 012, 2020.

[5] W. A. Hanafy, T. Molom-Ochir, and R. Shenoy, "Design considerations for energy-efficient inference on edge devices," in *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, 2021, pp. 302–308.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[7] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[8] J. Robles, C. Martín, and M. Díaz, "OpenTwins: An open-source framework for the design, development and integration of effective 3D-IoT-AI-powered digital twins," Jan. 2023.

[9] "Eclipse Ditto™ • open source framework for digital twins in the IoT." [Online]. Available: https://eclipse.dev/ditto/index.html

[10] S. Yun, J.-H. Park, and W.-T. Kim, "Data-centric middleware based digital twin platform for dependable cyber-physical systems," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2017, pp. 922–926.

[11] D. Kalasapura, J. Li, S. Liu, Y. Chen, R. Wang, T. Abdelzaher, M. Caesar, J. Bhattacharyya, J. Kim, G. Wang *et al.*, "Twinsync: A digital twin synchronization protocol for bandwidth-limited iot applications," in *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2023, pp. 1–1.

[12] E. Khalastchi, M. Kalech, G. A. Kaminka, and R. Lin, "Online data-driven anomaly detection in autonomous robots," *Knowledge and Information Systems*, vol. 43, no. 3, pp. 657–688, Jun. 2015.

[13] H. Huang, L. Yang, Y. Wang, X. Xu, and Y. Lu, "Digital Twin-driven online anomaly detection for an automation system based on edge intelligence," *Journal of Manufacturing Systems*, vol. 59, pp. 138–150, Apr. 2021.

[14] J. Li, R. Ma, V. S. Mailthody, C. Samplawski, B. Marlin, S. Chen, S. Yao, and T. Abdelzaher, "Towards an Accurate Latency Model for Convolutional Neural Network Layers on GPUs," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, Nov. 2021, pp. 904–909.

[15] D. Liu, T. Abdelzaher, T. Wang, Y. Hu, J. Li, S. Liu, M. Caesar, D. Kalasapura, J. Bhattacharyya, N. Srour, M. Wigness, J. Kim, G. Wang, G. Kimberly, D. Osipychev, and S. Yao, "IoBT-OS: Optimizing the Sensing-to-Decision Loop for the Internet of Battlefield Things," in *The 31st International Conference on Computer Communications and Networks (ICCCN 2022)*, May 2022, p. 10.

[16] S. Liu, T. Kimura, D. Liu, R. Wang, J. Li, S. Diggavi, M. Srivastava, and T. Abdelzaher, "FOCAL: Contrastive Learning for Multimodal Time-Series Sensing Signals in Factorized Orthogonal Latent Space," *Advances in Neural Information Processing Systems*, vol. 36, pp. 47 309–47 338, Dec. 2023.

[17] T. Wang, J. Li, R. Wang, D. Kara, S. Liu, D. Wertheimer, A. Viros-i-Martin, R. Ganti, M. Srivatsa, and T. Abdelzaher, "SudokuSens: Enhancing Deep Learning Robustness for IoT Sensing Applications using a Generative Approach," Feb. 2024.

[18] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 351–360.

[19] Z. Wang, R. Gupta, K. Han, H. Wang, A. Ganlath, N. Ammar, and P. Tiwari, "Mobility Digital Twin: Concept, Architecture, Case Study, and Future Challenges," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 452–17 467, Sep. 2022.

[20] L. V. Cakir, T. Bilen, M. Özdem, and B. Canberk, "Digital Twin Middleware for Smart Farm IoT Networks," in *2023 International Balkan Conference on Communications and Networking (BalkanCom)*. İstanbul, Turkiye: IEEE, Jun. 2023, pp. 1–5.

[21] S. Hu, M. Li, J. Gao, C. Zhou, and X. S. Shen, "Digital Twin-Assisted Adaptive DNN Inference in Industrial Internet of Things," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, Dec. 2022, pp. 1025–1030.

[22] C. Gao, H. Park, and A. Easwaran, "An anomaly detection framework for digital twin driven cyber-physical systems," in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, ser. ICCPS '21. New York, NY, USA: Association for Computing Machinery, May 2021, pp. 44–54.