Journal of Manufacturing Systems 83 (2025) 800–821

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

journal homepage: www.elsevier.com/locate/jmansys

Technical paper

Self-diagnosis service to support analysis of production performance, monitoring and optimisation activities

José Joaquín Peralta Abadía [a,*], Fabio Marco Monetti [b], Sylvia Nathaly Rea Minango [b], Angela Carrera-Rivera [a,c], Miriam Ugarte Querejeta [a], Mikel Cuesta Zabaljauregui [a], Felix Larrinaga Barrenechea [a], Miren Illarramendi Rezabal [a], Antonio Maffei [b]

[a] Mondragon Goi Eskola Politeknikoa, 20500, Arrasate, Spain
[b] KTH Royal Institute of Technology, 114 28, Stockholm, Sweden
[c] Escuela Superior Politécnica del Litoral, Guayaquil, Ecuador

ARTICLE INFO

Keywords:
Self-diagnosis
Cyber–physical systems
Tool condition monitoring
Cloud computing
Predictive maintenance
Case study

ABSTRACT

Self-diagnosis functionalities, as integral components of advanced manufacturing services within cyber–physical systems (CPSs), are made possible through cloud computing technologies and machine learning techniques. These services play a crucial role in enhancing the autonomy of CPSs and introducing cost-efficient and scalable solutions. Despite the promising outlook, a gap exists in the literature regarding the lack of clear architectural frameworks and requirements for implementing self-diagnosis services in industrial settings. This paper addresses this gap by presenting a comprehensive requirement set and developing a high-level architecture tailored for self-diagnosis services. The proposed approach is validated through a detailed case study of a cloud-based self-diagnosis service, demonstrating alignment with the established architecture and requirements. The anticipated outcome of this research is to offer concrete implementation guidelines to support researchers, engineers, and practitioners in deploying CPS-based self-diagnosis services and improving production processes and system performance.

1. Introduction

In the modern industrial landscape, cyber–physical systems (CPSs) are a pivotal technological framework that integrates physical assets with advanced computing and communication technologies. This integration enables enterprises to increase their automation levels, improve efficiency, and adapt to sudden changes [1–3]. Despite the potential they show to reshape industrial practices, integrating CPSs into existing manufacturing environments remains challenging and calls for additional guidance [4].

To support scalable and flexible deployments of CPSs, the concept of *cloud manufacturing* has gained traction [5]. By embedding CPSs within cloud computing environments, cloud manufacturing enables on-demand access to configurable computing resources – networks, servers, storage, applications, and services – with minimal management effort [5–7]. This approach is particularly promising for small and medium-sized enterprises (SMEs), which can benefit from improved responsiveness, reduced infrastructure overhead, and access to advanced analytics.

As CPSs and cloud-based infrastructures mature, increasing attention is paid to system autonomy. Inspired by autonomic computing [8], *self-X* behaviours have emerged as a pathway towards higher autonomy in manufacturing systems. These behaviours enable systems to adapt to unexpected variations or deviations by making cost-effective and timely decisions [9]. Among them, self-diagnosis has received considerable attention due to its potential to detect faults, support predictive maintenance, and minimise production downtime [10].

Self-diagnosis services represent advanced manufacturing capabilities that leverage cloud computing and machine learning (ML) techniques to enhance the autonomy of CPSs [11]. The significance of these services lies not only in improving autonomy, but also in introducing cost-efficient and scalable solutions, readily deployable in SME contexts [12]. This can serve to enable efficient fault detection and recovery [13]. Consequently, self-diagnosis services are envisioned to play a pivotal role in continuously improving production processes [14].

However, despite the promising outlook, a gap remains in the literature. While previous research has identified high-level requirements for self-diagnosis functionalities, there is a lack of formalised guidelines for self-diagnosis services implementation in industrial contexts. Additionally, the architecture underlying such services is not yet properly or

* Corresponding author.
E-mail address: jjperalta@mondragon.edu (J.J. Peralta Abadía).

https://doi.org/10.1016/j.jmsy.2025.11.010
Received 3 June 2025; Received in revised form 6 November 2025; Accepted 8 November 2025
Available online 11 November 2025

fully defined. Although some contributions exist [15,16], a thorough description of the underlying architecture for self-diagnosis services is lacking.

This work presents a comprehensive set of requirements and develops a layered reference architecture as a guideline for implementing self-diagnosis services in CPS environments. To support its practical application, a case study protocol is proposed that enables practitioners to map and analyse the level of compliance of their systems against the guideline. The protocol presents expected outcomes to indicate the degree of self-diagnosis readiness. Thereby, both a diagnostic view of current capabilities and a basis for identifying opportunities for improvement are provided. The protocol is generalisable and can be applied to a variety of CPS contexts to consistently measure alignment with the proposed framework.

The paper illustrates the application of this protocol through a case study of the OptiTwin project [17], a cloud-based tool condition monitoring (TCM) system funded by the Provincial Council of Gipuzkoa, Spain. OptiTwin integrates ML and deep learning (DL) for tool-wear prediction services. It serves as a relevant industrial example of how compliance with the guideline can be assessed in practice.

The paper is structured as follows. Section 2 describes the background concepts of *self-diagnosis*, as well as related work. Section 3 presents the methodology, detailing the steps followed to formally define the architecture and requirements for self-diagnosis services, as well as the steps to map the architecture and requirements to the case study. Section 4 presents the resulting definition of the architecture and requirements of the self-diagnosis service. Section 5 presents the case study and its results, validating the proposed self-diagnosis architecture and requirements. Subsequently, Section 6 discusses the results. Finally, Section 7 presents conclusions and future work.

2. Background

Industries are increasingly driven to adopt sustainable manufacturing practices that enhance efficiency, responsiveness, and long-term competitiveness, as underscored by the Industry 4.0 initiative [18]. This mandate represents a commitment to environmental management and provides opportunities for enhanced growth and profitability through resource optimisation. By leveraging fourth-industrial-revolution technologies jointly with sustainable design principles across the product lifecycle – from conception to production and servicing – industries can expand their market reach and embrace novel business models [19]. Notably, one of the foremost sustainability prospects within the Industry 4.0 framework lies in the realm of adaptable production systems [20].

The concepts of adaptability and flexibility in manufacturing have received significant attention since the beginning of mass customisation paradigms, e.g., holonic manufacturing systems [21], reconfigurable manufacturing systems [22], and evolvable production systems (EPS) [23]. The advent of CPSs embodied these paradigms and realised the Industry 4.0 transition, facilitating real-time monitoring, data-driven analysis, and decentralised decision-making in manufacturing contexts.

Achieving autonomy within production systems is essential for the ongoing transformation of the manufacturing landscape. This autonomy paves the way for innovative business models, such as equipment leasing and Platform-as-a-Service (PaaS) arrangements. These models align perfectly with the principles of a circular economy, promoting resource efficiency and sustainability. While aiming for automation, seamless integration through open standards must be achieved, which represents a challenge for existing, often heterogeneous and proprietary, legacy systems in industrial environments

Furthermore, one of the foremost challenges for automation is the effective exploitation of large volumes of industrial data for automated processing and model development [24]. Traditional diagnostic approaches often face significant limitations in these environments, as they tend to lack flexibility to noisy data and require large, annotated datasets that are both expensive and difficult to acquire in industrial settings [25].

Attempts have been made to define fundamental requisites of autonomous production systems to catalyse the transition towards autonomous production systems [21,23]. This includes modular architectures and *self-X* behaviours, e.g., self-configuration, self-organisation, self-diagnosis, self-learning, that may emerge in the pursuit of autonomy.

2.1. Self-X behaviours

A hierarchical framework of *self-X* properties was provided by [9] within the context of autonomic computing. The framework proposed three levels of *self-X* behaviours – primitive, major, and general – distinguished by the level of abstraction inherent to each category. For instance, self-adaptation, which is at the general level, may develop from processes like self-configuration or self-optimisation, at the major level, which in turn depends upon self-awareness, at the primitive level.

The primitive level is the foundation and comprises self-monitoring, self-awareness, and self-control. *Self-monitoring* entails the system's ability to track its own state, typically facilitated by sensor-derived data [26]. This foundational process often engenders more subtle *self-X* behaviours, such as self-diagnosis and self-organisation. *Self-awareness*, also referred to as self-knowledge, self-description, or self-explaining, pertains to the system's awareness of its internal characteristics and the surrounding environment, enabling adaptive responses [26,27]. Finally, *self-control* confers autonomous regulatory capabilities on the system and its constituent components [28].

Building upon the primitive level, the major level encompasses functionalities like self-configuration, self-optimisation, self-protection, self-diagnosis, and self-prediction. *Self-configuration*, sometimes referred to as self-tuning, allows for autonomous configuration and adjustment of system components [8,26]. In manufacturing contexts, this manifests as modules capable of operation without explicit programming [29]. *Self-optimisation* involves modifying dynamic behaviour to improve efficiency and resource utilisation [8,12,28]. *Self-protection* addresses the resilience of systems to adversarial incursions [8,26]. *Self-diagnosis*, also referred to as self-testing or self-inspection, enables automatic fault detection and understanding [26]. Finally, *self-prediction* (or self-compare) entails predictive capabilities based on historical data [12].

The general level is the most abstract and encompasses the highest-order self-X properties. These include self-management, self-healing, self-evolution, and self-adaptation. *Self-management* encompasses a spectrum of self-properties, including self-configuration and self-optimisation [8,26,28]. It constitutes a cornerstone of autonomic computing, addressing both system-level and component-level autonomy. *Self-healing*, also referred to as self-repairing or self-maintenance, denotes the system's capacity to detect and rectify anomalies by itself [8] or with minimal assistance [28]. *Self-evolution* allows the system to adapt to dynamic environments, akin to natural evolutionary processes [29]. Finally, *self-adaptation* involves awareness-driven behavioural adjustments aimed at maintaining desired conditions [26].

However, to incorporate a broader spectrum of *self-X* behaviours, the work of [30] introduced a new level and an additional commonality to all framework levels. The new level, organisational, comprises self-organisation and self-reconfiguration and is situated underneath the general level, since it emerges directly from the self-management process [31]. At this level, *self-organisation* entails structural adaptation in response to perturbations or unforeseen circumstances, often yielding emergent behaviours [27,28]. *Self-reconfiguration*, in contrast, denotes the modularity-driven adaptability of hardware and software components [26,29].

As for the addition common to all levels, *self-learning*, it encapsulates the cognitive dimensions of system functionality, representing the gradual evolution from rudimentary to comprehensive abstraction;

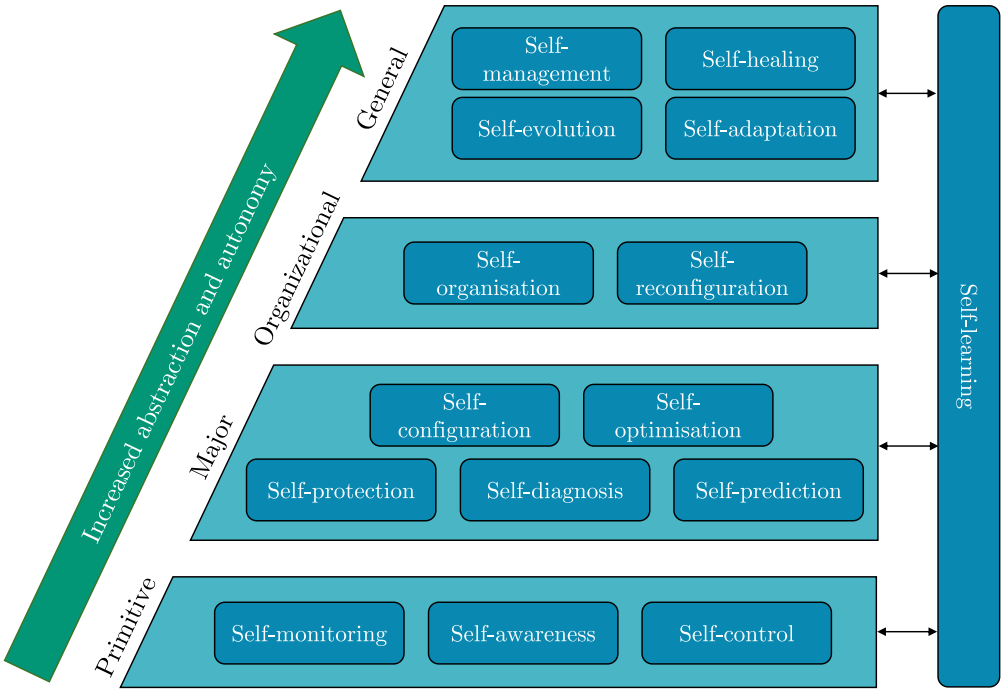

Fig. 1. Hierarchical levels of autonomy of *self-X* behaviours based on [9,30].

systems are expected to learn to self-adapt, evolve, predict, and self-organise [23]. Systems glean insights from past experiences to inform future actions, often leveraging ML techniques [32]. These learning processes augment adaptive behaviours, facilitating functionalities such as self-adaptation and self-organisation. Based on the proposals presented in [9,30], the hierarchical levels and their *self-X* behaviours are depicted in Fig. 1.

