International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Self-Hosted Cloud-Native Digital Twin Architecture for Multi-Station Discrete Manufacturing

Agam Ramteke, Pranay Tarjule, Pritam Kumar Rai, Jafar Ansari
B.Tech, Industrial IoT — Priyadarshini College of Engineering, Nagpur, India

Abstract—Digital twin platforms for discrete manufacturing are dominated by managed cloud offerings (Azure Digital Twins, AWS IoT TwinMaker, Siemens Xcelerator, PTC ThingWorx) and by a growing but fragmented set of open-source, Kubernetes-native frameworks (Eclipse Ditto, KTWIN, OpenTwins, IndustryFusion, OFacT). Two open questions motivate this work. First, whether a **self-hosted**, Kubernetes-orchestrated architecture can support multiple synchronized machine twins for a discrete production line — modeled with causal, empirically informed simulators rather than randomized sensor generators — while remaining operable on commodity infrastructure realistic for an SME. Second, whether representing **process-level** dependencies between independently deployed machine twins yields decision-support value beyond isolated per-machine monitoring. This paper specifies a five-station automotive-component line (Stamping → CNC → Weld/Assembly → Quality Inspection → Packaging/Dispatch), a Live-Twin/Scenario-Worker architectural split with a DAG-based process twin under frozen-snapshot semantics, reduced-order causal machine models calibrated against publicly available industrial datasets (PHM Society 2010 Milling, NASA IMS Bearing, AI4I 2020, Bosch Production Line Performance), and a full experimental protocol for synchronization, scalability, resilience, and model-fidelity evaluation. The related-work analysis situates the contribution against both managed platforms and the Kubernetes-digital-twin literature, with particular care to distinguish digital twins that run **on** Kubernetes (this work; KTWIN) from digital twins **of** Kubernetes itself (KAPETÁNIOS, KubeTwin) — a distinction the existing literature does not always make explicit. Quantitative claims about synchronization latency, scalability limits, and simulator fidelity are deferred to the experimental sections and marked pending until measured on the deployed system.

Index Terms—digital twin, Kubernetes, cloud-native architecture, MQTT, Industrial IoT, causal simulation, predictive maintenance, SME digitalization, discrete manufacturing

I. INTRODUCTION

A. Manufacturing Context

The target process is discrete, sequential automotive-component manufacturing: raw sheet stock proceeds through stamping, CNC machining, weld/assembly, quality inspection, and packaging/dispatch. This is a bounded, well-understood manufacturing topology (five stations, one dominant material-flow direction, no rework loops modeled in the base case), chosen specifically because it is small enough to model completely and causally rather than statistically.

B. Digital Twin Motivation

The manufacturing digital twin literature is mature at the level of individual machines (tool-wear monitoring, spindle/bearing prognostics) and mature at the level of managed enterprise platforms (whole-factory 3D twins with commercial support). It is comparatively thin at the specific intersection this work occupies: a small number of causally modeled, independently deployable machine twins, orchestrated as a cloud-native system, self-hosted on infrastructure a small or medium-sized enterprise (SME) could plausibly own outright.

C. SME Context

Section XIII develops this trade-off in full, but the framing claim is narrow and defensible: self-hosting removes mandatory dependence on a managed digital-twin service and its recurring usage-based fees, while shifting infrastructure, security, and maintenance responsibility onto the deploying organization. It is a trade-off, not a strict improvement; Siemens and Dassault both now market cloud-hosted, subscription-based "democratized" digital-twin offerings explicitly aimed at SMBs, which addresses the upfront-cost barrier without addressing the vendor-dependency and data-locality barrier.

D. Problem Statement

Can a small number of independently deployable, Kubernetes-orchestrated machine digital twins, built from reduced-order causal models calibrated against public industrial data rather than random signal generators, be synchronized through MQTT and composed into a process-level twin, at a fidelity and cost profile realistic for a resource-constrained deployment?

E. Research Gap

The gap is not that nobody has used Kubernetes for digital twins (KTWIN, IndustryFusion, and DT-Forge already do), and not that nobody has used MQTT for IIoT twins (this is close to default practice). The candidate gap, examined against the literature in Section III, is narrower: the **combination** of (a) empirically calibrated, not random and not purely ML-trained, causal machine simulators; (b) explicit separation of machine-twin, process-twin, and platform layers at the architecture level; and (c) a measured evaluation of synchronization and scaling behavior specifically on self-hostable, commodity-scale infrastructure for a multi-station discrete line. Individual pieces of this exist in isolation across the literature surveyed in Section III; the combination applied to a five-station automotive line does not appear to.

F. Contributions

- An architecture (Live Twin / Scenario Worker split, DAG process topology, frozen-snapshot semantics) for composing independently deployable machine twins into a process twin without a monolithic factory simulator.
- A reduced-order, causally structured simulation model for each of five station types, with variables classified Must, Useful, or Omit, and calibration sources identified per variable.
- An empirically grounded methodology for using public PHM, UCI, and Kaggle datasets as calibration sources, explicitly distinguished from dataset replay and from machine-learning training.
- An experimental protocol, specified in full, for measuring synchronization latency, Kubernetes scaling and recovery behavior, and single-host feasibility on commodity hardware; execution and quantitative results are pending deployment, as detailed in Section XI.

G. Research Questions

Ten research questions guide this study. Each is addressed in the section indicated: RQ1 (architecture) in Section IV; RQ2

(machine modeling) in Section VIII; RQ3 (empirical calibration) in Sections VI–VII; RQ4 (synchronization) in Section XI-A; RQ5 (scalability) in Section XI-C; RQ6 (resilience) in Sections XI-C–XI-D; RQ7 (self-hosted deployment) in Section XI-D; RQ8 (digital twin utility) in Sections IX and XII; RQ9 (predictive maintenance) in Section XI-E; and RQ10 (SME applicability) in Section XIII.

II. DIGITAL TWIN DEFINITIONAL FRAMEWORK

This section addresses a concern raised directly against purely simulated digital-twin systems: that they may read as "too abstract" to count as a digital twin at all. That concern deserves a direct, careful answer rather than a rhetorical one — the honest answer has real limits, and stating them precisely is more defensible than asserting the term applies unconditionally.

A. Four Things That Are Not the Same Thing

Table I distinguishes four terms this paper uses precisely and does not treat as interchangeable: the machine simulator, the machine digital twin, the process twin, and the factory/platform layer.

TABLE I. TERMINOLOGICAL DISTINCTIONS USED IN THIS WORK

Term	Definition (this work)	What It Is Not
Machine Simulator	Software that generates synthetic telemetry and state transitions for one machine, following a causal internal model.	Not persistent, not addressable, not queryable independent of a run. It is the source a twin observes.
Machine Digital Twin	A persistent service that maintains state for one machine, ingests telemetry (from the simulator, or in a future revision from a real PLC/OPC-UA source), exposes that state through a stable API/schema, and accepts commands.	Not the simulator itself — a twin that lost its simulator would still exist as a stateful service with a last-known state, the way a real machine's twin persists across the machine being powered off.
Process Twin	A representation of the dependency relationships between station twins (the DAG) — what propagates from Stamping's output into CNC's input, etc. — plus enough aggregate state to answer line-level questions no single machine twin can answer alone.	Not a sixth "factory" simulator; it does not simulate physics, it composes twin outputs.
Factory/Platform Layer	Kubernetes orchestration, the MQTT broker, persistence, the dashboard, and the scenario/analytics engine.	Not a digital twin at all — it is infrastructure that digital twins run on. Calling the dashboard "the digital twin" is among the most common conflations in weaker digital-twin papers and is avoided throughout this work.

B. Does a Twin of a Simulated Machine Qualify as a Digital Twin?

Stated plainly: **partially, and the honest boundary is worth stating rather than talking around.**

The strongest formal definitions of a digital twin — Grieves' original Product Lifecycle Management-derived formulation, and the NASA/USAF framing in Glaessgen and Stargel's airframe digital-twin work [1] — center the concept on a twin that mirrors a physical counterpart through a live, bidirectional data connection. A twin whose only counterpart is a simulator run in the same cluster does not satisfy that mirroring relationship: there is no physical asset being reflected, so in the strictest reading this is a **twin of a simulated asset**, sometimes formally distinguished in the literature as a digital model or digital shadow depending on whether the data flow is bidirectional.

What the architecture does legitimately satisfy, independent of whether the counterpart is physical:

- State persistence independent of the data source. The twin maintains reported, desired, and live state (the same three-way distinction Eclipse Ditto formalizes) regardless of whether that state currently comes from a simulator or, in a future hardware-anchored revision, an OPC-UA tag on a real PLC.
- A stable schema and API contract. Swapping the simulator for a real sensor feed is a data-source change behind the same twin interface, not an architecture change — which is itself evidence the twin/simulator separation above is doing real work, not just terminological hygiene.
- The command channel (Section X-B) is exercised identically whether the receiving end is a simulated CNC state machine or a real one, which is the actual object of study for RQ1 (twin architecture) independent of RQ7 (deployment feasibility).

The defensible framing: this work is a **twin-architecture and twin-behavior study**, where the physical-fidelity question is deliberately factored out so that the synchronization, orchestration, and process-composition questions can be answered cleanly, and where hardware-anchoring the CNC command channel to one real machine (Section X-B) is named explicitly as the smallest change that would move the strongest single station from digital model to digital twin in the strict sense. This is a stronger position than claiming the whole system consists unconditionally of "real" digital twins, and it directly pre-empts the abstraction objection by naming its scope rather than disputing that the objection has merit.

III. RELATED WORK

A. Digital Twins in Manufacturing (General)

The concept traces to Grieves' Product Lifecycle Management formulation (circa 2002–2003) and was formalized for engineering audiences through NASA/USAF airframe work [1], which is where the "virtual representation kept synchronized with a physical counterpart via live data" definition most literature still cites solidifies. Manufacturing-specific surveys trace the subsequent divergence into commercial platforms (GE Predix, since deprioritized by GE; Siemens MindSphere, now rebranded Siemens Insights Hub; PTC ThingWorx; IBM's IoT platforms) and open-source infrastructure such as Eclipse Ditto.

B. Automotive Manufacturing Digital Twins

Automotive is one of the most cited digital-twin application domains precisely because it combines discrete part flow (stamping, machining, welding — all present in the five stations modeled here) with high volume and tight tolerance requirements, making tool-wear and weld-quality prognostics economically material. DELMIA's PSA Peugeot Citroën body-

in-white case and Siemens' Brose Plant Simulation case are both real, publicly documented automotive-specific deployments of commercial digital-twin tooling (evidence class: vendor case study, not peer-reviewed).

C. Cloud-Native Digital Twin Architectures

A growing body of work treats "cloud-native" as a first-class architectural property rather than an implementation detail: containerized, independently deployable, horizontally scalable twin services. Fraunhofer IPT's cloud-native manufacturing architecture work (Sankal, Mohanram, König, and Schmitt, IEEE ICPS 2025) proposes a layered multi-cluster controller specifically for manufacturing OT/IT boundary requirements, which is directly relevant background for why a naive single-cluster design under-serves real shop-floor control/monitoring separation even at small scale.

D. Kubernetes-Based Digital Twins: A Critical Distinction

The literature on Kubernetes and digital twins splits into two genuinely different research problems, and conflating them is among the most common positioning errors a reviewer would flag.