Foundational behaviours (e.g., self-healing, self-adaptation) emanate from less complex *self-X* behaviours (e.g., self-monitoring, self-awareness) discernible at the highest level of abstraction (general level). Consequently, more tangible and fundamental behaviours assume paramount significance, as their collective efficacy underpins the emergence of generalised *self-X* behaviours and, by extension, heightened levels of autonomy. For instance, self-awareness, self-monitoring, self-control, self-diagnosis, and self-organisation collectively contribute to realising self-evolvable or self-adaptive processes.

2.2. Self-diagnosis

Self-diagnosis refers to the ability to detect and identify conditions within a system by analysing its own data [10]. In manufacturing, it enables systems to autonomously diagnose failures and their causes [21,29,33]. Implementing self-diagnosis leads to reduced accidental maintenance, more efficient processes, improved product quality, prevention of operational failures, higher production capacity, and enhanced decision support [34].

To achieve self-diagnosis behaviours, [35] identified specific requirements: connectivity, learning and predictability, self-explanatory results, evolvability, integration, and comprehensive fault identification. These requirements were evaluated through quality function deployment (QFD), and ranked in order of importance [15]: comprehensive fault identification, integration, and learning were assessed as the top three factors.

Various technical approaches have been used to realise self-diagnosis capabilities. These include reasoning models, fuzzy logic, neural networks, symptom- and functional-based reasoning [36], DL algorithms [37], and statistical methods combined with supervised and semi-supervised ML approaches for fault diagnosis [38,39]. Other conventional methods have included agent-based failure identification from sensor data [40], a problem-oriented multi-agent system for collaborative maintenance [41], internet-based diagnosis [42], and a hidden Markov model with principal component analysis (PCA) for defect diagnosis [43]. Several research studies have also addressed this subject, such as the multi-agent-based diagnostic data acquisition and management in complex systems (MAGIC) approach [44] and I-RAMP3 (intelligent network devices for fast ramp up) project [45], besides the self-diagnostic methods proposed for EPS [46] and evolvable assembly systems (EAS) [47].

These studies provide valuable insights into the potential role of self-diagnosis in manufacturing systems to optimise operational efficiency and maintenance procedures. However, significant challenges remain in achieving self-diagnostic capabilities. For instance, capturing domain knowledge, controlling error propagation, and efficient modelling for data processing, planning and execution [35]. To address these challenges, monitoring systems require data and architectural innovations to detect, analyse, and fix errors autonomously.

This requires integrating heterogeneous data – e.g., from sensors, machine logs, and control systems – into a multi-layered architecture that supports data abstraction, context awareness, data-driven modelling, and cross-layer communication. This would enable the alignment of low-level signal processing with knowledge extraction, thus enhancing real-time diagnosis and adaptability.

Although the technological groundwork for self-diagnosis is advancing, a gap remains between identifying high-level requirements and converting them into practical implementation guidelines. Recent work has attempted to formalise *self-X* behaviours using the MAPE-K framework [16,48], and a mapping with RAMI 4.0 [49], outlining the technologies and standards needed to deploy smart manufacturing. However, the authors also highlight that a comprehensive, generic guideline is lacking.

Moreover, while prior studies have proposed partial architectures for *self-X* behaviours [13,50–52], a detailed and generic architecture specifically tailored for self-diagnosis services has not yet been realised. Implementing such services requires specifying inputs, outputs, resources, and information flow in a formalised and reusable structure.

Standards such as ISO 23247, which define a general framework for digital twins in manufacturing, provide a useful reference point for

situating the proposed architecture within ongoing international standardisation efforts [53]. While our focus is on self-diagnosis services, the modular and layered design principles adopted here are compatible with such frameworks, as shown in other works [54].

2.3. Large language models

Recent research has highlighted the potential of large language models (LLMs) as advanced enablers for self-diagnosis services. Their strength lies in their ability to interpret and reason over unstructured and noisy data, integrate information from heterogeneous sources, and deliver context-aware insights [55]. These capabilities position LLMs as key enablers in advancing self-diagnosis systems towards what has been described as the fault diagnosis 2.0 (FD 2.0) era, where intelligent, perceptual learning allows the models to respond in alignment with user intent [24].

LLMs combine internal knowledge repositories with reasoning capabilities, making them suitable candidates for self-diagnosis systems. They can interpret data streams autonomously, identify anomalies, and generate control strategies without requiring task-specific training, commonly referred to as zero-shot learning [56]. Their generalisation capacity and scalability grant them a distinct edge over traditional, narrower AI models [57].

Beyond language-based tasks, such as comprehension, summarisation, and conversational dialogue, LLMs can be integrated into systems with multi-modal inputs, including textual and visual data [55]. In industrial settings, this enables them to break down and manage complex diagnostic tasks, such as time series preprocessing, model selection, result interpretation, and root cause identification. Acting as autonomous co-pilots, they can not only execute diagnostic routines but also deliver actionable recommendations. By adapting to user feedback, they promote continuous learning and refinement of the system [24].

Despite their strengths, LLMs are susceptible to producing erroneous or inconsistent outputs — an issue often referred to as "hallucination". This poses risks in safety-critical industrial environments [58]. To address this, careful prompt engineering is essential to direct the reasoning of the model and to align outputs with intended outcomes [24]. Robustness can be further improved through agentic frameworks that introduce validation and refinement loops. For example, a validator agent can test proposed actions against a digital twin, while a re-prompter agent can iteratively reformulate prompts in response to feedback [58]. Together, these mechanisms enable self-diagnosis systems to respond to performance degradation or unexpected disturbances without requiring direct human supervision.

3. Methodology

Building on the gaps identified in the previous section, the present work proposes a practical guideline for self-diagnosis. The methodology employed in this study aimed to ensure the effectiveness and practical applicability of self-diagnosis systems in manufacturing settings.

To define the guideline for the implementation of self-diagnosis in production, the methodology followed in this study comprised four sequential steps: (i) architecture design, (ii) requirement analysis for self-diagnosis, (iii) tiered classification of architectural components and requirements, and (iv) case study protocol for guideline evaluation. The methodology steps are depicted in Fig. 2.

This systematic process helped devise and prioritise a robust architectural framework and the essential criteria for self-diagnosis systems. It also enabled the formulation of a general case study protocol for assessing compliance. Collectively, these steps support the integration and evaluation of self-diagnosis services in autonomous manufacturing systems.

3.1. Architecture design

Establishing a robust architecture that can accommodate the evolving nature of technology and industry demands is crucial for implementing self-diagnosis services in industrial settings. The architecture design phase focused on supporting autonomous decision-making capabilities within CPSs. This focus on autonomy was balanced with the need for the architecture to be scalable and adaptable to function effectively under diverse system conditions.

The first step to define an architectural pattern and structure involved an in-depth analysis of existing architectures and frameworks related to self-configuration and self-diagnosis [13,50–52]. Through this analysis, a layered architecture emerged as the most suitable pattern.

The next step involved identifying the components, creating their specifications, and allocating them in the respective layers of the architecture. This included asset administration shell (AAS) components, which facilitate information exchange among Industry 4.0 assets and legacy systems by adhering to RAMI 4.0 principles [59]. Furthermore, the architecture was aligned with the MAPE-K framework and RAMI 4.0 architectures outlined in [16]. The last step focused on defining the data and control flow mechanisms within the architecture to support self-diagnosis services effectively.

3.2. Requirement analysis

Building upon the architectural framework developed in the previous phase, the requirements analysis stage delved into the specific needs and objectives of self-diagnosis behaviour. By breaking down high-level requirements into detailed functional sub-requirements, this phase aimed to provide a comprehensive guide for implementing self-diagnosis services in autonomous manufacturing systems.

Previous work [30] provided a shortened list of requirements for *self-X* services, based on a combination of existing literature and study of the state-of-the-art in CPS and its adoption in industry. Identification and prioritisation of requirements were based on a thorough QFD analysis and were enhanced by first-hand experience of researchers and practitioners. The present work revisited and expanded on the previously identified requirements, thoroughly examining and extending the list to ensure its comprehensiveness.

This stage involved three steps. The first step identified high-level requirements for self-diagnosis services based on the architecture proposed in this work and the list of requirements presented in [30], encompassing the objectives of self-diagnosis functionalities within CPSs. To make it as complete as possible and enhance the usability, the second step focused on breaking the high-level requirements into more specific functional sub-requirements that would make the list more interpretable. Finally, the last step mapped the requirements with the architecture, ensuring that the identified requirements support the key components and functionalities of the architecture.

3.3. Tiered classification

The architecture components and sub-requirements were assigned to one of three tiers: Core, Secondary, and Advanced. This tiered classification was established to ensure fair and interpretable evaluation across systems of varying technological maturity. This tiered structure enables both a global and tier-specific analysis. It provides a more nuanced assessment of the alignment of self-diagnosis CPSs with the proposed framework.

The *Core* tier contains the essential capabilities required for baseline self-diagnosis. The *Secondary* tier includes enhancements that improve integration, adaptability, and performance. Finally, the *Advanced* tier comprises optional and experimental features, such as LLM-enabled modules, that can augment functionality but are not essential for compliance.

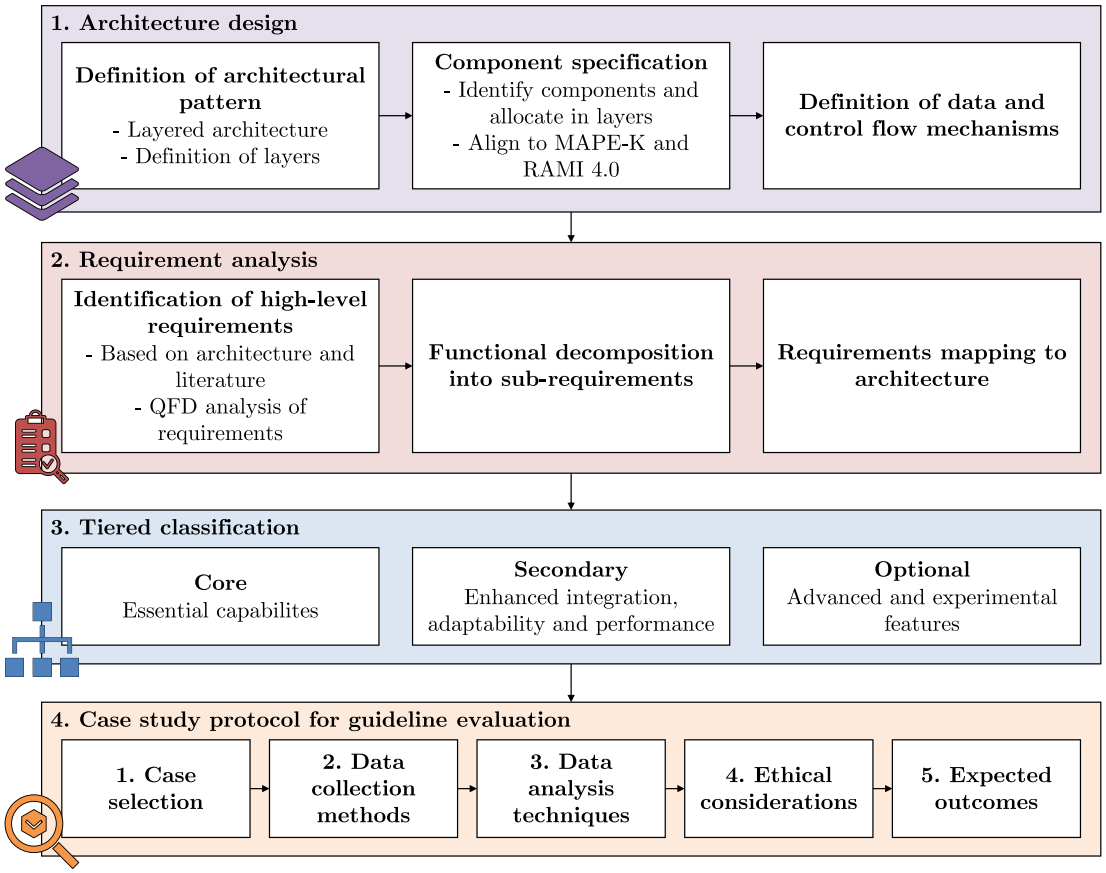

Fig. 2. Four-step methodology for the creation of self-diagnosis service architecture.

3.4. Case study protocol for guideline evaluation

A case study protocol was developed to assess the compliance and readiness level of a self-diagnosis CPS with the proposed architecture and requirements. The assessment is done by mapping and analysing the components and functionality of an existing system against the guideline. It follows the case study protocol structure defined in [60], which outlines the specific methods and procedures under which the case study is to be conducted.

Furthermore, the case study approach provides both a structured evaluation method and a transferable tool that can be applied across several industrial contexts. The case study protocol consists of five parts, defined in [60]: (i) case selection, (ii) data collection methods, (iii) data analysis techniques, (iv) ethical considerations, and (v) expected outcomes.

3.4.1. Case selection

The case selection phase is a key stage in preparing a case study for self-diagnosis evaluation. It precedes and shapes the subsequent protocol stages. The first step is to clearly define the system under study. In this context, the case must be a contemporary industrial CPS with some level of self-diagnostic capability.

It is also important to establish the boundaries of analysis. This involves distinguishing the self-diagnosis features to be evaluated from the broader operational context of the CPS. The scope should specify which technologies, organisational settings, and time frames are included in the study.

The choice and definition of the case must align directly with the objectives of the evaluation protocol. The goal is to determine how and to what extent the system complies with the proposed reference architecture and requirements. Clarity in case definition ensures that the mapping analysis is targeted, reproducible, and comparable.

Candidate cases may include industrial platforms, pilot demonstrators, or deployed systems in manufacturing environments. A key criterion is that the case provides sufficient access to documentation, stakeholders, or operational data.

Once selected, the case study protocol becomes the guide for data collection and analysis. It ensures that the evaluation remains focused on the defined system boundaries. This contributes to the reliability and validity of the compliance assessment. It also provides a foundation for comparing results across different CPS implementations.

3.4.2. Data collection methods

The data collection phase ensures that evidence about the system under study is gathered in a systematic and ethical way. Unlike controlled experiments or surveys, case studies rely on real-world settings where researchers and practitioners have limited control. This makes advance planning critical.

Two main objectives guide this phase. The first is to identify and gain access to relevant data sources. These may include technical documentation, system logs, stakeholder interviews, or direct observations of system operation. The second is to ensure that all data is collected ethically, with appropriate consent and safeguards for confidentiality.

Practical arrangements must be defined in advance. This includes securing access permissions, presenting credentials to system owners, and arranging logistics such as observation schedules, workspace availability, or data-sharing procedures. Clear communication with stakeholders is essential to minimise disruption and ensure that fieldwork proceeds smoothly. It is also important to prepare for contingencies. Case study environments may present unexpected constraints, such as limited access to facilities or changes in production schedules.

Handling of collected materials requires special attention. Whenever possible, electronic versions of documents and logs should be requested

to avoid the burden of carrying or copying physical materials. Secure storage practices must be followed to protect confidential data.

Finally, ethical safeguards are integral to this stage. All participants must be informed of the purpose of the study and give their consent before any observation or data collection takes place. No personal or sensitive information should be recorded unless explicitly approved, and procedures should be in line with institutional or organisational guidelines.

3.4.3. Data analysis techniques

The data analysis phase translates the collected evidence into insights about the system's level of compliance with the proposed guideline. In case study research, this step is not strictly automated or rigid. Instead, it requires systematically examining, categorising, and testing both qualitative and quantitative evidence. The overall aim is to link real-world observations to the components of the reference architecture and requirements.

For self-diagnosis evaluation, mapping analysis serves as the primary technique. System components and functionalities, identified through document analysis and observation, are mapped against the proposed architecture and the tiered set of requirements. This mapping allows practitioners to detect patterns of alignment and divergence, building a structured picture of system readiness.

The mapping process is guided by three evaluation criteria: (i) the presence and role of each system component defined in the proposed architecture; (ii) the flow of data and control mechanisms across architectural layers, and (iii) the extent to which each self-diagnosis requirement, and its sub-requirements, is fulfilled.

To quantify the evaluation criteria, several metrics are proposed in this study, including architectural mapping completeness, requirement fulfilment score, gap severity score, and coverage ratio. The *architectural mapping completeness* and *coverage ratio* metrics are reported both globally and separately for each tier, enabling a more nuanced interpretation of system readiness. The metrics are defined as follows:

- **Architectural mapping completeness ($\mathcal{A}$)**: This metric reflects the percentage of alignment between theory (proposed architecture) and practice. It is weighted by the number of components in each architectural layer. It ensures that layers with more components contribute proportionally more to the overall completeness score. It is defined as follows:

$$\mathcal{A} = \frac{\sum_{i=1}^{n} \mathcal{W}(i)}{\sum_{i=1}^{n} w_i} \tag{1}$$

  where:

  - n is the total number of architectural layers (e.g., asset, edge, fog, cloud).
  - w_i is the total number of theoretical components in layer i, i.e., the layer weight.
  - c_i is the fraction of components in layer i that are implemented, i.e., layer mapping completeness given by $\frac{\text{Mapped components in layer } i}{\text{Total components in layer } i} \times 100$.
  - $\mathcal{W}(i)$ is the weighted completeness in layer i. It is calculated as $w_i \times c_i$.

- **Tier-specific $\mathcal{A}$**: Calculated as the percentage of components that are at least substantially met within a given tier and all its preceding tiers. For example, the Secondary tier includes both Core and Secondary sub-requirements. It uses (1) to calculate the score. This metric enables separate reporting of baseline compliance and advanced capability adoption.
- **Requirement fulfilment score**: Provides a quantitative measure of how well the sub-requirements of a requirement are implemented, rated on a five-point scale:

  1. Not met: No evidence or not applicable.
  2. Minimally met: Minimal or ad-hoc support.
  3. Partially met: Clear but incomplete support.
  4. Substantially met: Most aspects covered, minor gaps.
  5. Fully met: Requirement entirely realised.

- **Gap severity score**: Prioritises which gaps (sub-requirements with fulfilment score ≤ 3) should drive new or refined requirements in the next development iteration, rated on a three-point scale:

  1. Low: Optional requirement with minimal to no impact on core functionality.
  2. Medium: Important for robustness or usability.
  3. High: Essential for reliable self-diagnosis.

- **Functional ratio ($\mathcal{F}$)**: Percentage of requirement satisfaction, focusing on the proportion at least substantially met (score ≥ 4). It is defined as follows:

$$\mathcal{F}_i = \frac{|\ r_i^*\ |}{|\ r_i\ |} \times 100 \tag{2}$$

  where:

  - r_i is the total number of sub-requirements of requirement i.
  - r_i^* is the number of sub-requirements with mean score ≥ 4 of requirement i.

- **Tier-specific $\mathcal{F}$**: Calculated as the percentage of sub-requirements in each tier that are at least substantially met. It uses (2) to calculate the score. This metric enables separate reporting of baseline compliance and advanced capability adoption.

3.4.4. Ethical considerations

The ethical considerations stage focuses on protecting the rights and well-being of all individuals involved in the case study. Participants must be informed of the purpose of the evaluation and provide their consent before any data is collected. No observation or documentation should be carried out without explicit approval.

A key priority is to prevent harm. This means ensuring that participation does not disrupt normal operations or create risks for workers, managers, or other stakeholders. Privacy and confidentiality must also be safeguarded. Sensitive system information, such as proprietary data, technical reports, or operational logs, should be handled securely and only shared with authorised parties.

Formal approval from the relevant ethics or institutional review board should be obtained before fieldwork begins. The evaluation team must commit to following established ethical standards and demonstrate care and sensitivity throughout the study.

3.4.5. Expected outcomes

Case studies are expected to bring insights into three key aspects:

1. **Coverage**: Reflects whether a system has implemented the minimum set of components to comply with each tier. It is determined by the *architectural mapping completeness* score. The tiers are defined as follows:

   - A system has **Core coverage** if at least 85% of Core components are present.
   - It has **Secondary coverage** if Core coverage is achieved and at least 75% of Secondary components are present.
   - It has **Comprehensive coverage** if Core coverage is achieved, Secondary coverage is achieved, and at least 50% of Advanced components are present.

2. **Functionality**: Indicates how well the implemented requirements function within each tier. Functionality is measured by the *requirement fulfilment scores* and the *functional ratio*. The tiers are defined as:

Table 1
Readiness matrix combining coverage and functionality tiers.

Coverage →/ Functionality ↓	Below Core coverage (<85% Core)	Core coverage (≥85% Core)	Secondary coverage (Core + ≥75% Secondary)	Comprehensive coverage (Core + Secondary + ≥50% Advanced)
Below Core functional (< 75% Core)	**Not ready**: lacks minimum self-diagnosis capabilities.	**Weak baseline**: Core structure present but functionality lacks.	**Weak extension**: broader scope attempted without sufficient functionality.	**Weak innovator**: advanced innovation undermined by low functionality.
Core functional (≥75% Core)	**Emerging baseline**: Core functions are effective but coverage remains limited.	**Baseline ready**: solid Core maturity, suitable as a reliable foundation.	**Enhanced ready**: functional Core combined with a broader scope of components.	**Advanced ready**: strong Core with early adoption of advanced components.
Secondary functional (Core + ≥75% Secondary)	**Unbalanced start**: higher-tier functionality attempted without Core coverage.	**Unbalanced growth**: Secondary functions are effective, but Core coverage is limited.	**Balanced expansion**: robust Core and Secondary tiers aligned in both scope and functionality.	**Extended expansion**: Secondary functionality complemented by broader advanced adoption.
Comprehensive functional (Core + Secondary + ≥50% Advanced)	**Premature exploration**: advanced functionality targeted without foundational coverage.	**Premature innovation**: advanced functionality achieved with only Core coverage.	**Emerging innovator**: broader scope coverage present with advanced functionality.	**Comprehensive innovator**: full maturity achieved across all tiers.

- A system is **Core functional** if at least 75% of Core requirements are substantially or fully met (mean score ≥4).
- The system is **Secondary functional** if it is Core functional and at least 50% of Secondary requirements are substantially or fully met.
- It is **Comprehensive functional** if it is Core and Secondary functional, and at least 50% of Advanced requirements are substantially or fully met.

3. **Improvement potential**: Identifies gaps that can inform refinements, optimisations, or the adoption of advanced capabilities. Gaps are identified through low fulfilment scores and prioritised using the gap severity metric. Importantly, improvement potential is relevant for systems at all levels of maturity: even high-scoring use cases can benefit from identifying opportunities for continuous improvement. Practitioners may then decide which opportunities to pursue based on resources, operational priorities, and cost–benefit considerations.

The combination of coverage and functionality offers a dual perspective on readiness. Coverage indicates how broadly the system aligns with the guideline across architectural layers. Meanwhile, functionality assesses how well the requirements are implemented. Together, they form a readiness matrix that practitioners can use to position their systems and interpret maturity profiles. The readiness matrix is presented in Table 1.

4. Reference architecture and requirements for self-diagnosis services

The results of this work provide implementation instructions to enable interested practitioners to apply *self-diagnosis* services to CPS. A formal architectural representation was provided to visualise and schematise how services would fare in the context of complete, complex CPSs. The resulting novel reference architecture and requirements for implementing self-diagnosis services are presented.

4.1. Reference architecture for self-diagnosis services

Before listing the identified requirements, it was necessary to frame the novel architecture for self-diagnosis services. The proposed architecture is depicted in Fig. 3.

The proposed architecture is structured into four interconnected layers: asset, edge, fog, and cloud. Each has a specific role in supporting self-diagnosis and data-driven decision-making within a cyber–physical system (CPS).

The **asset layer** is responsible for data acquisition from various, heterogeneous sources, and comprises up to n physical assets, each managed by human operators. Each asset consists of a machine integrated with sensors and a controller, which is responsible for acquiring process and status data in real time. Assets are the fundamental interface between the physical and digital worlds. They enable data collection and actuation based on feedback from higher layers.

To support semantic interoperability and standardised data access per RAMI 4.0, each asset should be associated with an AAS, which acts as its digital twin. The AAS provides a structured, machine-interpretable description of the properties, capabilities, and available services of the asset. It facilitates integration with higher-layer diagnostics and control applications.