1) Digital twins that run on Kubernetes:

(twins of external, physical, or simulated assets, using Kubernetes purely as the orchestration substrate):

- **KTWIN** (Wermann and Wickboldt, UFRGS, 2024) [2] — a serverless, Kubernetes-based platform for defining and deploying generic digital twins using open cloud-native tooling, reporting 60–80% cost savings versus statically overprovisioned deployments. This is the closest published analog to a Kubernetes-native platform for building twins, but it is domain-generic and event-driven-serverless rather than causal-manufacturing-specific.
- **IndustryFusion** (open source) [3] — explicitly SME-targeted IIoT connectivity with a semantic (NGSI-LD/RDF/OWL/SHACL) "Process Data Twin," Helm-chart Kubernetes deployment, and OPC-UA/MQTT gateway services running on an edge "smartbox." Closest match to this work's target market (SME manufacturing), though its emphasis is semantic interoperability rather than causal simulation.
- **OpenTwins** (Robles, Martín, and Díaz, Universidad de Málaga, Computers in Industry, 2023) [4] — a compositional digital-twin framework combining Kafka-ML for predictive models and Unity for 3D visualization, demonstrated on a petrochemical freezing-point-prediction use case.
- **OFacT / Open Factory Twin** (Fraunhofer-linked, Apache-2.0, Python) [5] — a simulation-based digital twin for production and logistics material flow using discrete-event and agent-based simulation. The closest published open-source project to this work's domain, though built on agent-based/discrete-event simulation rather than SimPy specifically.
- **DT-Forge** (PyPI package) — a six-layer Python digital-twin framework whose simulation layer explicitly lists SimPy alongside SciPy ODE solvers and ONNX/scikit-learn surrogates, synchronizing through Eclipse Ditto, with MongoDB/InfluxDB/Redis persistence. It is the closest technology-stack analog found in this search (SimPy plus a FastAPI-family plus a MongoDB-family stack), which sharpens rather than undermines the present contribution claim: it confirms the stack choice is sound and well-trodden, while this work's specific contribution is the manufacturing-calibrated, multi-station DAG application of that kind of stack.
- **Eclipse Ditto** (Eclipse IoT) — not Kubernetes-specific, but the de facto open-source reference for what a twin service does: Java microservices (Policies, Things, Things-Search, Gateway, Connectivity, Concierge) backed by MongoDB, connecting over MQTT (via Eclipse Hono) or REST/WebSocket, and formalizing the reported/desired/live three-way state distinction this work adopts in Section V. Its documented minimum footprint for a single local instance is approximately 2 CPU cores and 4 GB of RAM — a concrete data point for the single-host feasibility question in Section XI-D.

2) Digital twins of Kubernetes itself:

(Kubernetes-the-orchestrator is the object being twinned, for autoscaling or configuration optimization — categorically different from the category above):

- **KAPETÁNIOS** (Zerwas, Krämer, Ursu, Asadi, Rodgers, Wong, and Kellerer, TU Munich/Rakuten Mobile, NoF 2022) [6] — a digital twin of a Kubernetes cluster used to autonomously retune the Horizontal Pod Autoscaler, reporting a 37% reduction in total pod-seconds with request latency largely unaffected.
- **KubeTwin** (Foschini, Poltronieri, Stefanelli, and Tortonesi, 2023, concept paper [7]; full validated framework in Borsatti et al., IEEE Transactions on Network and Service Management, 2024 [8]) — a simulation-based digital twin of Kubernetes deployments, validated in a multi-access edge computing scenario.
- Related: TELKA (Twin-Enhanced Learning for Kubernetes Applications, same research group) and a Portuguese-language K8s-DT paper using stochastic models — both further instances of this category.

Why this distinction matters for situating the present contribution, stated precisely: this work's twins have Kubernetes as their deployment substrate and manufacturing machines as their twinned subject. KAPETÁNIOS and KubeTwin have Kubernetes itself as the twinned subject, evaluated through simulation for autoscaling and configuration-optimization purposes. These solve different problems with superficially similar names, and a reader skimming "digital twin" together with "Kubernetes" without this split would reasonably, and wrongly, assume they compete with this work. They do not: they are adjacent, not overlapping, and citing both categories while stating the distinction explicitly is a stronger position than citing only the first category and hoping the distinction goes unnoticed.

E. MQTT / IIoT Synchronization

MQTT as the default telemetry transport for constrained IIoT devices is close to unchallenged practice at this point (publish/subscribe, quality-of-service tiers, low overhead) rather than a novel choice; its treatment in Section X is accordingly about topic design, quality-of-service selection, and failure behavior, not about whether to use MQTT at all. Sparkplug B (an Eclipse Foundation/Cirrus Link specification layered on MQTT) adds standardized birth/death-certificate semantics and a defined payload schema for industrial telemetry, which is a candidate for Section X if the ad hoc topic hierarchy proves insufficiently self-describing at multi-consumer scale — noted here as a design option, not adopted by default.

F. Predictive Maintenance

CNC tool-wear monitoring integrated directly with a digital twin is an active, well-published sub-area. Ströbel, Bott, Wortmann, and Fleischer [9] propose a cycle-segmentation, anomaly-detection approach to tool and component wear, computed on-machine and transmitted as a condensed wear matrix to a remote twin — directly informative for the CNC degradation-signal design in Section VIII-B. A 2025 study in the Journal of Intelligent Manufacturing reports an LSTM-based real-time tool-life predictor achieving 33.17 µm RMSE in training and 119.36 µm RMSE in real-time deployment on a milling dataset — a useful order-of-magnitude reference point for the predictive-maintenance evaluation in Section XI-E.

G. Digital Twin Synchronization (Evaluation Literature)

Formal, measured synchronization-latency studies specifically for multi-twin manufacturing deployments are comparatively rare relative to the volume of architecture papers; most cited works propose an architecture and demonstrate feasibility qualitatively rather than reporting P95/P99 latency distributions under controlled network degradation. This is itself a modest evidentiary observation supporting the contribution claim in Section XIV: a measured synchronization evaluation (Sections XI-A–XI-B) is less crowded ground than architecture proposals are.

H. SME Digitalization

Covered in full in Section XIII; the key sources are the Change2Twin "nine barriers" position paper (a multi-national expert team led by Paolo Pileggi, TNO) [10] and Ragazzini, Negri, and Macchi's 2025 low-cost data-integration framework for make-to-order SMEs (Politecnico di Milano) [11], both of which frame SME barriers as organizational and integration-complexity issues at least as much as raw infrastructure cost.

I. Open-Source and Commercial Platforms: A Comparison

Table II situates this work against the managed commercial platforms and the self-hosted/open-source frameworks identified in the literature search.

TABLE II. COMPARISON OF RELATED DIGITAL TWIN PLATFORMS

Platform	Deployment	Twin Modeling	Communication	Edge	Self-Hosted	Orchestration	Analytics	Open Source	Cost Model
Azure Digital Twins	Managed cloud (PaaS)	DTDL twin graph, relationships	IoT Hub / Event Grid, REST/query API	Via IoT Hub device twins	No	Azure-managed	Power BI, 3D Scenes Studio (preview)	No	Usage-based (ops/messages/queries)
AWS IoT TwinMaker	Managed cloud	Knowledge graph (entities/components)	Data connectors (SiteWise, Kinesis Video, Timestream, third-party)	Limited	No	AWS-managed	Managed Grafana, 3D scenes	No	Per API/entity/query — status caveat, see note
Siemens Xcelerator / Tecnomatix	On-prem, cloud, or hybrid (SMB cloud tier exists)	3D process/robotics/plant simulation	Vendor-specific; Siemens Insights Hub for IIoT	Yes (industrial ecosystem)	Partial	Vendor tooling	Built-in KPIs; NVIDIA Omniverse-based Digital Twin Composer (2026)	No	Enterprise license / subscription
PTC ThingWorx	On-prem or cloud	Thing / Thing Template / Thing Shape model	Kepware middleware, OPC-UA, MQTT	Yes (Kepware edge)	Partial	Vendor tooling	ThingWorx Analytics (separate license), Vuforia AR	No	Enterprise quote, no free tier
IBM Maximo Application Suite	Cloud-based	EAM/CMMS asset model; digital twin is an integration point	IoT data integration	Limited	No	Vendor tooling	AI-infused anomaly/health scoring	No	Committed-term / AppPoints licensing
Dassault DELMIA (3DEXPERIENCE)	Cloud or on-prem	"Virtual twin experience," process/robotics/MOM (Apriso)	Vendor-specific	Yes	Partial	Vendor tooling	Built-in; NVIDIA partnership	No	Subscription incl. SMB tiers
Eclipse Ditto	Self-hosted (Docker/K8s)	Thing model, reported/desired/live state	MQTT (via Hono), REST, WebSocket	Yes	Yes	Not opinionated	None built-in	Yes (EPL-2.0)	Free / infrastructure cost only
KTWIN	Self-hosted Kubernetes (serverless)	Generic digital-twin model, open standards	Event-driven	Not evaluated	Yes	Kubernetes-native (core contribution)	Not a focus	Yes	Free / infrastructure cost only
IndustryFusion	Self-hosted Kubernetes (Helm)	Semantic NGSI-LD/RDF process data twin	OPC-UA + MQTT gateways	Yes (edge "smartbox")	Yes	Kubernetes-native	Limited	Yes (SME-targeted)	Free / infrastructure cost only
OpenTwins	Self-hosted (Docker)	Compositional 3D-IoT-AI twins	Kafka-ML	Planned (future work)	Yes	Kafka-based, not Kubernetes-specific	ML predictions via Kafka-ML	Yes	Free / infrastructure cost only
OFacT	Self-hosted (Python)	Agent-based/discrete-event production and logistics model	Not Kubernetes-native	Not a focus	Yes	Not Kubernetes-native	Excel integration for KPIs	Yes (Apache-2.0)	Free / infrastructure cost only
This work	Self-hosted Kubernetes	Live Twin / Scenario Worker split, DAG process topology, causal per-machine models	MQTT	Designed for, not yet validated	Yes	Kubernetes-native	Scenario/what-if engine (Section XII)	Yes (student project)	Free / infrastructure cost only

Evidence classification: platform capabilities are drawn from vendor documentation (Azure, AWS, PTC, IBM, Dassault, Siemens) or from project documentation and peer-reviewed sources (Ditto, KTWIN, IndustryFusion, OpenTwins, OFacT), current as of a search conducted in September 2026; readers verifying specific vendor claims should consult current vendor documentation, as such details are subject to change. On AWS IoT TwinMaker specifically: a reseller catalog source describes the service as no longer accepting new customers as of early 2026, amid a broader wind-down of adjacent AWS IoT services (Fleet Hub end-of-life October 2025, IoT Analytics end-of-life December 2025, IoT Events end-of-life May 2026); no official AWS end-of-life notice for TwinMaker itself was found in this search, and the status should be treated as unconfirmed pending direct verification.

J. Research Gap, Restated Precisely

No single platform in Table II combines self-hosted Kubernetes-native orchestration, causal (non-random, non-purely-black-box) machine simulation, empirical calibration against public industrial datasets, explicit machine/process/platform layer separation, and a measured multi-station synchronization and scaling evaluation. Individual rows satisfy some of these; none in this search satisfies all. That combination, not any single element of it, is the candidate contribution — consistent with the more cautious treatment in Section XIV of what counts as genuinely novel versus merely a sound engineering choice.