Machines, transportation units, and other shop-floor equipment in this layer are monitored continuously. In this context, operators play an active role in responding to system outputs. To evaluate mapping completeness, this layer is decomposed into four core elements: AASs, machines, sensors, and controllers.

The **edge layer** is deployed on-site, close to the assets. It typically includes a local server responsible for preliminary data processing, filtering, and simple analysis. Furthermore, the edge layer provides the runtime environment for AAS instances. The instances support the digital representation of physical assets to be maintained and queried locally. This supports standardised interfaces for accessing asset data, enhances interoperability, and facilitates integration with the fog and cloud layers.

This layer may also host lightweight services, such as local databases or file storage systems, to support raw data buffering and preprocessing at predefined sampling intervals. By handling early-stage data, the edge layer reduces latency, minimises bandwidth usage, and ensures that only relevant data is forwarded for more complex processing.

In scenarios involving highly sensitive enterprise data, the edge layer can execute selected ML or DL models locally to preserve data privacy and comply with security policies, avoiding the need to transmit confidential information to the cloud. In the mapping completeness evaluation, this layer comprises four elements: data processing and analysis, local storage, AAS runtime, and ML/DL inference.

The **fog layer** provides a higher level of operational intelligence, focusing on operations management. It synthesises shop-floor data with enterprise-level systems, such as enterprise resource planning (ERP), customer relationship management (CRM), materials requirement planning (MRP), and supply chain management (SCM). These integrations allow the system to contextualise shop-floor data with broader production, inventory, and logistics information.

While the fog layer can enrich real-time sensor streams with production schedules, inventory levels, and logistics context, it is not

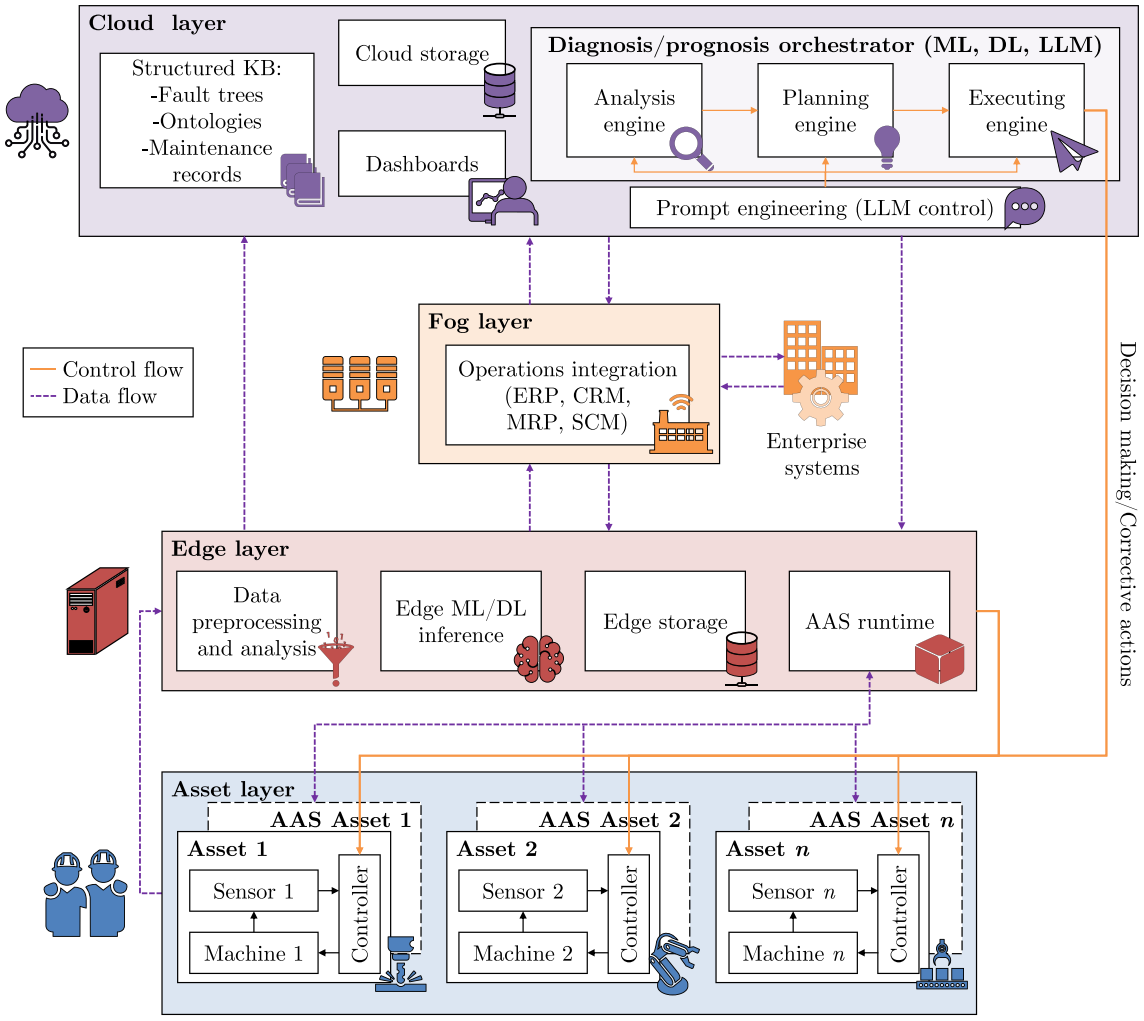

Fig. 3. Layered architecture for self-diagnosis services in CPS, comprising four layers: Asset, Edge, Fog, and Cloud. Each layer handles a distinct function, from data acquisition and preprocessing to operations management and high-level decision-making, enabling real-time fault detection and corrective actions.

strictly required for basic edge-to-cloud communication. In smaller installations or early prototypes, data may flow directly between edge nodes and the cloud. However, it becomes indispensable in large-scale industrial deployments. For example, integration with enterprise systems such as ERP, MRP, or supply-chain management often requires a dedicated fog tier to synchronise production data with planning processes in real-time. Similarly, in regulated or security-critical domains, the fog layer enables local processing of sensitive information before it is transmitted to the cloud, reducing both latency and exposure.

These scenarios highlight that although the fog layer is not strictly required, broader industrial adoption may find it a non-optional element. For mapping completeness assessment, this layer is represented by its core function: operations management integration.

The **cloud layer** delivers scalable computation, storage, and advanced analytics capabilities. It hosts cloud-based databases for monitoring, historical analysis, and long-term data retention. In addition, dashboard services are available to query the historical and real-time status of assets.

To support reasoning, explanation, and knowledge reuse, the cloud layer should incorporate a structured knowledge base (KB). This can include fault trees, maintenance records, causal relationships, and semantic models, such as ontologies. These KBs provide a foundation for consistent interpretation and decision support. The key component of this layer is a diagnosis/prognosis orchestrator, which comprises three engines:

- **Analysis engine**: Ingests preprocessed feature data and historical records to perform fault detection, classification, and RCA using ML/DL models or LLMs.
- **Planning engine**: Consumes analysis outputs and KB insights to generate knowledge-driven contingency plans with confidence-scored recommendations.
- **Execution engine**: Orchestrates secure dispatch of approved plans back to edge and asset layers. It also monitors execution feedback, and updates system status.

LLMs can enhance the diagnosis/prognosis engine by reasoning over unstructured, incomplete, or heterogeneous data, synthesising contextual insights, and autonomously recommending corrective or preventive actions. They can also generate actionable outputs, such as recommended interventions and confidence-scored diagnoses, which inform decision-making at the edge and asset layers. A prompt engineering component can guide and constrain LLM behaviour across these internal engines to ensure safe, accurate, and context-relevant outputs aligned with user goals and safety standards.

Once the orchestrator has run its internal pipeline, outputs are transmitted back to the edge layer, where they inform decision-making processes. Operators are notified via local human–machine interfaces (HMIs) in the asset layer, which present recommended actions or raise alerts when human intervention is required. After actions are taken, system status updates return to the orchestrator, completing

Table 2
Mapping of architecture layers to RAMI 4.0. The Communication layer of RAMI is included across all architecture layers because reliable data exchange, interoperability, and integration of components across hierarchy levels and system functions are essential for self-diagnosis services.

Architecture layer	RAMI 4.0 Hierarchy Level	RAMI 4.0 Layers
Asset	Product and Field device	Asset, Integration, and Communication
Edge	Control device and station	Integration, Communication, and Information
Fog	Work centre and Enterprise	Communication, Information, and Functional
Cloud	Enterprise and Connected world	Communication, Functional, and Business

the diagnostic feedback loop and ensuring system learning and adaptation. In the mapping completeness evaluation, this layer includes: cloud storage, diagnosis/prognosis orchestrator, structured KBs, prompt engineering mechanisms, and dashboards.

Table 2 maps the architecture layers to the layers and hierarchy levels of RAMI 4.0 [49]. It is worth noting that the services in this architecture can integrate with other services to enhance their analysis capabilities and provide more comprehensive insights and competencies. In the context of self-diagnosis, outputs can be published to the cloud layer for storage, analysis, and integration with additional services. These outputs can then be re-utilised as inputs for downstream applications and services, such as maintenance planning services or performance analytics services.

The proposed architecture directly addresses integration with diverse industrial landscapes, including legacy systems, through several mechanisms. Its modular, layered design (asset, edge, fog, and cloud) inherently supports incremental integration, allowing specific layers to interface with existing infrastructure. The adoption of open standards, such as REST APIs and MQTT, is essential for bridging communication gaps, even if initial custom adaptations are required for proprietary protocols.

KBs are central for integrating with legacy systems. They provide a formal and customisable structure for capturing diagnosis and maintenance procedures, enabling unambiguous information exchange between heterogeneous systems with varied standards by leveraging semantic representations. Furthermore, the integration of LLMs into the architecture provides capabilities to understand and reason with unstructured data, such as maintenance reports and operational logs, or process time series data into textual representations, offering a critical pathway for incorporating valuable information from older, less digitised systems. The adoption of Industry 4.0 standards, such as asset administration shell (AAS), further enhances compatibility and seamless integration with existing manufacturing systems.

4.2. Requirements of a self-diagnosis service

Self-diagnosis services are designed to monitor and analyse the operational state of machines in real-time, enabling proactive maintenance and minimising downtime. These services leverage advanced ML and DL algorithms to detect anomalies, predict faults, find the root cause of failures, and provide recommendations for maintenance actions. They are responsible for analysing the data collected through sensors from machines and equipment to determine their health and performance. Sensor data typically includes variables such as temperature, pressure, and vibration. The system then analyses these data to find and predict anomalies and issues, and sends an output to specify possible solutions.

Six major requirements for a self-diagnosis service were identified in this study, each with its dedicated, extended explanation. The requirements are detailed as follows:

R1. Connectivity: Enabling connectivity among various equipment is essential. Therefore, all manufacturing resources should be accessible and capable of communicating with other resources on the network. This entails establishing connections between data source elements (e.g., sensors), manufacturing assets (e.g., machines), services (e.g., fault diagnosis), and all network layers, including edge, fog, and cloud. To achieve seamless interoperability and facilitate data exchange among these resources, it is crucial to employ open standards and protocols for connectivity in Internet of Things (IoT) and Industrial IoT (IIoT) networks. At the field level, commonly utilised transport protocols include Profibus, Ethercat, and CAN. In addition, open standards such as OPC UA, AutomationML, and MT connect play a pivotal role in enabling data exchange and fostering interoperability within the IIoT and IoT. This requirement implicitly supports integration with legacy devices by mandating the flexibility to establish connections across diverse data sources and network layers, often requiring custom adapters for proprietary industrial protocols prevalent in older equipment. In practice, many manufacturing environments rely on legacy protocols such as Profibus or Modbus. To support such contexts, the guideline anticipates the use of protocol converters and OPC-UA gateways as bridging mechanisms. When direct replacement is not feasible, encrypted retrofits, such as tunnelling legacy traffic within TLS or VPN layers, can safeguard data confidentiality and integrity. Cybersecurity measures, such as firewalls, OAuth access tokens, and TLS security protocols, must also be deployed to safeguard the confidentiality and integrity of the gathered data during transmission from the edge to the cloud and while at rest in storage.

R2. Data management: Manufacturing infrastructure should have the capability to collect data from edge devices, transfer data between layers, and store it in storage systems. Each layer is in charge of a specific data management task:

(a) The asset layer is responsible for data acquisition from physical assets, as well as displaying process data in the HMI of the assets.
(b) The edge layer is responsible for data management and storage in local, data preprocessing, and transfer of the preprocessed data to the fog layer.
(c) The fog layer serves as the bridge between the edge and the cloud layers. It is responsible for data transfer and sharing data with third-party systems, like enterprise resource planning (ERP) systems.
(d) The cloud layer possesses higher data storage capacity and high computing power. It hosts services that run on the transferred data and serves as the central hub for advanced analytics and decision-making support. In addition, the cloud layer is responsible for storing the trained ML and DL models used for fault detection, prediction, and other smart services. This layer must also support the management and storage of structured knowledge, as outlined in R6, which is critical to address and manage large volumes of data.

LLMs can be leveraged across this layered infrastructure to act as autonomous data pipeline managers. This includes tasks such as automated data preprocessing, cleaning, labelling, and visualisation, which significantly reduce manual effort and human-induced subjectivity. Furthermore, LLMs can intelligently infer missing values in time series data or generate event series when data is incomplete, ensuring continuity and reliability. They can

also manage, integrate, and evaluate multiple domain-specific models using a scoring mechanism to aggregate and prioritise them, enabling optimal model selection for downstream analysis and diagnosis [24].

Particular attention must be paid to data security and privacy throughout the data management process. This includes implementing required data security protocols, data protection tools, and well-defined policies to ensure robust protection across layers. Additionally, secure access to information displayed on business dashboards and HMIs must be enforced, potentially using two-factor authentication (2FA) mechanisms to prevent unauthorised access. For highly sensitive enterprise data, selected models or datasets may need to be deployed locally on the edge to ensure compliance with security and privacy policies.

Therefore, it is necessary to establish a tailored IT infrastructure. This includes defining the data acquisition frequency (i.e., sampling period), determining the required data storage capacity, information structure and data models (e.g., AADL, UML, MARTE), and implementing the necessary IT resources for data transfer (e.g., routers, switches, servers, and gateways).

It is essential to note that, while the cloud layer provides significant computational power and storage capacity, it introduces latency to data transfer and analysis. Therefore, if real-time data analysis is required, the edge layer should provide additional data storage and computing capability to perform time-critical tasks.

R3. Monitoring and analysis: Continuous data monitoring and analysis are essential to identify faults, breakdowns, malfunctions, and anomalies, as well as to diagnose the underlying causes of failures. This encompasses a range of events, from simple changes in resource states to more complex events, such as fluctuations in value and intricate patterns. Therefore, observed data must be continuously processed in the edge layer. First, raw data should be preprocessed by applying data filtering and cleaning, eliminating outliers through density clustering, normalisation, and scaling, as well as feature extraction techniques. ML algorithms like decision trees can be employed for feature extraction. The extracted feature data should be sent to the cloud to be stored as historical data and to be used by the diagnosis/prognosis engine for fault identification and classification. The engine can also be used for RCA or prognostics of the asset. For this, establishing the required key performance indicators (KPIs) is crucial.

This requirement can be significantly enhanced by integrating LLMs. They can act as controllers for prediction tasks, recommending optimal methods and initiating their deployment. For RCA, LLMs can identify underlying causes by reasoning over failure data and multi-event failures. They can supervise and refine the causal inferences of traditional algorithms by integrating domain knowledge and logic. Furthermore, LLMs can also function as decision-support agents by explaining diagnoses and suggesting mitigations, while referencing KBs or dynamically querying networked resources for unfamiliar faults.

The outcome of this phase should be a change request. This request will be further explored in R4 for contingency purposes.

R4. Planning: Self-diagnosis requires a mechanism for making reports and contingency plans. Fault assessment reports should be derived from expert knowledge and the KB component, as well as from RCA reports from R3. This requirement emphasises that planning is fundamentally knowledge-driven. It involves leveraging the internal KB to generate detailed fault assessment reports, RCA outputs, and recovery plans.

In the event of failures, contingency plans should be created to address the issues, make recovery plans, or provide necessary actions to the operators to mitigate the problem. The developed plan can range from basic actions, such as shutting down the system, to more intricate tasks, like altering the process structure or the control model.

The diagnosis/prognosis engine should produce specific, actionable insights that enable informed decision-making. These outputs should include high-level recommended actions (e.g., "Perform corrective maintenance on component X" or "Adjust process parameters Y to value Z") and be accompanied by confidence scores for diagnoses or prognoses (e.g., "95% confidence in root cause identification").

LLMs can support in developing these knowledge-driven, actionable plans. They can autonomously generate detailed fault assessment reports, RCA outputs, and recovery strategies based on contextual input and structured knowledge. They can also recommend actions for operators or automated execution, such as maintenance interventions or parameter adjustments, utilising the KB described in R6. By doing so, LLMs can ensure plans are consistent with system context, operational policies, and business priorities. Lastly, new knowledge derived from failure reports should be used to provide feedback to the internal knowledge storage systems described in R6, supporting the continuous adaptation of system KPIs and policies.

R5. Executing: Contingency plans developed in R4 should be implemented either in the target software or through operator guidance. Execution may involve terminating a task, correcting the behaviours of actuators, or issuing clear instructions to the operators for execution. In addition, all failure reports and recommended actions must be promptly alerted and visualised through business dashboards and HMIs for relevant stakeholders, e.g., operation supervisors and maintenance technicians. Given the potential sensitivity and confidentiality of diagnostic information, secure access to the information must be enforced through appropriate cybersecurity measures, such as 2FA and role-based access control (RBAC).

LLMs may support this requirement by enhancing action validation, refinement, and policy evolution. They can evaluate proposed responses for safety and effectiveness (e.g., via digital twin simulations), refine actions to optimise their impact through iterative adjustments, and facilitate long-term adaptability by integrating feedback into evolving decision policies. These capabilities can increase robustness and adaptability but remain non-critical, to be adopted selectively depending on system needs and available resources.

R6. Knowledge: All data and acquired knowledge should be stored in edge/cloud storage databases, as they form the core body of the self-diagnosis service. This knowledge must be organised into a structured and formalised KB, which serves as the central repository for reasoning, decision-making, and continuous improvement in the system. The KB includes the following components:

- **Historical diagnostic and maintenance data**: A repository of past problem-solution pairs, annotated diagnostic records, expert assessments, and corrective/preventative action (CAPA) plans. This historical data supports case-based reasoning and model retraining for continuous improvement.
- **Policies**: Prescriptive strategies and governance rules that inform autonomous behaviour and guide operator interventions under varied fault conditions.
- **Formal ontologies**: Semantic representations of equipment structure, components, operating conditions, failure modes, observable symptoms, and associated maintenance procedures. These ontologies enable semantic interoperability, structured reasoning, and domain-specific inference. This semantic structure enables the formalisation and reuse of knowledge from various sources, including legacy systems.

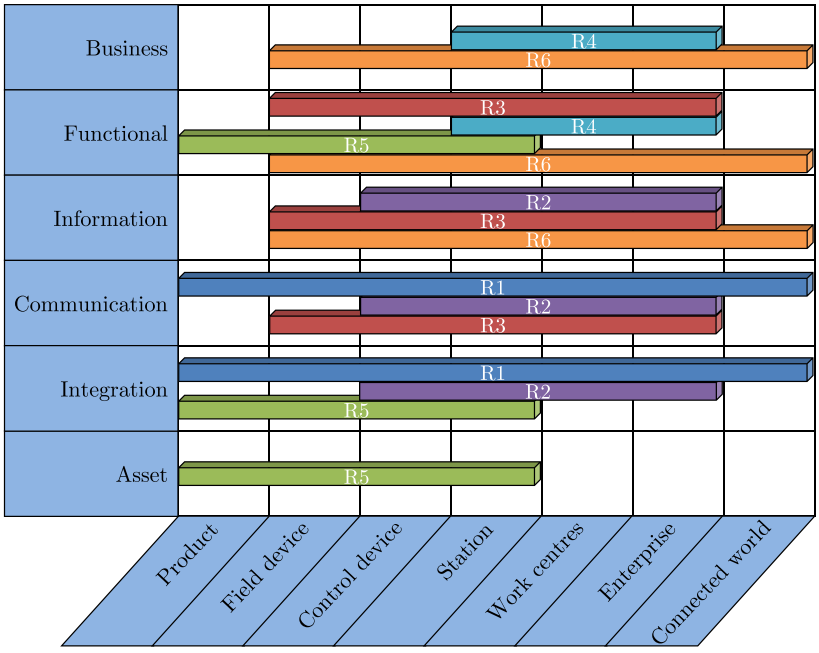

Fig. 4. Visual mapping of self-diagnosis system requirements (R1–R6) across the RAMI 4.0 architecture. Each requirement intersects multiple layers and hierarchy levels, indicating its technical and functional scope.

- **Fault trees and rules**: A library of predefined diagnostic rules and fault trees that capture known fault patterns using "if...then" logic. These are used for automated RCA and early fault detection, enabling rule-based reasoning alongside data-driven methods.
- **Runtime information**: Current system states, KPIs, and contextual runtime information needed for accurate fault interpretation and decision support.

The KB must be designed to support automated reasoning, allowing the system to infer likely causes and appropriate responses to emerging issues. Moreover, it must support continuous learning and refinement, incorporating feedback from new diagnostic outputs, execution outcomes, and both automated and human-led interventions. This ensures the evolving accuracy and relevance of the knowledge used by self-diagnosis services over time and across various scenarios.

Fig. 4 illustrates a visual mapping of the self-diagnosis requirements (R1–R6) across the RAMI 4.0 architecture. Table 3 maps the requirements to the architecture layers proposed in this work. Furthermore, the complete list of requirements is schematically presented in Tables 4, 5, 6, 7, 8, and 9, with each table corresponding to a specific major requirement. The tables can be read as follows: from left to right, each row contains specific *sub-requirements*, which define the necessary *resources* for implementation. Each resource is also represented in an *example*, to support interpretation. Additionally, the associated *inputs* and *outputs*, as well as their respective *sources*, are also specified. With such a structure, SMEs and big companies can access and interpret the information conveniently.

4.3. Tiered-classification of architectural components and sub-requirements

To account for differences in technological maturity and to avoid penalising solutions that do not yet incorporate advanced capabilities, each architectural component and requirement sub-element defined in this section is classified into one of three tiers: Core, Secondary, or Advanced. The Core tier contains the fundamental elements necessary for a cyber–physical system to perform baseline self-diagnosis and therefore represents the minimum compliance target. The Secondary tier encompasses enhancements that improve capability, integration, or performance, but are not strictly required for baseline operation. The Advanced tier includes advanced or experimental features, such as LLM-enabled modules, that can significantly augment functionality but whose absence does not prevent compliance with the core requirements.

Table 10 summarises this classification for both the architecture and the requirements, providing a structured reference for the evaluation process.

Table 3
Mapping of requirements to the proposed architecture.

Architecture layer	Requirements
Asset	R1, R2, and R5
Edge	R1, R2, R3, R5, and R6
Fog	R1, R2, and R5
Cloud	R1, R2, R3, R4, R5, and R6

5. Case study demonstration

This section demonstrates the case study protocol defined in Section 3. The protocol was applied to the OptiTwin system [17], a cloud-based TCM CPS with self-diagnostic capabilities. The aim was not to validate OptiTwin itself, but to show how the protocol can be used to assess compliance with the self-diagnosis guideline and to determine the readiness level of systems.

The analysis mapped the components and functionalities of OptiTwin to the proposed guideline. The degree of requirement fulfilment was then evaluated, and overall alignment was quantified using the defined metrics. This process also revealed implementation gaps that may guide future refinements.

The study shows how practitioners can apply the protocol to position their own self-diagnosis solutions within the readiness matrix

Table 4
Connectivity requirements for self-diagnosis services.

Sub-requirement	Resources	Example	Stakeholders	Inputs	Input sources	Outputs	Output targets
Communication protocols	Open communication standards	OPC UA, MTConnect, AutomationML, EtherCAT, CAN, Profibus	Automation engineers, network administrators	Industrial communication specs	Standards bodies (e.g., OPC Foundation)	Interoperable communication streams	Machines, sensors, controllers
Device and network interoperability	Routers, IoT gateways, adapters, IIoT platforms	MQTT brokers, protocol converters, edge middleware	System integrators, IT architects	Device configuration	Equipment vendor documentation	Unified data interfaces	IIoT platform, edge nodes
Wireless infrastructure	Industrial wireless networks	5G, Wi-Fi 6, private LTE, LoRaWAN	IT infrastructure team, network planners	Network design parameters	Network service providers	Low-latency, reliable connectivity	Field and edge devices
Legacy device integration	Protocol bridges, custom adapters	OPC UA wrapper for Modbus device	System integrators, OT engineers	Legacy device specifications	Manufacturer documentation	Backwards-compatible data pipelines	Custom middleware, legacy controllers
Cybersecurity for data transmission	Security protocols and tools	TLS, firewalls, OAuth tokens, VPN	IT security officers, compliance teams	Cybersecurity policies	Security guidelines, IT standards	Secured data in motion and at rest	Edge, fog, and cloud platforms

Table 5
Data management requirements for self-diagnosis services.