IV. PROPOSED ARCHITECTURE

A. The Live Twin / Scenario Worker Split

The architecture adopted for this work separates two roles that most reference architectures, including KTWIN and OFacT, leave merged inside a single "twin service":

- **Live Twin** — the persistent, addressable service holding one machine's current reported/desired/live state (Section V-A), subscribed to that machine's MQTT telemetry topics, and exposing the command channel (Section X-B). Exactly one Live Twin exists per machine, at all times, independent of whether anyone is running a what-if scenario.
- **Scenario Worker** — an ephemeral, disposable process that forks state from one or more Live Twins at a point in time and runs a hypothetical forward simulation (Section XII) without ever writing back to the Live Twins it forked from.

Why this split is the right call, evaluated rather than assumed: the alternative — running what-if scenarios directly against Live Twin state — creates two failure modes this design avoids by construction. First, a long-running scenario would either block real telemetry ingestion or require the Live Twin to interleave real and hypothetical state, which is exactly the kind of state-correctness bug class (which update is "real"?) that is expensive to debug in a distributed system. Second, without a hard fork point, a scenario cannot be repeated against the same starting conditions once real telemetry has moved on, which would compromise reproducibility for the one part of the system that most needs it. The cost of the split is real and worth naming: a Scenario Worker's fork is a snapshot, so any Live Twin state change during a running scenario is, by design, invisible to that scenario — which is precisely the frozen-snapshot semantics below, not an accidental side effect.

B. DAG-Based Process Topology with Frozen-Snapshot Semantics

The process twin (Section IX) represents station dependencies as a directed acyclic graph (DAG) — Stamping → CNC → Weld/Assembly → Quality Inspection → Packaging/Dispatch — rather than a generic graph or a full discrete-event factory simulator. **Frozen-snapshot semantics** means that when the process twin, or a Scenario Worker, needs a consistent view of the whole line's state to answer a query or run a scenario, it takes a single frozen snapshot of every station's state at one instant rather than reading each station's live state independently and non-atomically. This avoids the classic distributed-snapshot inconsistency at the cost of that snapshot being, by definition, immediately stale — a trade-off worth stating explicitly rather than treating as free.

Table III summarizes the alternative designs considered for this layer and the reasons each was not adopted, consistent with a design process that evaluates rather than uncritically accepts the initially proposed architecture.

TABLE III. ALTERNATIVE ARCHITECTURAL DESIGNS CONSIDERED

Alternative	Why Considered	Why Not Adopted
Single monolithic factory simulator (one process modeling all five stations)	Simplest to reason about; no distributed-state problem at all.	Directly contradicts RQ1's premise of independently deployable machine twins and the goal of independent scaling and failure recovery — a monolith cannot be partially restarted or partially scaled.
Fully generic dependency graph (allows cycles/rework loops)	More general — real factories do have rework and scrap loops.	Adds real complexity (cycle detection, convergence semantics) the five-station base case does not need; a DAG is the correct minimal structure, and generalizing it is future work (Section XVII).
Event-sourced/CQRS full history replay instead of frozen snapshots	Gives perfect historical replay for free.	Substantially more implementation surface for a benefit that scenario simulation from a single snapshot point does not require; revisit only if Section XII proves insufficient.
Push-based propagation (each twin pushes directly to the next)	Removes the process twin as a component entirely.	Collapses the machine-twin/process-twin separation argued in Section II to be conceptually load-bearing, and makes "what happens if a station goes down" require polling every twin.

C. System Architecture

Fig. 1 shows the full system: a simulated physical layer of five station simulators, an MQTT broker mediating telemetry and commands, and a Kubernetes cluster hosting the Live Twins, the Process Twin, the Scenario Worker, an analytics/predictive-maintenance component, persistence, and a dashboard API.

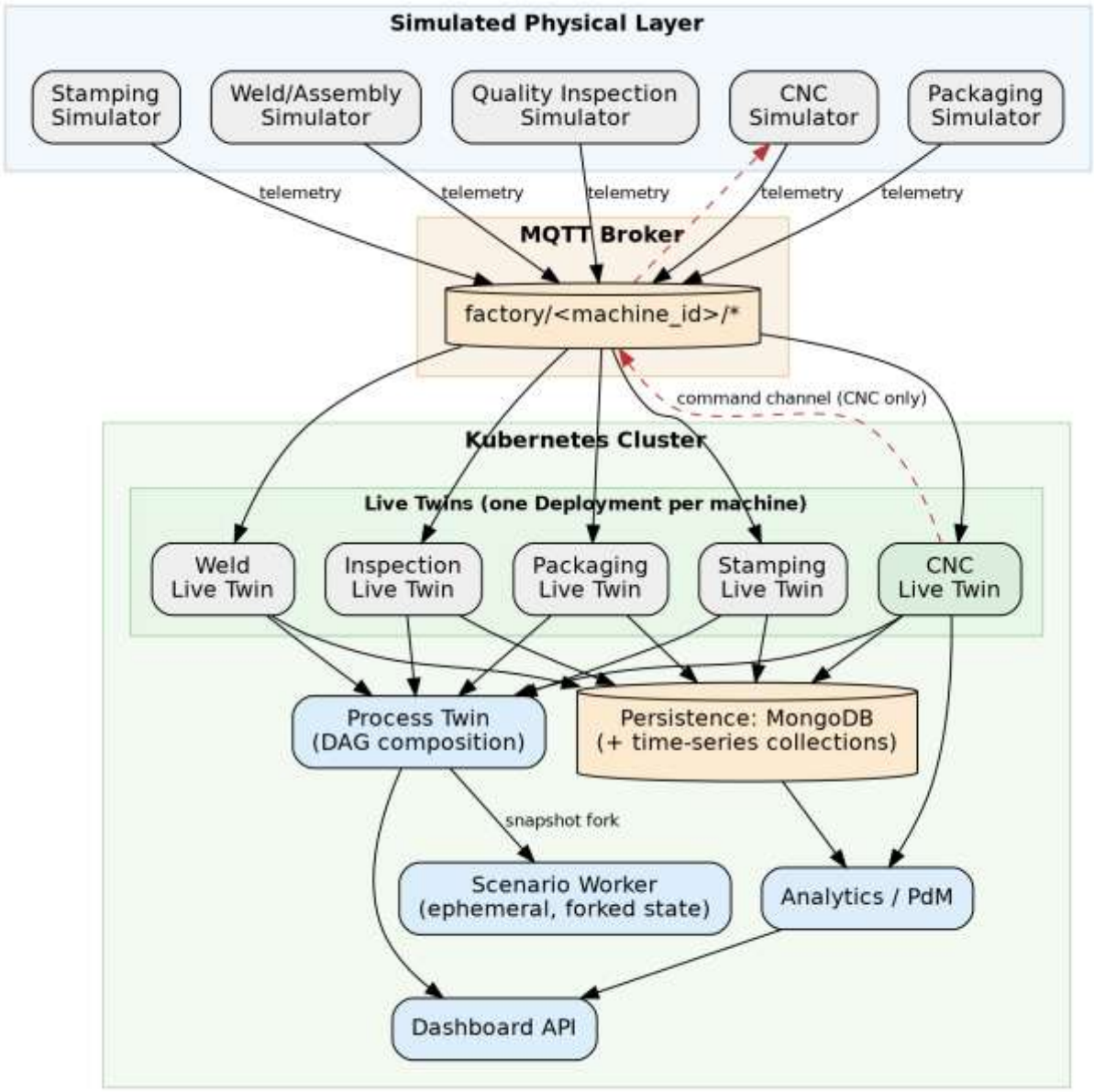


Fig. 1. System architecture: the simulated physical layer, the MQTT broker, and the Kubernetes cluster hosting Live Twins, the Process Twin, the Scenario Worker, Analytics, and the Dashboard API.

Fig. 2 details the production-token flow between station Live Twins and each twin's state contribution into the Process Twin, corresponding to the five-machine process described in Section VIII.

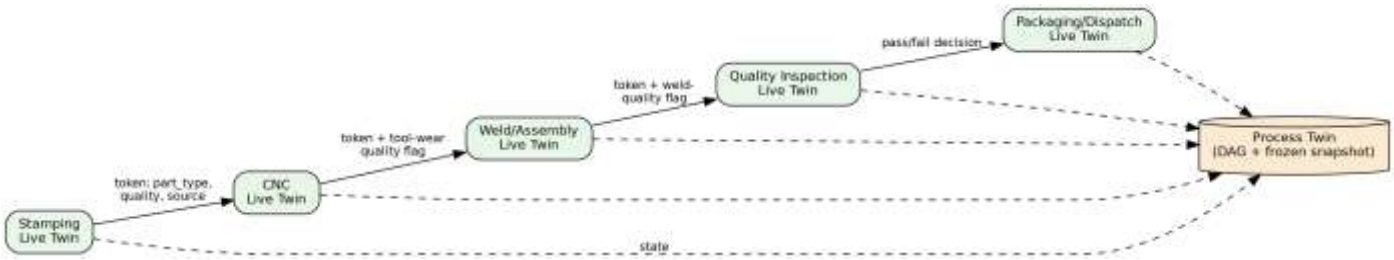


Fig. 2. Production-token flow between station Live Twins, and each twin's state contribution into the Process Twin.

V. STANDARDIZED MACHINE/TWIN DATA MODEL

A. Category Schema

Table IV adopts seven categories, cross-checked against the reported/desired/live state distinction Eclipse Ditto formalizes and against ISO 23247's Observable Manufacturing Element (OME) categories (Personnel, Equipment, Material, Process, Facility, Environment, Product, Supporting Document; ISO 23247-3 [12]).

TABLE IV. MACHINE/TWIN STATE CATEGORY SCHEMA

Category	Contents	Update Cadence	Ditto State Type
Identity	machine_id, machine_type, station_index, schema_version	Static (set at deploy)	N/A
Operational State	Current state-machine node (e.g., RUNNING, FAULT)	Event-driven, low rate	live
Telemetry/Sensors	Continuous signals (vibration, current, temperature, force)	High rate, periodic	reported
Health	Derived indices (tool-wear %, health score) computed from telemetry	Derived, medium rate	reported
Production	Cycle count, part_type, cumulative output	Event-driven, per-cycle	reported
Faults	Active/historical fault codes, severity	Event-driven, low rate	reported
Commands	Pending/acknowledged commands (Section X-B)	Event-driven, operator- or scenario-issued	desired

This maps cleanly onto MongoDB: one collection per category is unnecessary overhead at this scale. A single twins collection keyed by machine_id, with these seven categories as sub-documents, plus a telemetry time-series collection (MongoDB 5.0+ native time-series collections, not a separate time-series database) for the high-rate Telemetry category, keeps the entire persistence layer inside the stack already in use rather than adding InfluxDB or TimescaleDB as a new dependency purely for this purpose.