Sub-requirement	Resources	Example	Stakeholders	Inputs	Input sources	Outputs	Output targets
Multi-layered data acquisition	Sensors, controllers, HMIs, asset interfaces	PLCs, industrial sensors, machine HMIs	Process engineers, OT technicians	Process variables, status data	Physical assets	Real-time sensor data streams	Asset layer
Fog-layer integration	Middleware, fog nodes, API bridges	ERP connectors, message brokers	System integrators, IT architects	Preprocessed edge data	Edge layer	Aggregated and shared datasets	Fog layer
Cloud-based analytics and storage	Cloud storage, compute instances, model hosting services	AWS S3, Azure ML, BigQuery	Data scientists, AI engineers	Aggregated data, knowledge structures	Fog layer, training pipelines	Models, analytics results, decision insights	Cloud layer
Semantic data modelling and structuring	Ontologies, meta-models, schema languages	AADL, MARTE, AutomationML, UML	System architects, domain modellers	Design specs, system structure	Engineers, CAD/PDM systems	Structured knowledge graphs	Knowledge base
Data privacy and access control	Encryption tools, TLS, RBAC, 2FA, firewalls	OAuth tokens, TLS protocols, access roles	Cybersecurity officers, IT admins	Credentials, access policies	Security team	Secured and traceable data access	All layers
Local model hosting for sensitive data	Edge nodes, local repositories, container runtimes	On-premise ML services, Docker images	IT operations team, compliance officers	Sensitive enterprise data	Internal processes	Privacy-preserving models and outputs	Edge layer
LLM-powered data pipeline automation	LLM runtime, orchestration scripts	Data cleaning, labelling, value inference	Data engineers, AI researchers	Raw/incomplete data	Sensors, logs	Enriched and cleaned datasets	Edge/cloud LLM services

and to identify opportunities for improvement. This section presents both the practical implementation of the protocol and the results of the OptiTwin evaluation, providing a concrete example of how to assess *coverage*, *functionality*, and *improvement potential* in real-world settings.

For the sake of brevity, only the case selection, data analysis techniques, and expected outcomes are summarised in this paper. The full description of the case study is provided in Appendix A as supporting material (OptiTwin_case_study.pdf).

5.1. Case selection

OptiTwin is a project funded by the Provincial Council of Gipuzkoa, Spain (Department of Economic Promotion, Rural Environment, and Territorial Balance) under the 2020 call for the "Support Program for the Gipuzkoan Network of Science, Technology, and Innovation". It is a collaborative effort involving three research groups of MU: (i) High-performance machining,[1] (ii) Software and systems engineering, and (iii) Data analysis and cybersecurity.[2]

OptiTwin was chosen because it represents a state-of-the-art industrial CPS. It offers heterogeneous data integration, multi-layered architecture, and data-driven self-diagnosis functionalities. Furthermore, its accessibility to the research team made it suitable for detailed mapping and evaluation. Specifically, key elements that make OptiTwin an exemplary case for studying the effectiveness of self-diagnosis services in CPS include heterogeneous data integration, multi-layered architecture, advanced feature extraction and data-driven modelling. OptiTwin

[1] http://www.mondragon.edu/mar
[2] http://www.mondragon.edu/danz

Table 6
Monitoring and analysis requirements for self-diagnosis services.

Sub-requirement	Resources	Example	Stakeholders	Inputs	Input sources	Outputs	Output targets
Edge-level preprocessing	Preprocessing libraries and ML models	Moving average filter, Min-Max scaling, z-score, feature extraction	Data engineers, data analysts	Raw time-series data	Asset layer	Cleaned and enriched data	Edge layer
Fault detection/ identification	Classification algorithms, LLMs, fault rules	ML models for fault detection and state classification	AI engineers, cloud engineers	Preprocessed data, historical logs	Edge layer, cloud storage	Fault types, condition labels	Analysis engine
RCA	Fault trees, causal inference models, LLMs	Fault tree analysis, probabilistic reasoning, domain-guided RCA	Maintenance engineers, reliability engineers	Fault logs, failure patterns	Cloud models, KBs	Identified root causes	Analysis engine
Prognostics	Time-series models, LLMs	Condition monitoring, RUL estimation	Data scientists, planners	Cleaned historical data, health trends	Edge layer, cloud storage	Prognostic forecast	Analysis engine
Performance analysis	KPI templates, monitoring dashboards	Performance monitoring software: Grafana, Prometheus	Operations managers, quality analysts	Health indicators, failure data	Edge layer, diagnosis/ prognosis engine	Prediction of failure/predictive maintenance	Dashboard service
LLM-based orchestration	LLM runtime, orchestration scripts, prompts	Model selection, RCA reasoning, mitigation suggestions	AI specialists, diagnostic engineers	Fault logs, candidate models, queries	Cloud services, KBs	Selected models, RCA explanations, change requests	LLM orchestrator

Table 7
Planning requirements for self-diagnosis services.

Sub-requirement	Resources	Example	Stakeholders	Inputs	Input sources	Outputs	Output targets
Alerting and accessibility	HMI interfaces, alarm systems, secure dashboards	Failure visualisation, alarm signals, role-based dashboards with 2FA	Operations supervisors, maintenance technicians	Fault/anomaly detection results	Analysis engine	Alarms, failure alerts, operator prompts	HMI, alerting systems
Fault assessment	Ontologies, RCA results	Fault severity report with RCA trace and confidence score	Maintenance team	RCA output, historical failure data	Analysis engine, KBs	Fault assessment report	Maintenance team
Policies adjustments	KBs, user feedback, fault analysis results	"Update inspection interval for pump X", "Revise process model Y"	Policy managers, process engineers	Feedback and analysis results	Analysis engine, maintenance logs, operators	Policy adjustment proposal	KBs
Adaptation plan/Recovery plan	Domain policies, fault taxonomies, LLM planners	"Shut down unit A","Adjust valve B to X", "Replace component C"	Maintenance team	Diagnostic results, business policies	Analysis engine, KBs	Actionable recovery plan	HMI, ERP, maintenance team

employs statistical feature extraction (e.g., mean, variance, RMS, skewness, and kurtosis) from sensor data to train ML models for tool condition monitoring. Handling both raw and preprocessed data in real time requires the platform to balance computational efficiency with diagnostic reliability, and real-time monitoring and decision support.

OptiTwin operates in the digital manufacturing domain to create data-driven models for enhancing digital twins used in machining operations. Specifically, it has emphasised signal acquisition and analysis, along with tool wear prediction. This integration has been achieved by incorporating a cloud-based system into a CNC machining centre. As an example of the practical application of OptiTwin, a dataset containing raw signals and corresponding tool-wear measurements has been published for open research use [61].

5.2. Data analysis techniques

Mapping analysis was employed as the primary data analysis technique. The functionality and components of the OptiTwin system – identified through document analysis and system observation – were manually mapped to the elements of the proposed self-diagnosis architecture and requirements. The mapping was performed according to the criteria defined in Section 3.4.3.

Although the analysis was conducted manually without specialised software tools, the approach enabled a detailed and context-aware assessment, yielding fine-grained insights into how the system aligned with or diverged from the proposed architecture and requirements. The outcomes of this mapping identified key features of the self-diagnosis capabilities of OptiTwin and hinted towards effective implementation

Table 8
Executing requirements for self-diagnosis services.

Sub-requirement	Resources	Example	Stakeholders	Inputs	Input sources	Outputs	Output targets
Plan execution	Actuation interfaces, operator instructions, control software	Execute recovery plan: shutdown, reconfiguration, part replacement	Maintenance team	Recovery plan	Planning engine	Executed control action, status confirmation	Physical system, operator terminals
Action validation	Digital twin, simulation models, LLM agents	Evaluate "Is action X safe under Y conditions?" using prior cases and simulation	AI engineers, safety engineers	Proposed actions, safety constraints	Planning engine, digital twin	Action approval or rejection	Planning engine
Action refinement	LLM prompt loops, adaptive tools, simulation feedback	Re-generating plan if unsafe: adjust parameters or escalate action level	Autonomous agents, LLM orchestration framework	Validation results, policy limits	Simulation outcomes, KBs	Refined action or replan	Planning and execution engines
Fault correction	Spare parts, manuals, work permits, software patches, actuators	Replacing faulty component, restart machine, updates on software, actions on actuators	Maintenance team, automation system	Execution plan, asset state	Execution engine, asset sensors	Executed action on asset, maintenance report, updated asset state	Physical system
Reporting	Edge/cloud storage, business dashboards, RBAC, 2FA access control	Display root cause + recommended action with restricted visibility	Operations supervisors, data analysis team	Failure data, diagnosis outcome, plan status	Planning engine	Failure report, dashboard view, secure alerts	Operators

Table 9
Knowledge requirements for self-diagnosis services.

Sub-requirement	Resources	Example	Stakeholders	Inputs	Input sources	Outputs	Output targets
Historical diagnostics	Edge/cloud storage	Historical data, annotated fault logs, maintenance outcomes, expert notes	Data scientists, knowledge engineers	Production data	Sensors, analysis engine	Reports	Data scientists
Policies	KBs	Maintenance rules, intervention criteria, safe operating limits	Compliance officers, policy administrators	Rules and conditions for failures and faults	Expert knowledge	Policies	Policy administrators
Ontologies	OWL-based semantic models	Ontology of machines	Domain experts, knowledge engineers	Expert models, documentation, standards	External knowledge, system integrators	Formal ontology	Planning and execution engines
Fault trees and rules	Rule engines, fault-trees libraries, expert-defined models	"If temperature > X and vibration > Y, then bearing failure"	Reliability engineers, AI developers	Historical failures, expert input	Maintenance KBs, domain experts	Structured rules, diagnostic logic	Analysis engine
Runtime information	Time-series DB, real-time dashboards, system monitors	Live asset condition, KPIs	Operations manager	Sensor data, live metrics	Edge layer	Context-aware input for inference	Analysis and planning engines
Knowledge refinement	Feedback processors, LLMs, policy trainers	Adjust rules and policies based on new outcomes	AI developers, policy administrators	Diagnostic feedback, execution logs	Planning and execution outcomes	Updated KB entries and weights	KBs

strategies. The following subsections present the results of this analysis, structured around the mapping of architecture and requirements.

5.2.1. Architecture mapping

Fig. 5 illustrates how the architecture of OptiTwin is mapped to the architecture proposed in this study. The components and services offered by OptiTwin are mapped as follows.

- **Asset layer**: The primary asset in the OptiTwin demonstrator is a Lagun GVC 1000 CNC machining centre with a Fagor CNC 8065 controller. The embedded system in the machine collects signals from several internal sensors. These include the actual revolutions per minute (RPM), spindle angular position (in degrees), spindle position in the X, Y and Z axes (in μm), root-mean-square (RMS) current feedback (in Arms), active power of the spindle motor (in

Table 10
Tiered classification of architecture components and sub-requirements.

Element	Tier	Description
Architecture components	**Asset layer**	
	Core	Machines, controllers, sensors.
	Secondary	AAS.
	Edge layer	
	Core	Data preprocessing and analysis, edge storage.
	Secondary	AAS runtime, edge ML/DL inference for privacy-critical tasks.
	Fog layer	
	Secondary	Operations integration (ERP, CRM, MRP, and SCM systems).
	Cloud layer	
	Core	Cloud storage, ML/DL-based analysis engine, structured KB (fault trees, ontologies), dashboards.
	Secondary	Rule/model-based planning and execution engines.
	Advanced	LLM integration into diagnosis/prognosis orchestrator, prompt engineering for LLM control.
Requirements and sub-requirements	R1 **Connectivity**	
	Core	Communication protocols, device and network interoperability, cybersecurity for data transmission.
	Secondary	Legacy device integration, wireless infrastructure.
	R2 **Data management**	
	Core	Multi-layered data acquisition, cloud-based analytics and storage, data privacy and access control.
	Secondary	Semantic data modelling and structuring, fog-layer integration.
	Advanced	LLM-powered data pipeline automation, local model hosting for sensitive data.
	R3 **Monitoring and analysis**	
	Core	Edge-level preprocessing, fault detection, prognostics.
	Secondary	Fault identification, RCA, Performance analysis.
	Advanced	LLM-based orchestration.
	R4 **Planning**	
	Core	Alerting and accessibility.
	Secondary	Fault assessment, policies adjustments, adaptation/recovery plans.
	Advanced	LLM-generated adaptation and recovery plans.
	R5 **Executing**	
	Secondary	Plan execution, action validation, action refinement, fault correction, reporting.
	Advanced	LLM-based action validation and refinement.
	R6 **Knowledge**	
	Core	Historical diagnostics, policies, fault trees and rules, runtime information.
	Secondary	Ontologies, knowledge refinement.
	Advanced	LLM-based knowledge refinement.