B. Standards Trade-off: Adopt vs. Simplify

The honest finding here, stated by ISO 23247's own text, is that no single standard is meant to be adopted wholesale: ISO 23247-3 explicitly states that no single standard defines information for all OME types, naming ISO 10303 (STEP), IEC 62264 (ISA-95), the Asset Administration Shell, OPC-UA (IEC 62541), MTConnect, and QIF (ISO 23952) as components an implementer selects from. A NIST-cited survey of published manufacturing digital-twin architectures against the ISO 23247 reference architecture found that real implementations concentrate on functional entities and largely skip the standard's non-functional entities (security, plug-and-play device support, peer interfaces, access control) — that is, even published research prototypes do not fully implement the standard.

Decision for this work: adopt the conceptual OME categorization of Table IV without adopting the full ISO 23247 information-exchange machinery (Part 4) or a full Asset Administration Shell submodel implementation. This is the correct choice for a capstone-scale system for the same reason the survey above implies it is the modal choice for published research: full standards compliance is a substantial, mostly orthogonal engineering effort that does not itself answer any of the research questions in Section I-G. The schema in Table IV is presented explicitly as "ISO-23247-informed, not ISO-23247-conformant," a defensible, precisely scoped claim rather than an overclaim.

TABLE V. FIVE-MACHINE SIMULATION MODEL OVERVIEW (FULL DETAIL IN SECTION VIII)

Machine	Physical Components	Internal Variables	Sensors	Actuators	States	Faults
Stamping Press	VIII-A	VIII-A	VIII-A	VIII-A	VIII-A	VIII-A
CNC Machining Center	VIII-B	VIII-B	VIII-B	VIII-B	VIII-B	VIII-B
Weld/Assembly	VIII-C	VIII-C	VIII-C	VIII-C	VIII-C	VIII-C
Quality Inspection	VIII-D	VIII-D	VIII-D	VIII-D	VIII-D	VIII-D
Packaging/Dispatch	VIII-E	VIII-E	VIII-E	VIII-E	VIII-E	VIII-E

VI. MACHINE SIMULATION PHILOSOPHY AND EMPIRICAL CALIBRATION METHODOLOGY

A. The Causal Pipeline

Every station's simulator follows the same layered causal structure rather than sampling output variables directly, as shown in Fig. 3.

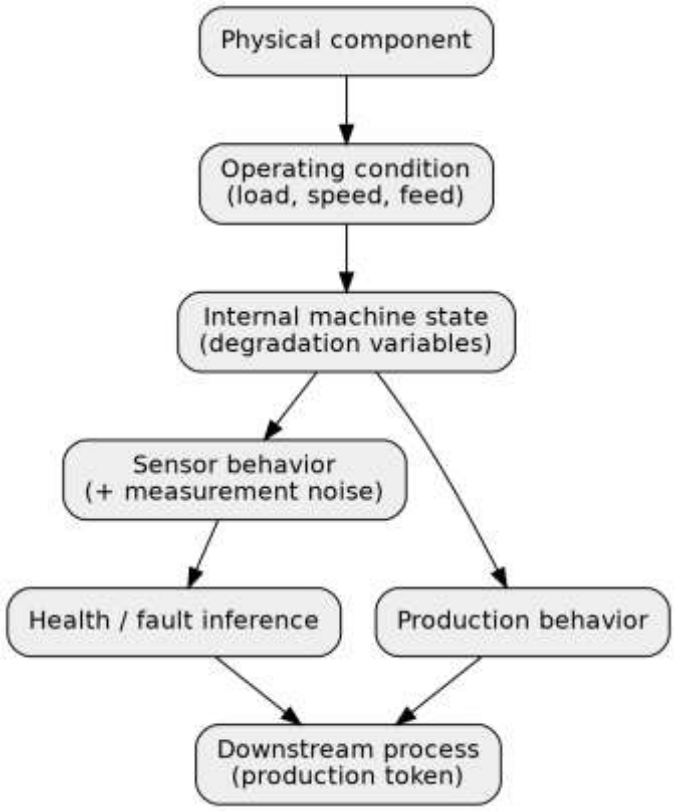


Fig. 3. The causal pipeline: physical component → operating condition → internal state → sensor behavior (with noise) → health/fault inference, with production behavior branching in parallel toward the downstream process.

A model of the form temperature = random.uniform(40, 80) is explicitly rejected anywhere in this system. Randomness enters only at the sensor-behavior layer, as measurement noise layered on top of a value that is itself a deterministic-or-stochastic function of the internal state and operating condition. The AI4I 2020 dataset's own generative logic [13] is a useful template for this: failure is not sampled directly, it is derived from physically meaningful products of state variables (power equals torque times angular velocity; overstrain equals tool wear times torque, compared against a product-type-dependent threshold), with randomness confined to which random-uniform time a wear-driven failure lands within a bounded window, and to a genuinely tiny (0.1%) irreducible random-failure term standing in for unmodeled causes. That is the level of randomness discipline this work imitates: physically derived causal structure with noise as a bounded residual, not noise as the model.

B. Dataset-Driven Calibration, Distinguished From Two Adjacent Things

- **Calibration** (what this work does): use a public dataset's statistical properties — variable ranges, noise distributions, degradation slopes, correlation structure between load and vibration, and so on — to set the parameters of a hand-written causal model. The simulator can generate trajectories the dataset never contained.
- **Dataset replay** (explicitly rejected as the simulation mechanism): feeding recorded dataset rows back out as if they were live telemetry. This produces a system that can only ever reproduce exactly the scenarios in the training data, which defeats the what-if objectives of Section XII entirely.
- **Machine-learning training** (explicitly a separate concern, addressed in Section XI-E): fitting a predictive model to detect anomalies or estimate remaining useful life from generated or real telemetry. This is downstream of the simulator, consumes its output, and does not itself generate machine behavior. A threshold-based fault detector and a hand-tuned causal simulator are both present in this architecture and are not the same component.

C. Calibration Methodology (Per Variable, Per Machine)

The calibration pipeline proceeds in fixed stages: a public dataset of raw signals and wear/fault labels is cleaned (resampled to a common rate, units normalized); explored (distribution shape, stationarity); reduced to features (RMS vibration, spectral peaks, rolling statistics); checked for correlation and dependence (does the dataset actually show load-to-vibration coupling?); analyzed for trend (is the degradation slope linear, exponential, or stepped?); analyzed for distribution (what noise distribution fits the sensor residual?); fitted to a simple empirical model (a low-order function fitted to the trend); and finally converted into a calibrated simulator parameter, where the fitted coefficients become constructor arguments to the causal model of Section VIII, not a lookup table of dataset rows.

This methodology is intentionally the same regardless of which of the datasets in Section VII feeds it; what changes per dataset is which stage yields usable output and which stage the dataset is too mismatched to inform, flagged explicitly per dataset in Section VII rather than glossed over.

VII. PUBLIC DATASET ASSESSMENT

Table VI assesses the five public datasets considered as calibration sources for the machine models of Section VIII, including one (NASA C-MAPSS) retained deliberately despite a poor domain match, for the reason stated in its row.

TABLE VI. PUBLIC DATASET ASSESSMENT

Dataset	Machine/Process	Signals	Degradation Variable	Real/Synthetic	Relevance	Intended Use Here	Limitations
PHM Society 2010 Milling Data Challenge (IEEE DataPort/Kaggle, CC0) [14]	High-speed CNC milling, ball-nose cutter, dry milling	3-axis dynamometer, 3-axis accelerometer, acoustic emission (7 channels)	Flank wear (µm) after each cut, 6 cutters (c1–c6)	Real	High — direct match to the CNC station (Section VIII-B)	Calibrate CNC wear-to-vibration and wear-to-force coupling; wear-progression slope shape	Only 6 tool records (small n); single machine/material
NASA IMS Bearing Dataset (University of Cincinnati IMS Center / NASA PCoE) [15]	Rexnord ZA-2115 bearings, 2000 RPM, 6000 lbs radial load	Vibration, 20 kHz, 20,480 points/file, 3 run-to-failure sets	Bearing fault progression to failure	Real	Moderate to high — CNC spindle bearings / generic rotating components	Calibrate CNC spindle-bearing degradation to vibration-signature relationship (Section VIII-B)	Test-rig geometry differs from a machine-tool spindle; shape, not absolute magnitudes, transfers

Dataset	Machine/Process	Signals	Degradation Variable	Real/Synthetic	Relevance	Intended Use Here	Limitations
AI4I 2020 Predictive Maintenance Dataset (Matzka 2020; UCI id 601, CC BY 4.0) [13]	Generic/synthetic milling-adjacent process	Air/process temperature, rotational speed, torque, tool wear, product variant	Five explicit causal failure modes (TWF/HDF/PWF/OSF/RNF)	Synthetic	Moderate — not a real machine, but a directly usable template	Worked example of deriving failure conditions from products of physical quantities (Section VI-A)	Cannot be cited as evidence real machines hit these specific thresholds — methodological value only
Bosch Production Line Performance (Kaggle, 2016) [16]	4 real lines, 52 stations (anonymized)	4,000+ anonymized numeric/categorical/date features	Binary pass/fail quality outcome (not degradation)	Real (anonymized)	Moderate, structural only — no physical-variable calibration possible	Structural analog for process-level design (Section IX): multiple flow paths, rare-event quality-failure base rate	Anonymization prevents calibrating any single physical variable in Table V
NASA C-MAPSS (Saxena and Goebel, NASA Ames PCoE, 2008) [17]	Simulated turbofan jet engine (MATLAB/Simulink)	21 sensor channels plus 3 operational settings, 4 subsets	Run-to-failure degradation of engine components	Synthetic	Low, explicitly mismatched — turbofan aerothermal physics does not transfer to any of the five stations	Not used for calibration. Remaining-useful-life methodology and benchmark reference only	Flagged explicitly given this mismatch, and retained only for the stated methodological purpose

Supplementary candidates were identified but not adopted as primary sources: the Kaggle "CNC Mill Tool Wear" dataset (University of Michigan SFI Lab, approximately 2,304 instances) as a second CNC source if the PHM-2010 six-cutter sample proves too small; and SECOM (UCI, 591 anonymized semiconductor-fabrication features) and PRONOSTIA/FEMTO-ST (17 accelerated bearing run-to-failure experiments) as alternative process-quality and bearing sources.

VIII. FIVE-MACHINE CAUSAL MODELING

Variable classification convention used throughout: **Must** denotes a variable required for the causal chain of Section VI-A to produce believable behavior; **Useful** denotes a variable that adds realism or supports a specific downstream analysis but is not required for the system to function; **Omit** denotes a variable present on a real machine but adding no research or demonstration value at this scope — explicitly not modeled, not "modeled crudely."

A. Stamping Press

Physical components represented: hydraulic/mechanical drive, ram/slide, die/tooling, feed system, press frame.

Causal structure: die wear (internal, monotonically increasing with cycle count) drives required press force (rising as die wear rises, to maintain part dimension), which drives motor current (a sensor reading, plus noise). Die wear also drives vibration (rising with die wear, sharper near end-of-life; a sensor reading, plus noise), which in turn drives the part-quality flag (degrading once press force exceeds a die-wear-dependent tolerance band). Separately, production load (cycles per hour) drives cycle time (roughly constant per part type) and hydraulic pressure (a secondary signal).