W), and torque feedback of the spindle and the table motors in the X, Y and Z axes (in Nm). These signals are retrieved from the Fagor CNC API by the OptiTwin experiment designer and are sent in real time to the **edge layer**. Process information is also transmitted to the edge layer. All Core-tier components defined in the reference architecture for the asset layer are fulfilled, yielding a $\mathcal{A}$ **of 100%**. For the Secondary tier, which only includes the AAS component, the **Secondary** $\mathcal{A}$ **is 0%**, since this component is not implemented.

- **Edge layer**: A laboratory workstation with OptiTwin is located in the edge layer. Data preprocessing and analysis are conducted on the raw signals obtained from the **assets layer**. Signal cleaning is performed through a moving-average filter, followed by z-normalisation. Additionally, feature extraction is carried out to derive KPIs for machining processes. These KPIs include statistical characteristics from time, frequency, and time–frequency domains. Two additional functions are available in this layer: (i) local storage of process data, raw signals, and KPIs; and (ii) monitoring of condition ranges in the preprocessed signals and KPIs. Upon meeting the condition ranges, the data is sent to the **cloud layer** for tool wear prognosis. All Core-tier components of the edge layer reference architecture are implemented in OptiTwin, resulting in a $\mathcal{A}$ **of 100%**. For the Secondary tier, the corresponding components are not implemented, yielding a **Secondary** $\mathcal{A}$ **of 0%**.
- **Fog layer**: At present, a fog layer is not integrated into the OptiTwin project. The system is still in an early developmental phase. Current priorities focus on establishing the core functionalities of tool wear prediction and ensuring precision and dependability. As the project progresses, including a fog layer will be considered. This would enhance real-time capabilities, facilitating connectivity with production and logistics systems, such as MRP and SCM systems. No score is reported for the Core tier, since this layer has no Core-tier components. For the Secondary tier, the operations integration component is not implemented, resulting in a **Secondary** $\mathcal{A}$ **of 0%**.
- **Cloud layer**: OptiTwin offers cloud storage in two ways: time-series data storage through InfluxDB and file-based data storage via Dropbox. As an analysis engine, this layer hosts a ML/DL-based tool wear monitoring service for milling and drilling operations. Additionally, a Grafana dashboard is integrated to enable real-time and historical data visualisation.
  The tool wear monitoring service operates as part of the analysis engine within the Diagnosis/Prognosis orchestrator. It leverages

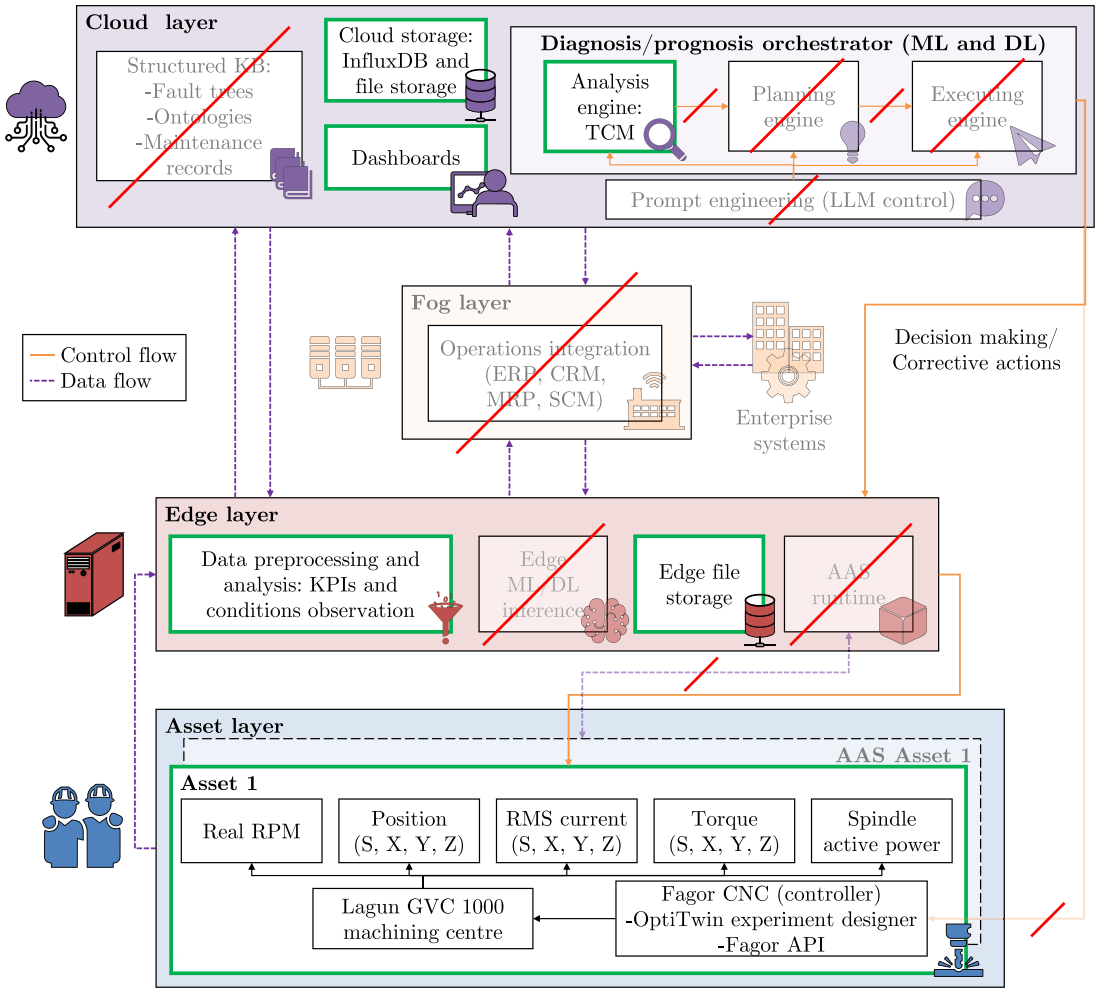

Fig. 5. Architecture mapping of OptiTwin to the proposed guideline. Aligned components are highlighted with green borders. Missing components and data/control flows are greyed out and marked with a red diagonal line.

preprocessed signals and extracted KPIs from the **edge layer** as inputs. The DL models predict the tool wear, and the results are stored in the cloud. Predictions are then forwarded back to the edge layer and ultimately to the CNC controller in the **asset layer**, where they are displayed and used to trigger warnings if a threshold is exceeded.

According to the reference architecture, the Core tier of the cloud layer comprises four components: (i) data storage, (ii) dashboards, (iii) the analysis engine and services, and (iv) the structured knowledge base (KB). In OptiTwin, the first three are implemented, while the structured KB is missing. This results in a **Core-tier $\mathcal{A}$ of 75%**. For the Secondary and Advanced tiers, no components are implemented, giving both tiers a $\mathcal{A}$ of 0%.

The OptiTwin architecture demonstrates strong alignment with the proposed self-diagnosis architecture, particularly at the Core tier. It enables real-time monitoring of processes and predictive assessment of tool wear, supporting improved machining efficiency. The system integrates sensor data collection, KPI feature extraction, and predictive model deployment to provide accurate, real-time insights.

Architectural mapping completeness was calculated based on (1). As shown in Table 11, the Core tier achieved an $\mathcal{A}$ of 88.89%, while the Secondary and Advanced tiers remain unimplemented. The results indicate a global $\mathcal{A}$ of 47.06% and classify OptiTwin as having *Core coverage*. This tiered assessment highlights the current focus on essential functionalities, while indicating potential areas for future enhancement.

5.2.2. Requirements mapping

The process of requirements mapping linked the self-diagnosis requirements to the components and functionalities of the OptiTwin platform. Tables 12 and 13 detail the connectivity and planning requirements, which serve as illustrative examples of the complete requirement list. The full requirement list is detailed in the supplementary material (OptiTwin_requirement_analysis.xlsx) of Appendix A. It presents an overview of how the requirements from Section 4 are associated with the implementation of OptiTwin. The tables report the fulfilment score for the sub-requirements, along with the resources, inputs, and outputs. Tier-specific and global $\mathcal{F}$ were calculated based on (2).

The fulfilment of the **connectivity** requirement demonstrates that OptiTwin supports multiple industrial communication mechanisms across the CNC machine, edge, and cloud. Communication protocols are fully realised through a combination of CNC APIs, MQTT brokers, HTTPS services, and Ethernet, ensuring robust and interoperable data transmission. Device and network interoperability is achieved through the use of MQTT streams, CNC APIs, and CSV exchange, although no protocol converters or middleware are deployed. Wireless infrastructure and legacy device integration are not applicable, since OptiTwin operates exclusively with Ethernet-based networking and modern CNC equipment. Cybersecurity is comprehensively addressed through encrypted HTTPS channels, firewall protection, and strict VPN-based access control to the edge and asset layers, ensuring that external connectivity is both secure and compliant with industrial IT practices.

Table 11
Layer-wise and tier-specific evaluation of architectural mapping completeness for the OptiTwin system. The weighted contribution (W_i) of each layer to Core, Secondary, and Advanced tiers is detailed, as well as the global alignment with the proposed self-diagnosis architecture.

Layer	w_i	Core $\mathcal{W}_i$	Secondary $\mathcal{W}_i$	Advanced $\mathcal{W}_i$	$\mathcal{W}_i$
Asset	4	300% (3/3)	0% (0/1)	–	300%
Edge	4	200% (2/2)	0% (0/2)	–	200%
Fog	1	–	0% (0/1)	–	0%
Cloud	8	300% (3/4)	0% (0/2)	0% (0/2)	300%
Total	17	800%	0%	0%	800%
				Core $\mathcal{A}$	**88.89%**
				Secondary $\mathcal{A}$	**0%**
				Advanced $\mathcal{A}$	**0%**
				Global $\mathcal{A}$	**47.06%**

Table 12
Analysis of the self-diagnosis **Connectivity** requirement for OptiTwin.

Sub-requirement	Fulfilment score	OptiTwin resources	OptiTwin inputs	OptiTwin outputs
Communication protocols	5 - Yes	• CNC API (CNC-to-edge data collector) • MQTT broker (edge-to-cloud dashboard) • HTTPS (edge-to-cloud tool wear service) • Ethernet (IEEE 802.3)	CNC machining data	Interoperable formatted data
Device and network interoperability	4 - Substantially met	• CNC API • MQTT data streams • CSV file exchange (edge-to-cloud)	Device configuration, CNC logs	Unified data interfaces (CSV, MQTT)
Wireless infrastructure	1 - Not met (Severity: 1 - Low)	Not applicable, ethernet-based deployment only	–	–
Legacy device integration	1 - Not met (Severity: 1 - Low)	Not applicable, modern CNC equipment, no legacy interfaces	–	–
Cybersecurity for data transmission	5 - Yes	• HTTPS for data transfer • Firewalls at edge and cloud interfaces • VPN access restriction for external connections to edge and asset layers	Cybersecurity policies	Secured data in motion and at rest (edge, fog, cloud)

Table 13
Analysis of the self-diagnosis **Planning** requirement for OptiTwin.

Sub-requirement	Fulfilment score	OptiTwin resources	OptiTwin inputs	OptiTwin outputs
Alerting and accessibility	5 - Yes	On-screen alarm	Tool wear	Alarm indicating tool change
Fault assessment	1 - Not met (Severity: 2 - Medium)			
Policies adjustments	1 - Not met (Severity: 2 - Medium)	Planned for future implementation: • ISO 8688-1:1989 for maximum tool wear value • Manual adjustment of the maximum acceptable tool wear value to suit specific requirements • Continual learning techniques for fine-tuning of DL models as new data becomes available	Machining data	Adjusted policies and model weights
Adaptation/ Recovery plan	1 - Not met (Severity: 2 - Medium)			

This requirement achieves a **Core-tier $\mathcal{F}$ of 100%**, a **Secondary-tier $\mathcal{F}$ of 0%**, and a **global $\mathcal{F}$ of 60%**.