TABLE VII. STAMPING PRESS — VARIABLE CLASSIFICATION

Variable	Class	Notes
Die wear (internal)	Must	Drives everything else; monotonic increase, resettable on maintenance event
Motor current	Must	Directly derivable from required force; cheap, high-value sensor to simulate
Press force (required vs. actual)	Must	The Must-level causal link between die wear and everything downstream
Cycle time	Must	Needed to drive the process twin's throughput and part-token emission rate
Vibration	Useful	Adds a second, correlated observable for the anomaly-detection baseline of Section XI-E
Hydraulic pressure	Useful	Secondary confirmation signal; not load-bearing for the core narrative
Part quality flag	Must	The production-token field that propagates downstream (Section IX)
Feed-system jam fault	Useful	A discrete fault mode distinct from gradual die-wear degradation, for the fault-injection matrix of Section XI-B
Acoustic/noise-level sensing	Omit	Adds no causal or downstream value not already carried by vibration
Detailed hydraulic fluid temperature/viscosity model	Omit	A second full thermal model would duplicate effort already spent on the CNC coolant model (Section VIII-B)

States: IDLE → SETUP → RUNNING → (FAULT | MAINTENANCE) → RUNNING.

B. CNC Machining Center (Flagship Model)

This is the highest-detail model, deliberately, because it is the only station with two independent, well-matched real calibration datasets: PHM 2010 Milling for tool wear, and NASA IMS for spindle bearings (Section VII).

Causal graph (Fig. 4).

Operating load (feed rate, depth of cut) drives spindle motor current, temperature, and load-driven vibration. Tool wear, the internal degradation variable, independently drives cutting force, wear-driven vibration, wear-driven temperature rise, wear-driven current rise, and machining quality. Spindle bearing degradation (IMS-calibrated) drives a distinct bearing-signature vibration component, and coolant-system failure drives a temperature fault. All raw physical quantities converge on three sensor nodes (vibration, current, temperature), each adding measurement noise, while machining quality feeds the production token.

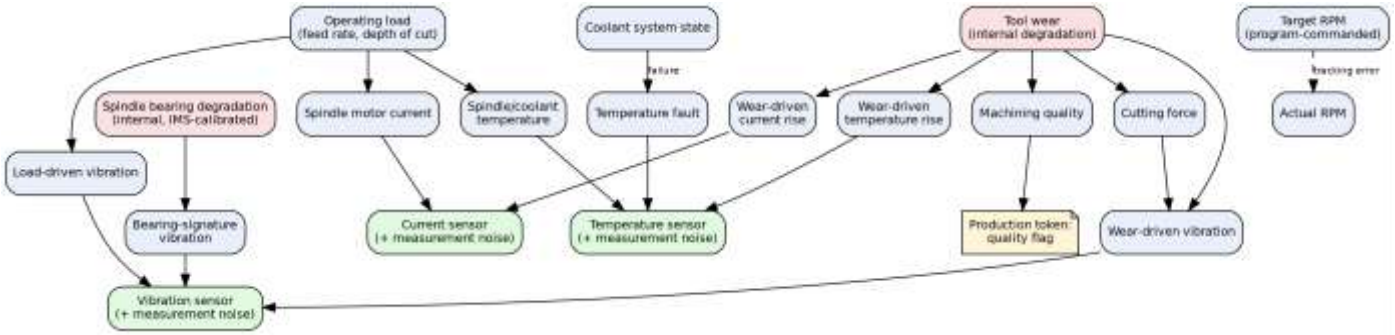


Fig. 4. CNC causal model: operating load and tool wear as independent drivers converging on vibration, current, and temperature sensors (each with measurement noise), plus the separately modeled target-vs-actual RPM tracking error.

Why target-vs-actual RPM is modeled separately: a real CNC controller commands a target spindle speed, and the drive tracks it with some error or lag, particularly under load or as bearing degradation increases drag. Modeling only a single RPM value collapses a genuinely useful fault signal, growing target/actual divergence as an early bearing-wear indicator, into nothing observable. The minimum causal form is rpm_actual = rpm_target minus a function of load and bearing degradation, plus noise.

State machine: IDLE → SETUP → LOADING → RUNNING → TOOL_CHANGE → RUNNING, with RUNNING → FAULT → MAINTENANCE → IDLE reachable from any running state. TOOL_CHANGE resets the tool-wear variable to zero and is triggered either by a wear threshold or an operator/scenario command.

TABLE VIII. CNC MACHINING CENTER — VARIABLE CLASSIFICATION AND CALIBRATION SOURCE

Variable	Class	Calibration Source
Tool wear (internal)	Must	PHM 2010 Milling — degradation-slope shape (near-linear with an accelerating tail near end-of-life)
Cutting force	Must	PHM 2010 Milling — dynamometer channel, wear-to-force coupling
Vibration (composite: load + wear + bearing)	Must	PHM 2010 (load/wear components) plus NASA IMS (bearing-signature component)
Spindle motor current	Must	PHM/general — current rises with force, a cheap high-value signal
Spindle/process temperature	Must	Qualitative shape from general machining literature; no dataset gives absolute values for this rig
Target vs. actual RPM	Must	Tracking-error scaled qualitatively against bearing degradation — a design choice, not an empirical claim
Bearing degradation (internal)	Must	NASA IMS — three independent run-to-failure trajectories give three plausible degradation-curve shapes
Machining quality (derived)	Must	The production-token field feeding Quality Inspection (Section VIII-D)
Coolant-failure fault mode	Useful	A discrete fault distinct from gradual wear; needed for the fault-injection matrix of Section XI-B
Tool-holder / tool-changer sub-model	Useful	Adds a believable TOOL_CHANGE duration distribution; not causally connected to anything else
Workholding fixture state	Omit	No calibration source, and no research question here requires this causal link
Full 3-axis servo-loop dynamics	Omit	Real control-theory depth with no dataset to calibrate against and no research question that needs it
Lubrication subsystem detail	Omit	Folded into the coolant-failure fault mode rather than modeled separately

C. Weld/Assembly Station

Causal structure: electrode wear (internal, monotonic) drives weld resistance/voltage drift, which drives weld quality. Clamp/fixture degradation (internal) drives clamp pressure variance, which drives part-presence/misalignment risk. Pneumatic system pressure (load-dependent) drives cycle-time variance.

TABLE IX. WELD/ASSEMBLY STATION — VARIABLE CLASSIFICATION

Variable	Class	Notes
Electrode wear (internal)	Must	Primary degradation driver for this station
Welding current	Must	Directly observable, wear-correlated
Welding voltage	Must	Paired with current; voltage/current ratio drift is a standard real weld-quality indicator — two correlated signals from one causal driver
Weld quality (derived)	Must	Production-token field
Clamp pressure	Useful	Secondary fault channel (misalignment) distinct from electrode wear
Pneumatic pressure	Useful	Feeds cycle-time variance, a minor realism addition
Part-presence sensor	Useful	Needed if a "missing part" discrete fault is desired for Section XI-B
Cycle time	Must	Throughput driver for the process twin
Robotic motion trajectory detail	Omit	This work models weld process signals, not robot kinematics — a real precedent (an AGH University of Science and Technology welding digital twin using machine-learning/fuzzy-logic control over welding power, deployed on Kubernetes with GitOps) confirms current/voltage-based control is the standard abstraction level for this station type

States: IDLE → SETUP → RUNNING → FAULT → MAINTENANCE → IDLE.

D. Quality Inspection Station

This station is modeled deliberately as an **abstraction over upstream state**, not a computer-vision pipeline. Whether that simplification is defensible is examined directly below.

The abstraction, stated as a causal chain: CNC tool wear (from the CNC Live Twin's state) drives machining quality, already computed in Section VIII-B. Weld quality (from the Weld Live Twin's state, Section VIII-C) is a second input. Both feed a defect probability, computed as a bounded function of machining quality and weld quality rather than a lookup table, and the inspection result is a Bernoulli draw against that defect

probability — the only point at which true randomness stands in for an entire unmodeled subsystem.

Is this defensible? Yes, with the scope of the claim stated precisely. The manufacturing literature robustly supports the qualitative direction of this relationship: tool-wear-driven part-dimensional and surface-quality degradation propagating into downstream inspection failure is exactly the causal chain the CNC tool-wear-monitoring literature (Section III-F) describes. What is not independently evidenced by any source in Section VII is the specific functional form mapping a numeric machining-quality score to a numeric defect probability; that mapping is a design choice (evidence class: proposed design choice, not peer-reviewed finding), stated explicitly rather than implying the function itself is empirically derived. Bosch Production Line Performance's sub-1% real-world end-of-line defect rate (Section VII) is used only to sanity-check that the chosen function's typical output is of a plausible order of magnitude, not as a source for the function's shape.

TABLE X. QUALITY INSPECTION STATION — VARIABLE CLASSIFICATION

Variable	Class	Notes
Defect probability (derived, upstream-fed)	Must	The core abstraction
Inspection result (pass/fail)	Must	Bernoulli draw against defect probability; production-token field
Inspection cycle time	Useful	Minor throughput contribution
Full machine-vision / dimensional-gauging model	Omit	Explicitly rejected; the upstream abstraction described above is the deliberate substitute, not an oversight

States: IDLE → RUNNING → (PASS | FAIL routing) → IDLE.

E. Packaging/Dispatch Station

Causal structure: minimal by design, since this station represents downstream production rather than a second complex machine. The inspection result gates entry (only PASS parts reach this station), driving the conveyor/counting state and the cumulative dispatched count; packaging cycle time is roughly constant, with minor load variance.

TABLE XI. PACKAGING/DISPATCH STATION — VARIABLE CLASSIFICATION

Variable	Class	Notes
Cumulative dispatched count	Must	Final throughput metric for the whole line
Conveyor state (running/jammed)	Useful	One simple discrete fault mode for completeness of the fault-injection matrix
Packaging cycle time	Useful	Minor
Weighing/counting sensor	Useful	Cheap, believable, ties count to a sensor rather than an internal-only counter
Barcode/RFID and dispatch-logistics interface	Omit	Explicitly excluded by design — adds logistics complexity with no bearing on any research question

States: IDLE → RUNNING → FAULT → RUNNING.

IX. PROCESS-LEVEL DIGITAL TWIN

A. The Production-Token Abstraction

Station-to-station propagation uses a lightweight token rather than full part geometry, unique serialized part IDs, or a discrete-event factory simulator engine: a JSON object carrying token_id, part_type, quality, source_station, emitted_at, and a carried_flags object with per-station quality scores (for example, machining_quality_score and weld_quality_score).

Sufficiency argument: every downstream causal dependency identified in Section VIII (tool-wear → machining quality → defect probability → inspection rejection → packaging throughput impact) is expressible entirely in terms of a numeric quality score and a categorical pass/fail outcome carried on this token. None of the research questions in Section I-G require tracking an individual part's full geometric history or a unique identity beyond what is needed to correlate an inspection failure back to which station degraded it. Building a full discrete-event simulator (part queues, blocking/starvation logic, buffer capacities) would be a legitimate addition for a throughput-optimization research question, but is explicitly not needed here, and is named in Section XVII as a natural extension.

B. What the Process Twin Computes That No Single Machine Twin Can

This directly answers RQ8. A process twin composing the DAG of Section IV-B can answer, from Live Twin state alone, questions no single machine twin's state contains:

- **Bottleneck attribution**: which station's current cycle time is the binding constraint on line throughput right now, requiring a comparison of all five stations' current cycle times.
- **Propagated-defect tracing**: given a spike in Quality Inspection's failure rate, which upstream station's quality score most correlates with the spike over the recent window, requiring a join of CNC and Weld state against Inspection outcomes over time.
- **Downstream-impact scenarios** (Section XII): for example, if CNC-01 fails now, when does Packaging's output rate visibly drop — a question requiring the DAG's edge structure plus each station's buffering assumption, not any single station's state.

C. Only Physically Meaningful Dependencies

By design, the DAG's five edges correspond exactly to the five physical material-flow edges in the stated production sequence. No additional edges, such as a synthetic "packaging depends on inspection's sensor calibration schedule" dependency, are introduced without a physical justification traceable to the causal models of Section VIII.

X. COMMUNICATION ARCHITECTURE (MQTT)

A. Topic Hierarchy and Quality of Service

TABLE XII. MQTT TOPIC HIERARCHY AND QOS ASSIGNMENT

Topic	QoS	Notes
factory/<machine_id>/telemetry	0	High-rate sensor streams; loss-tolerant — a missed vibration sample is not individually significant
factory/<machine_id>/state	1	Operational state-machine transitions; must arrive, duplicates tolerable
factory/<machine_id>/fault	1, retained	A restarted twin (Section XI-B) immediately sees the last fault state without waiting for the next event

Topic	QoS	Notes
factory/<machine_id>/production	1	Production-token emission
factory/<machine_id>/command	1 or 2	Depends on command criticality; Section X-B discusses this per command

QoS 0 for telemetry, not QoS 1 or 2 everywhere: this is a deliberate choice worth justifying rather than defaulting to "higher is safer." Telemetry in this system is periodic and high-rate; a single dropped sample is invisible in the health/state derivations of Section VIII (computed over rolling windows), while QoS 1 or 2's acknowledgment overhead on every message would be a self-inflicted synchronization-latency cost for a robustness guarantee this signal class does not need. State, fault, and production messages are comparatively rare and each one is individually meaningful, which is exactly the profile QoS 1 is for.

Retained messages are used specifically for the fault topic, so that a twin restarting mid-fault recovers correct state immediately, and are deliberately not used for telemetry (which would waste broker memory) or state (a retained stale RUNNING message would actively mislead a twin that just restarted after the machine has since faulted).

Reconnection and persistent sessions: each Live Twin connects with a persistent MQTT session (clean_session = false) so that fault, state, and production messages published while a twin pod is down (Section XI-C) are queued by the broker and delivered on reconnect rather than silently lost — precisely the mechanism whose latency the recovery-time metric in Section XI-C measures.

MQTT over WebSockets is not required for machine-to-broker or twin-to-broker traffic, since both run inside the cluster or simulated-network boundary, but is the natural choice if the dashboard ever subscribes to live topics directly from a browser — noted as a future option, not built now.

On REST as a fallback: REST is used in this architecture only for the Dashboard API and for scenario-worker control, both request/response, human-or-UI-triggered interactions that MQTT's publish/subscribe model does not fit naturally. It is not proposed as a redundant transport for telemetry, and "MQTT plus REST" is not claimed anywhere in this work as a novel combination; it is default, unremarkable practice, correctly not claimed as a contribution.

B. Bidirectional Interaction: One Fully Implemented Command Path

By design, only the CNC command channel is fully implemented, rather than building symmetric command support for all five machines: Dashboard/Scenario Worker → factory/CNC-01/command → {START, STOP, RESET_FAULT, SET_RPM}. The CNC Live Twin validates the command against its current state machine (Section VIII-B), forwards it to the CNC simulator, which acknowledges via factory/CNC-01/state.

Why CNC, and why one is sufficient: the CNC station has the richest state machine (Section VIII-B) and the most causally interesting command, SET_RPM, which directly exercises the target-vs-actual RPM tracking-error model. Demonstrating command validation, forwarding, and acknowledgment end-to-end on the single richest station is sufficient to establish that the architecture pattern (RQ1) works; replicating identical plumbing four more times would add implementation hours without adding evidence for any research question, and is named explicitly in Section XVII as a mechanical extension rather than a research gap.

Classification, stated precisely: this system is an **interactive digital twin** (commands are accepted and change simulated machine behavior) but explicitly **not a closed-loop digital twin** in the industrial-control sense, since there is no continuous automatic feedback loop in which the twin autonomously issues corrective commands based on its own state; every command originates from a human operator or an explicit Scenario Worker action. Describing this as "closed-loop control" would be an overclaim; "an interactive, human- or scenario-triggered command channel, demonstrated on one station" is the accurate and still substantive claim.

XI. EXPERIMENTAL METHODOLOGY AND EVALUATION PROTOCOL

Every subsection below specifies a complete, executable protocol. None of the numeric results are invented: where a result would go, this paper states explicitly that it is **pending deployment data** and specifies exactly what to run and what to record. Filling these in is mechanical once the cluster exists; the intellectual work of deciding which metric, why, and at what granularity is complete now.

A. Synchronization Metrics

TABLE XIII. SYNCHRONIZATION METRICS

Metric	Definition	Instrumentation
End-to-end telemetry latency, L	t(dashboard update) − t(simulator event)	Timestamp embedded at generation; compared against Dashboard API receipt/render timestamp. Reported as mean, median, P95, P99 — tail latency matters most for an operator-facing dashboard
MQTT delivery latency, L_mqtt	t(broker received) − t(message created)	Broker-side timestamping (plugin/hook) or client-side round trip via a loopback subscriber
Twin processing latency, L_twin	t(twin updated) − t(message received)	Instrumented inside the Live Twin's message handler
State freshness	Age of a twin's latest stored state relative to simulated time, sampled at query time	Computed on demand by the Dashboard API or a monitoring probe
Synchronization error, E(t)	A weighted sum of absolute differences between snapshot state and twin state, computed only where an independent reference state exists	Terminology note: since both sides originate from the same simulator pipeline, this is a snapshot-consistency error, not a physical-to-reality fidelity metric — fidelity to reality is the subject of Section XI-F

[Pending deployment data: L, L_mqtt, L_twin, and state-freshness distributions at baseline telemetry rate.]

B. Network-Degradation Experiments

TABLE XIV. NETWORK-DEGRADATION EXPERIMENTAL CONDITIONS

Condition	How Induced	Metrics to Record
Artificial latency	tc netem delay on the broker's network namespace	L, L_mqtt shift
Packet loss	tc netem loss X%	Message-loss rate, state-freshness impact
Message bursts	Scripted telemetry-rate spike on one or more simulators	Queueing delay; whether QoS-0 telemetry loss increases disproportionately
Broker outage	Kill the MQTT broker pod/process for a fixed window	Recovery time once the broker returns; whether persistent sessions (Section X-A) correctly redeliver queued QoS-1 messages
Duplicate delivery	Force broker-side redelivery (QoS 1 at-least-once is expected to duplicate on reconnect)	Confirm Live Twin handlers are idempotent — duplicates must not double-count a token or re-trigger a transition

[Pending deployment data: the full row of degraded-vs-baseline metrics per condition.]

C. Kubernetes Scaling Experiments

Twin-count progression: 1 → 5 → 10 → 25 → 50. This work's own five-station baseline is the "5" point; higher points scale by running multiple simulated lines in parallel, a legitimate stress test even though only one line is operationally meaningful — noted explicitly so the scaling curve is not misread as fifty machines in one factory.

Recorded at each step: CPU utilization (per-pod and cluster-aggregate), memory utilization, telemetry throughput sustained, L/L_mqtt/L_twin at that scale, pod startup time, and scheduler placement latency.

Failure-recovery protocol: terminate one Live Twin pod (via kubectl delete pod or a chaos-engineering tool) and measure the full chain: failure, Kubernetes detection, rescheduling, container startup, MQTT reconnection with persistent-session redelivery, and state restoration from the last persisted document. Total wall-clock recovery time is reported, and separately, how much of it is Kubernetes-level (detection, rescheduling, startup) versus application-level (MQTT reconnect, state restore); that split is itself informative for the resilience research question, RQ6.

Resource-constrained behavior: a subset is repeated under explicit CPU/memory limits, not just requests, tightened below comfortable headroom, at both baseline and elevated telemetry rates.

Autoscaling, if implemented: the trigger condition used, time from trigger to new-pod-ready, stabilization time, and the resource overhead of the Horizontal Pod Autoscaler (HPA) controller itself are all reported, and autoscaling is explicitly not claimed to be beneficial without this measurement. A Live Twin is a stateful, one-per-machine service, so naive HPA, which assumes stateless, horizontally fungible replicas, may not even be the correct scaling primitive here; testing that assumption is itself a finding worth reporting either way.

[Pending deployment data: the full scaling curve and recovery-time breakdown.]

D. Single-Host Deployment Feasibility

Target configuration: an MQTT broker, Kubernetes (K3s is the natural choice for a single-host capstone deployment over full upstream Kubernetes, given its lighter resource footprint), the simulators, the Live Twins, the Process Twin, MongoDB, Analytics, and the Dashboard, all on one machine.

Recorded: host CPU, RAM, and disk utilization at idle and at the baseline five-station telemetry rate; the maximum stable telemetry rate before latency or loss degrades noticeably; and the maximum twin count the single host sustains before the same happens.

A concrete comparison point is already available from the related-work survey of Section III-D: Eclipse Ditto's documented minimum single-instance footprint is approximately 2 CPU cores and 4 GB of RAM, a useful lower bound against which to compare this work's own measured footprint, since Ditto is a mature, production-oriented reference performing a structurally similar job at a known resource cost.

[Pending deployment data: a resource-utilization table across a student workstation and a representative "small edge server" specification.]

E. Predictive-Maintenance Evaluation

Baselines are evaluated from weakest to strongest specifically so that a more complex model must earn its place over a simpler one rather than being adopted by default: threshold-based detection (flagging when the tool-wear internal variable, or its vibration proxy, crosses a fixed or rolling-percentile threshold); Isolation Forest (unsupervised anomaly detection on the multivariate sensor vector); Random Forest or XGBoost (supervised, using the simulator's known internal degradation state as ground truth for evaluation, available uniquely because this is a simulated system, even though it is hidden from the detector itself); and an LSTM or GRU model, adopted only if the first three demonstrably underperform on this specific task, so as to avoid adopting deep learning for its own sake.

Metrics: accuracy, precision, recall, F1, and ROC-AUC for classification; RMSE and MAE for any remaining-useful-life-style regression; and wall-clock training and inference cost for each, since a resource-constrained self-hosted deployment (Section XIII) should visibly weigh a model's compute cost, not just its accuracy.

As a reference point from the literature (Section III-F), a 2025 LSTM-based real-time tool-life predictor reported 33.17 μm RMSE in training and 119.36 μm RMSE in real-time deployment on a real milling dataset, useful for sanity-checking that results land in a plausible range, not as a target to beat.

[Pending deployment data: a full metrics table per baseline.]

F. Model-Fidelity Evaluation

Comparison methodology: for each variable where a real dataset exists (PHM 2010's vibration, force, and wear; NASA IMS's vibration), the calibrated simulator's generated trajectories are compared against the source dataset's held-out

portion on mean, variance, distribution shape (via a Kolmogorov–Smirnov statistic or Wasserstein distance), Pearson/Spearman correlation between paired variables, autocorrelation structure, and degradation-slope error (RMSE between the fitted trend and the held-out trend).

Explicit limitation, stated to avoid overclaiming from a distribution match: matching PHM-2010's vibration distribution and degradation slope demonstrates that the simulator's statistical behavior is consistent with one real dataset's observed behavior. It does not demonstrate that the simulator is physically accurate in the sense of correctly modeling the underlying mechanics of cutting-tool wear; a model can match observed statistics for the wrong causal reasons. This work treats "statistically consistent with PHM-2010" and "physically validated" as different, non-interchangeable claims, and makes only the former.

[Pending deployment data: fidelity metrics and a real-vs-simulated overlay figure.]

G. Baselines Matrix

TABLE XV. BASELINES MATRIX

Axis	Weaker Baseline	Stronger Baseline	This Work
Architecture	Monolithic simulator	Independent microservices (no orchestrator)	Kubernetes-orchestrated independent twins
Communication	REST polling	WebSocket	MQTT pub/sub
Machine model	Random/heuristic	Causal, uncalibrated	Causal, empirically calibrated (Sections VI–VII)
Deployment	Single-process	Containerized, non-orchestrated	Kubernetes deployment
Predictive maintenance	Threshold	Classical machine learning	LSTM only if justified (Section XI-E)

H. Reproducibility Checklist and Experimental Configuration

Reproducibility measures adopted: dataset versions and download dates (Section VII) pinned in a requirements/manifest file; simulator random seeds fixed and logged per experimental run; container image tags pinned rather than left at "latest"; the Kubernetes/K3s version recorded; the MQTT broker (Mosquitto or EMQX, with one selected and its exact version and configuration documented); experiment hardware specification recorded (CPU model, RAM, whether virtualized); telemetry rates and network-condition parameters for every Section XI-B run logged alongside results; and evaluation scripts committed to the repository alongside this paper, not kept only as ad hoc notebooks.

TABLE XVI. PLANNED EXPERIMENTAL CONFIGURATION

Experiment	Independent Variable	Dependent Metric(s)	Hardware	Repetitions
Baseline synchronization	Telemetry rate (fixed at design rate)	L, L_mqtt, L_twin, state freshness	Dev workstation	≥ 5 runs, report distribution
Network degradation	Latency / loss / burst / outage condition	Same as above, plus message-loss rate and recovery time	Dev workstation	≥ 5 runs per condition
Twin scaling	Twin count (1/5/10/25/50)	CPU%, RAM%, throughput, L at each step	Dev workstation or small cluster	≥ 3 runs per step
Failure recovery	Which twin pod is killed	Recovery time (total, plus Kubernetes-stage and application-stage split)	Same as scaling	≥ 5 runs
Single-host feasibility	Telemetry rate, twin count	Maximum stable rate/count before degradation	Single host, specification recorded	≥ 3 runs
Predictive-maintenance baseline comparison	Detector (threshold/Isolation Forest/Random Forest/XGBoost/LSTM)	Accuracy, precision, recall, F1, ROC-AUC, RMSE, and time	Dev workstation	Standard train/test split
Model fidelity	Simulator variable vs. held-out real data	Kolmogorov–Smirnov/Wasserstein statistic, correlation, autocorrelation, trend RMSE	N/A (offline analysis)	Single deterministic comparison

A corresponding performance-results table is intentionally left as a template rather than populated with placeholder numbers, since a placeholder number risks being mistaken for a real one on a skim; it will report, per the rows of Table XVI, the measured values once the deployment described above has been carried out. Likewise, the latency-under-load, twin-scalability, failure-recovery, and real-vs-simulated overlay figures referenced throughout this section are pending the same deployment and will accompany the completed evaluation.

XII. WHAT-IF / SCENARIO SIMULATION DESIGN

Each scenario in Table XVII is executed by a Scenario Worker (Section IV-A) forking Live Twin state at one instant and running a forward simulation without writing back.

TABLE XVII. WHAT-IF / SCENARIO SIMULATION DESIGN

Scenario	Mechanism	What Is Measured
"What happens if CNC-01 fails?"	Force CNC's forked state into FAULT; run the DAG forward	Time until Weld/Assembly starves for input; Packaging's output-rate drop, via production-token flow (Section IX-A)
"What happens if tool wear rises rapidly?"	Override the forked CNC twin's tool-wear internal variable with an accelerated slope	Time-to-threshold for a quality-flag drop; whether Inspection's defect probability crosses an operationally meaningful level before the next scheduled tool change
"What happens if telemetry becomes delayed?"	Inject artificial latency into the forked scenario's simulated message timestamps	State-freshness degradation as experienced by a downstream station's decisions — whether the delay is large enough to change an operational outcome
"What happens if inspection rejection increases?"	Override the forked Inspection twin's defect-probability parameter	Packaging throughput impact; whether bottleneck attribution (Section IX-B) correctly shifts to flag Inspection as the new constraint

Scenario	Mechanism	What Is Measured
"What happens if one station is unavailable?"	Generalization of the CNC-failure case to each station in turn	Confirms the DAG structure (Section IV-B) produces the physically correct downstream-blocking behavior for every edge

Scenario replay versus historical telemetry display, stated precisely: the dashboard's normal view displays what already happened, real telemetry history queried from persisted state. A Scenario Worker computes what would happen next from a hypothetical starting point, using the same causal models as the Live Twins but running forward under a modified assumption. The two are easy to conflate in a demonstration, since both eventually render as a chart, but they are architecturally and epistemically distinct: one is a query over recorded fact, the other is a forward simulation over a counterfactual.

[Pending deployment data: scenario execution time and state-divergence measurements once Scenario Workers are implemented and run.]

XIII. SME AND SELF-HOSTED DEPLOYMENT POSITIONING

A. The Honest Version of the Argument

The claim is not that self-hosting is cheaper than a managed cloud offering; that comparison requires a rigorous cost model, covering infrastructure, staff time, and the opportunity cost of do-it-yourself maintenance, which this work has not built and does not claim to have built. The defensible claim is narrower: **a self-hosted architecture removes mandatory dependence on a managed digital-twin service and its recurring usage-based fees, while shifting infrastructure, security, and maintenance responsibility onto the deploying organization.**

B. Advantages, Argued Specifically

- **No mandatory managed-service dependency.** This is not a hypothetical risk: this search surfaced a live illustration in AWS's own IoT portfolio (Section III-I) — Fleet Hub, IoT Analytics, and IoT Events have documented, dated end-of-life notices from AWS in 2025–2026, and independent sources suggest IoT TwinMaker itself may no longer be accepting new customers, unconfirmed by an official AWS notice but consistent with the pattern. An SME that had built its entire operational digital-twin workflow on a since-discontinued managed service faces a forced migration on the vendor's timeline, not its own. A self-hosted architecture cannot be discontinued out from under its operator this way; the corresponding cost, named honestly, is that nothing patches, monitors, or backs it up automatically either (Section XIII-C).
- **Data locality and reduced network dependence** for a facility with inconsistent or metered connectivity, since telemetry never needs to leave the local network for the twin layer to function.
- **Modular, vendor-independent deployment**, since the Kubernetes/MQTT/MongoDB stack uses no proprietary APIs, so migrating between cloud providers, or between cloud and on-premises, is a redeployment rather than a rewrite.

C. Disadvantages, Argued Specifically

- The organization now owns patching, upgrading, and securing a Kubernetes cluster and an MQTT broker: real, ongoing operational labor that a managed service otherwise absorbs.
- Hardware failure becomes the operator's problem to detect and fix, with no vendor service-level agreement.
- Scalability is bounded by owned or rented commodity hardware, not by a hyperscaler's effectively elastic capacity; the experiments of Sections XI-C–XI-D are explicitly a study of where that bound sits, not a claim that it does not exist.
- Enterprise integrations, such as ERP/MES connectors, compliance certifications, and vendor support contracts bundled with Siemens, PTC, or Dassault platforms, would need to be built or bought separately.
- **The support burden does not disappear, it moves.** A managed platform's support contract is replaced by whatever in-house or community expertise the SME can muster, which for a genuinely small manufacturer may be scarcer than the subscription fee itself.

D. The Nuance Current Market Data Adds

Two 2025–2026 developments complicate a simple self-hosted-versus-hyperscaler framing and are represented honestly rather than glossed over.

- **Siemens and Dassault are both now marketing subscription-based, cloud-hosted "democratized"/SMB-tier digital-twin offerings**, explicitly to reduce the upfront-cost barrier for smaller manufacturers. This addresses affordability without addressing vendor lock-in or data locality, a genuinely different point on the trade-off space than self-hosting, not evidence against self-hosting's value.
- **The Change2Twin position paper's "nine barriers" framing** [10] explicitly argues that organizational and integration-complexity barriers are at least as significant as raw cost for SME digital-twin adoption, meaning a technically excellent self-hosted architecture does not by itself solve the hardest part of SME adoption, which is often organizational readiness rather than infrastructure.
- **IndustryFusion** (Sections III-D, III-I) is the one open-source project found in this search whose stated mission is explicitly SME-targeted IIoT connectivity on self-hosted Kubernetes, meaning this positioning is not unprecedented, but it is also not crowded ground with a causal-manufacturing-simulation focus, which is where this work's actual technical contribution sits (Section XIV).

XIV. RESEARCH CONTRIBUTION ANALYSIS

A feature is not considered "novel" merely because it has not appeared in this exact combination before in the literature search conducted for this work.

A. Already Established (Cited, Not Claimed as Contribution)

- Digital twin architectures generally; MQTT as an IIoT transport; Kubernetes as a container-orchestration substrate; predictive maintenance via anomaly detection or remaining-useful-life prediction; and the existence of both

cloud-managed and self-hosted/open-source digital-twin platforms, as Table II alone establishes.

- Kubernetes-native digital-twin platforms specifically: KTWIN, IndustryFusion, and DT-Forge all predate and generalize beyond this work's scope.
- CNC tool-wear-to-digital-twin monitoring specifically: Ströbel et al. [9] and the broader body of work in Section III-F already establish this as an active area with real industrial deployments.

B. Engineering Contribution (Real, but Not a Research Claim)

- A concrete, working implementation: the Live Twin/Scenario Worker split, the DAG process twin with frozen-snapshot semantics, the seven-category twin schema (Section V-A), and the containerized deployment of all of it.
- A standardized internal data contract (the schema of Table IV) that is deliberately not a full standards implementation (Section V-B), a defensible engineering simplification rather than a research finding.
- The five-machine simulator implementation itself, including the Must/Useful/Omit variable discipline of Section VIII applied consistently across all five stations.

C. Candidate Genuine Research Contribution (Stated Carefully, Evidence Pending)

Per the restated gap of Section III-J, the strongest candidate is the combination, evaluated together and measured:

- Empirically calibrated, not random and not ML-trained-on-the-target-task, reduced-order machine models, calibrated against real PHM/NASA datasets with the calibration methodology of Section VI-C made explicit and the dataset-mismatch case (C-MAPSS, Section VII) called out rather than glossed over.
- A measured, not merely architecturally argued, evaluation of multi-twin synchronization under degraded network conditions (Sections XI-A–XI-B), for a self-hostable, commodity-scale deployment specifically (Section XI-D), comparatively under-populated ground relative to architecture-proposal papers, per the observation in Section III-G.
- Process-level coupling between independently deployed machine twins, evaluated via concrete, physically grounded what-if scenarios (Section XII) rather than asserted as a capability.
- The explicit twin-of-Kubernetes-versus-twin-on-Kubernetes framing (Section III-D) is not itself a research contribution; it is a positioning clarification. But stating it precisely is what makes the actual contribution above legible against the literature.

What this paper does not claim until the results of Section XI exist: that this architecture synchronizes better than any specific alternative, that it scales to any specific twin count, or that its causal models are more realistic than a named competing approach on any specific quantitative metric. Those are exactly the claims Section XI is designed to either support or fail to support.

XV. LIMITATIONS

Stated without hedging:

- **Simulated, not physical, machines.** Section II already treats this at length; conclusions about architecture, synchronization, and orchestration behavior transfer to a hardware-anchored deployment, while conclusions about "realism" transfer only as far as the fidelity metrics of Section XI-F can support, a statistical-consistency claim, not a physical-validation claim.
- **Reduced-order machine models** deliberately omit real subsystems (full servo dynamics, full thermal models, workholding mechanics) per the Omit classifications in Section VIII, a scope decision rather than an oversight, but one that means the models cannot answer questions about those omitted subsystems.
- **Five stations, one line, no rework loop.** Conclusions about DAG-based process modeling do not automatically generalize to a factory topology with cycles, parallel lines, or shared-resource contention between lines.
- **Dependence on public datasets not originally collected for this exact purpose.** PHM 2010's six-cutter sample and NASA IMS's three run-to-failure trajectories are both small-n by modern machine-learning standards; the calibration methodology of Section VI-C extracts shape rather than requiring large-n statistical power, but this bounds how strong a fidelity claim Section XI-F can ultimately support.
- **An explicit dataset-to-machine mismatch exists and is used anyway, for a stated purpose.** NASA C-MAPSS is retained only as a remaining-useful-life-methodology benchmark (Section VII), never as calibration evidence.
- **Simplified quality and production-flow modeling.** The production-token abstraction (Section IX-A) does not model buffer capacity, blocking or starvation between stations, or part-level uniqueness beyond what is needed for defect attribution.
- **Single-host and small-cluster experiments only.** The scalability conclusions of Sections XI-C–XI-D describe behavior up to whatever twin count and hardware class is actually tested; they do not extrapolate to industrial-scale deployments.
- **No enterprise integrations** (ERP, MES, or SCADA connectors) are built or evaluated.
- **Cybersecurity is out of scope.** The MQTT broker, Kubernetes RBAC, and API surfaces are not hardened or evaluated against any threat model.
- **No actual industrial PLC/SCADA integration** exists at this stage; the CNC command channel (Section X-B) terminates at a simulator, not real hardware.

What remains valid despite these limitations: the architectural pattern (Live Twin/Scenario Worker split, DAG process twin, seven-category schema, MQTT topic design), the calibration methodology independent of which specific dataset feeds it, and the experimental protocols of Section XI are all reusable and defensible regardless of the simulation-versus-hardware limitation, because none of them assume a physical machine on the other end of the telemetry stream; they assume a stream, full stop. **What cannot yet be generalized:** any claim about how this architecture performs against a real factory's telemetry patterns, real network conditions on a real shop floor, or real machines' actual, as opposed to dataset-calibrated, degradation behavior.

XVI. DISCUSSION

A. What Is Novel or Scientifically Valuable

Not any single technology choice: Kubernetes, MQTT, causal simulation, and public-dataset calibration are each independently established (Section XIV-A). What is novel is the

specific combination of empirically calibrated causal machine models, explicit machine/process/platform separation, and a self-hostable Kubernetes-native deployment, applied to a five-station discrete line and evaluated with a measured, not merely argued, synchronization and scalability protocol. This combination does not appear, as a whole, in any single platform or paper surfaced by a literature search spanning commercial platforms, open-source frameworks, the Kubernetes-digital-twin literature, and the calibration-dataset literature (Sections III and VII).

B. What Can Be Demonstrated Quantitatively, Once Section XI Is Run

Synchronization latency distributions (mean, P95, P99) at baseline and under controlled network degradation; a scaling curve across the specified twin-count progression with resource utilization at each point; a measured failure-recovery time, decomposed into Kubernetes-level and application-level stages; single-host resource feasibility at a stated telemetry rate and twin count; a predictive-maintenance baseline comparison across four algorithm classes with both accuracy and compute-cost metrics; and simulator-versus-real-dataset statistical fidelity on the two stations, CNC via PHM-2010 and NASA IMS, where real calibration data exists.

C. What Cannot Yet Be Supported

This should not be claimed even informally while Section XI's results remain pending: that this architecture is faster, cheaper, or more scalable than any named commercial or open-source alternative, since no comparative benchmark against another running system exists or is planned; that the simulated machines are physically validated, when they are only statistically consistent with specific public datasets on specific variables (Section XI-F); that the architecture has been demonstrated at any scale beyond whatever Section XI-C's actual test run reaches; that self-hosting is economically superior to a managed or SMB-cloud alternative, since no cost model has been built (Section XIII-A); and that the CNC command channel constitutes closed-loop industrial control (Section X-B) rather than an interactive, human- or scenario-triggered command path.

XVII. CONCLUSION AND FUTURE WORK

This paper has specified a self-hosted, Kubernetes-orchestrated digital-twin architecture for a five-station discrete manufacturing line: a Live Twin/Scenario Worker split with a DAG-based process twin under frozen-snapshot semantics; a seven-category machine/twin data model informed by, though not conformant to, ISO 23247; empirically calibrated causal simulators for each of five station types, with an explicit Must/Useful/Omit variable discipline and calibration sourced from public PHM, NASA, UCI, and Kaggle datasets; an MQTT-based communication architecture with a fully implemented bidirectional command path on the CNC station; and a complete experimental protocol for synchronization, scalability, resilience, single-host feasibility, predictive-maintenance baselining, and model-fidelity evaluation. The related-work analysis situates this contribution against both managed commercial platforms and the Kubernetes-digital-twin literature, with particular care to distinguish digital twins that run on Kubernetes from digital twins of Kubernetes itself, a distinction the existing literature does not always make explicit.

Quantitative claims about synchronization latency, scalability limits, and simulator fidelity are, at the time of writing, pending execution of the protocol in Section XI on a deployed cluster. In priority order, the future work that would most efficiently move this contribution from architecture-and-protocol to a fully evidenced result is as follows.

- **Run the baseline synchronization measurement of Section XI-A first.** It is the cheapest experiment, requiring no fault injection and no scaling, just the system running normally, and it immediately populates the first rows of the results table, which makes every later section less hypothetical.
- **Run the model-fidelity comparison for the CNC station (Section XI-F) in parallel**, independent of the cluster being under load; it is an offline statistical comparison between PHM-2010/NASA IMS and whatever the CNC simulator generates, and does not require Kubernetes at all.
- **Run the scaling and failure-recovery experiments of Section XI-C**, the most implementation-heavy item, requiring multiple simulated lines for the higher twin counts and a scripted pod-kill protocol; these are sequenced after the two items above.
- **Revisit the Abstract and the contribution claims of Sections I-F and XIV** once real numbers exist. Several hedges in this paper, "candidate contribution," "pending," are placeholders for exactly the sentence a real result allows one to write with confidence instead.

Beyond the immediate experimental program, natural extensions include generalizing the process DAG to allow rework loops and parallel lines (Section IV-B); replacing the production-token abstraction with a full discrete-event simulator if a throughput-optimization research question is later adopted (Section IX-A); extending the bidirectional command channel from the single CNC station to all five stations (Section X-B); and, most significantly for the digital-twin claim itself, hardware-anchoring the CNC command channel to one real machine, named in Section II-B as the smallest change that would move the strongest single station from a digital model to a digital twin in the strict sense.

REFERENCES

[1]E. Glaessgen and D. Stargel, "The digital twin paradigm for future NASA and U.S. Air Force vehicles," in Proc. 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conf., 2012.

[2]A. G. Wermann and J. A. Wickboldt, "KTWIN: A serverless Kubernetes-based digital twin platform," arXiv:2408.01635, 2024.

[3]IndustryFusion Foundation, "IndustryFusion / DigitalTwin," open-source repository, github.com/IndustryFusion/DigitalTwin.

[4]J. Robles, C. Martín, and M. Díaz, "OpenTwins: An open-source framework for the development of next-gen compositional digital twins," Computers in Industry, vol. 152, p. 104007, 2023.

[5]Open Factory Twin (OFacT) project, github.com/openfactorytwin/ofact; openfactorytwin.org.

[6]J. Zerwas, P. Krämer, R.-M. Ursu, N. Asadi, P. Rodgers, L. Wong, and W. Kellerer, "KAPETÁNIOS: Automated Kubernetes adaptation through a digital twin," in Proc. 13th Int. Conf. Network of the Future (NoF), 2022.

[7]L. Foschini, F. Poltronieri, C. Stefanelli, and M. Tortonesi, "Modeling digital twins of Kubernetes-based applications," University of Ferrara, IRIS repository, 2023.

[8]D. Borsatti, W. Cerroni, L. Foschini, G. Y. Grabarnik, L. Manca, F. Poltronieri, D. Scotece, L. Shwartz, C. Stefanelli, M. Tortonesi, and M. Zaccarini, "KubeTwin: A digital twin framework for Kubernetes deployments at scale," IEEE Trans. Netw. Service Manag., vol. 21, no. 4, pp. 3889–3903, 2024.

[9]R. Ströbel, A. Bott, A. Wortmann, and J. Fleischer, "Monitoring of tool and component wear for self-adaptive digital twins: A multi-stage approach through anomaly detection and wear cycle analysis," Machines, vol. 11, no. 11, p. 1032, 2023.

[10]Change2Twin Consortium (P. Pileggi, lead), "Overcoming nine digital twin barriers for manufacturing SMEs," position paper, TNO / Change2Twin project.

[11]L. Ragazzini, E. Negri, and M. Macchi, "Low-cost data integration framework for integration with simulation-as-a-service digital twins for make-to-order manufacturing SME," Politecnico di Milano, 2025.

[12]ISO 23247:2021, Automation Systems and Integration — Digital Twin Framework for Manufacturing, Parts 1–4, International Organization for Standardization, 2021.

[13]S. Matzka, "Explainable artificial intelligence for predictive maintenance applications," in Proc. 3rd Int. Conf. Artificial Intelligence for Industries (AI4I), 2020, pp. 69–74. [AI4I 2020 dataset: UCI Machine Learning Repository, DOI 10.24432/C5HS5C]

[14]PHM Society (Y.-C. Chen, organizer), "2010 PHM Society Conference Data Challenge" [Dataset], IEEE DataPort, 2010.

[15]University of Cincinnati Center for Intelligent Maintenance Systems / NASA Prognostics Center of Excellence, "IMS Bearing Data Set" [Dataset].

[16]Bosch GmbH, "Bosch Production Line Performance" [Dataset], Kaggle, 2016.

[17]A. Saxena and K. Goebel, "Turbofan Engine Degradation Simulation Data Set (C-MAPSS)" [Dataset], NASA Ames Prognostics Center of Excellence, 2008.