Regarding the **data management** requirement, OptiTwin demonstrates strong capabilities in multi-layered data acquisition and cloud-based analytics and storage. The system collects a wide range of machining signals, such as RPM, spindle position, torque, and power consumption. It uses a sampling frequency of 250 Hz and has storage capacities across edge and cloud layers. Cloud-hosted ML and DL models for tool wear prediction further extend the analytical capabilities of the system. They provide predictive insights about the condition of the tool. Robust security measures, including firewall protection, OAuth access tokens, TLS encryption, and user authentication, ensure the confidentiality and integrity of data during transfer and access. However, no fog-layer integration or semantic data modelling is implemented. The latter is an important functionality for the Secondary tier and is marked with a medium severity (2). Advanced features, such as local model hosting or LLM-powered automation, are not supported. This requirement achieves a **Core-tier $\mathcal{F}$ of 100%**, a **Secondary-tier $\mathcal{F}$ of 0%**, and an **Advanced-tier $\mathcal{F}$ of 0%**, resulting in a **global $\mathcal{F}$ of 43%**.

The requirement for **monitoring and analysis** highlights the capabilities of OptiTwin in edge-level preprocessing and prognostics. Noise reduction and feature engineering are performed using techniques such as moving average filtering, z-score normalisation, statistical descriptors, spectral features, and wavelet-based measures, producing enriched datasets suitable for downstream analysis. Prognostic functions are implemented through DL models for milling and decision trees for drilling, enabling effective forecasting of tool wear progression. Performance analysis is substantially met through dashboard-based KPI monitoring, which provides insights into energy efficiency, throughput, and machine state, although predictive KPI analysis is not yet implemented. In contrast, fault detection, root cause analysis, and LLM-based orchestration are not supported, reflecting a focus on tool condition monitoring rather than full self-diagnosis functionality. These gaps vary in importance, with fault detection rated as high severity (3), RCA as medium severity (2), and LLM-based orchestration as low severity (1).

Overall, the requirement achieves a **Core-tier $\mathcal{F}$ of 67%**, a **Secondary-tier $\mathcal{F}$ of 33%**, and an **Advanced-tier $\mathcal{F}$ of 0%**. The result is a **global $\mathcal{F}$ of 50%**.

The **planning** requirement is only partially addressed by OptiTwin. The system fulfils the alerting and accessibility sub-requirement by providing on-screen alarm notifications, ensuring that operators receive timely prompts for tool changes. However, deeper planning functionalities are not supported. Fault assessment is absent, preventing the generation of structured severity reports or knowledge-based insights. This gap is rated with medium severity (2) due to its relevance for informed maintenance decisions. Similarly, the lack of a policy adjustment mechanism limits the adaptability of the system to evolving operational contexts and standards, also rated with medium severity (2). Finally, the absence of an adaptation or recovery plan represents a high-severity gap (3), as no systematic corrective actions can be derived from diagnostic results. This requirement achieves a **Core-tier $\mathcal{F}$ of 100%**, while both the functional-tier and Advanced-tier $\mathcal{F}$s remain at 0%, resulting in a **global $\mathcal{F}$ of 25%**.

The **execution** requirement is partially supported by OptiTwin. The system fully meets the reporting sub-requirement through Grafana dashboards, which provide stakeholders with real-time insights into machining and energy consumption data. However, the absence of automated plan execution limits the capacity of the system to translate diagnostic results into concrete corrective actions. This gap is assigned a high severity score (3), reflecting its critical role in enabling autonomous response, even though it is classified as a secondary-tier requirement. Fault correction is likewise not implemented, requiring manual intervention for remedial measures. This limitation is rated with medium severity (2). In addition, advanced-tier functions such as action validation and action refinement are also absent, although their omission carries lower operational risk. This requirement achieves a **Secondary-tier $\mathcal{F}$ of 20%**, an **Advanced-tier $\mathcal{F}$ of 0%**, and a **global $\mathcal{F}$ of 20%**.

The **knowledge** requirement highlights the ability of OptiTwin to structure information. Historical diagnostics are fully supported through the systematic storage of process data in InfluxDB, user and role data in PostgreSQL, and experimental datasets in Dropbox and the edge layer. Policies are defined, drawing on ISO 8688-1:1989 thresholds for maximum tool wear and expert-defined condition ranges. Runtime information is also addressed, with dashboards providing statistical KPIs derived from machining parameters and cutting conditions. Fault trees and rules are minimally satisfied through basic alarm thresholds, rather than a structured rule or fault-tree engine, limiting interpretability. Advanced capabilities, such as semantic ontologies and dynamic knowledge refinement, are not present. Accordingly, the requirement achieves a **Core-tier $\mathcal{F}$ of 75%** and **Secondary- and Advanced-tier $\mathcal{F}$s of 0%**. This results in a **global $\mathcal{F}$ of 50%**.

Out of the 33 sub-requirements, OptiTwin fulfils 14, resulting in an **overall $\mathcal{F}$ of 42.42%**. When disaggregated by tiers, the system demonstrates strong performance at the **Core tier** with 85.71% fulfilment (12 out of 14 sub-requirements). On the other hand, coverage is substantially lower at the **Secondary tier** (12.5%, 2 out of 16) and entirely absent at the **Advanced tier** (0 out of 7). These results classify OptiTwin as being *Core functional*. It should be noted that certain sub-requirements, such as fault detection/identification (R3), adaptation/recovery plans (R4), action validation (R5) and action refinement (R5), are distributed across tiers. Hence, the tiered sums do not directly match the global count of fulfilled requirements. This distribution underscores the maturity of OptiTwin in core diagnostic functions. Furthermore, it highlights the need for further development of higher-tier capabilities. A global $\mathcal{F}$ below 50% is consistent with prototype-level technology readiness, where challenges such as legacy integration, safety validation, and advanced automation commonly delay comprehensive fulfilment.

It is worth mentioning that this case study does not discuss the specifics of implementation, as it should be preceded by an in-depth feasibility study and an evaluation period to assess system performance. Before deployment, a comprehensive analysis must be performed to validate the efficiency of the system.

5.3. Expected outcomes

The evaluation positions OptiTwin within the readiness matrix of the proposed methodology. With Core $\mathcal{A} = 88.89\%$, the system achieves **Core coverage**, meeting the threshold of at least 85%. Furthermore, with Core $\mathcal{F} = 85.71\%$, the threshold of at least 75% is satisfied. Therefore, the system is also **Core functional**. Together, these results place OptiTwin in the **Baseline ready** category of the readiness matrix. This indicates that the system provides a solid and reliable foundation for self-diagnosis. However, limited fulfilment of Secondary and Advanced tiers highlights clear improvement potential, particularly in planning, execution, and advanced knowledge refinement, which are necessary for broader autonomy and resilience.

6. Discussion

This study presented a self-diagnosis guideline for CPS, aimed at supporting production performance analysis, monitoring, and improvement. To illustrate how the guideline can be applied in practice, a case study protocol was proposed as part of the research methodology. The OptiTwin system was used as an illustrative example to demonstrate the application of the protocol. OptiTwin integrates a cloud-based TCM service to optimise machining processes. The assessment highlights how the protocol can be applied to an existing self-diagnosis solution to determine its readiness level. The correlation between the self-diagnosis service architecture and requirements proposed in this research and the case study forms the basis for the subsequent points of discussion.

Comparing the mapping from this work with the R&D challenges identified in the CIRP entry on CPS and cyber–physical production system (CPPS) [62] shows a correspondence between the proposed requirements and the broader research agenda for cyber–physical production systems. This highlights the importance and the potential of developing an architecture specifically targeting self-diagnosis that is also extendable to other advanced services. Furthermore, the requirements touch on core challenges in ICT, such as managing time in distributed systems, integrating continuous and computational dynamical systems models, aligning with emerging standards, and maintaining security across hardware, software and operational layers.

From the CPPS research perspective, R1–R3 address the challenges of advanced connectivity, intelligent data management, and continuous monitoring. R4 and R5 extend the alignment by operationalising planning and execution capabilities, reflecting the call for adaptive systems and robust scheduling. Finally, R6 formalises the role of structured knowledge and reasoning, thereby contributing to the human–machine perspective.

This alignment suggests that the proposed architecture serves as an initial response to both immediate implementation needs and longer-term research objectives in CPS and CPPS. The convergence of these agendas puts emphasis on the potential of the proposed requirements, while also raising expectations for their applicability. Being able to perform this transfer in industrial settings remains an open claim that will require targeted empirical verification.

6.1. Case-study analysis

The OptiTwin case study provided a quantitative assessment of the proposed guideline, revealing valuable insights into its practical applicability and areas for refinement. The case study surpassed the Core $\mathcal{A}$ target (88.89% versus 85%). However, the Secondary and Advanced architectural tiers remained largely unimplemented (0%), resulting in a global $\mathcal{A}$ of 47.06%. This positions OptiTwin as having *Core coverage* within the proposed readiness matrix. Regarding functionality, the system demonstrated high compliance with the Core tier, achieving an 85.71% $\mathcal{F}$, satisfying the 75% threshold for Core functionality. Consequently, OptiTwin is classified as *Core functional*. However, the Secondary tier showed a significantly lower $\mathcal{F}$ of 12.5%, and the

Advanced tier was entirely unfulfilled (0%). The overall global $\mathcal{F}$ for OptiTwin stood at 42.42%. The results collectively place OptiTwin in the *Baseline ready* category of the readiness matrix (Table 1). This indicates that OptiTwin provides a solid and reliable foundation for self-diagnosis, although its limited fulfilment in the Secondary and Advanced tiers highlights clear potential for improvement.

The tiered assessment underscores that full compliance with every sub-requirement is neither necessary nor realistic in an initial prototype. Instead, deficiencies should be evaluated against the specific operational needs and priorities of the target environment. Achieving Core-tier coverage in a live, production-grade demonstrator, such as OptiTwin, still underscores the practical viability of the proposed guideline. It also sets a concrete roadmap for closing remaining gaps in self-diagnostic industrial systems. Prioritising high-severity sub-requirements, such as automated recovery and fault assessment, will enhance system robustness and reliability in future iterations.

Balancing anticipated results, current and future business needs, and the investment required for implementation is essential. For instance, implementing advanced LLM for fault assessment, policy adjustment, and adaptation may require a higher investment in computational power, potentially requiring cloud-based solutions or hardware upgrades. Finding the optimal balance ensures that the implemented self-diagnosis service delivers added value without exceeding budgets and straining existing infrastructure capabilities.

Due to its modular structure, the TCM service can easily expand to include more assets or improve its capabilities by integrating its outcomes with other self-contained services. As indicated in the architectural diagram, the machining centre, CNC, and sensors are treated as a single asset instance. Even with the inclusion of additional machines carrying out similar machining operations, the service components in the edge and cloud layers remain consistent.

The OptiTwin case study served as a valuable demonstrator of the practical effectiveness of the proposed self-diagnosis guideline, both in terms of architecture and requirements. The case study confirmed that a self-diagnosis service complying with the proposed guideline can function as a self-contained service that supports process improvement. The architecture provides a blueprint of essential components and their interactions, while the requirements guide robust and effective service implementation. Without such a foundation, development efforts risk producing services that lack the structure and functionality necessary for meaningful process improvement. In contrast, a well-defined framework accelerates the adoption of CPS in industrial settings.

6.2. Guideline generalisability

The application of OptiTwin as an exercise to validate a use case according to an architecture and its requirements for self-diagnosis services revealed additional insights:

- The results of the case study demonstrator confirm the value of introducing a tiered classification scheme (Core, Secondary, and Advanced) to the guideline. This scheme provides a structured pathway for incremental implementation, allowing practitioners to focus initial development on core functionalities (e.g., real-time data acquisition, basic fault detection) and progressively add more advanced features (e.g., automated recovery planning, advanced fault assessment, online policy refinement). This phased strategy accommodates system complexity and available resources. It also enables stakeholders to observe tangible benefits at each stage, guiding iterative refinements in line with evolving requirements.
- The modularity embedded in the architecture facilitates seamless integration and enhances scalability of self-diagnosis services. The modular design allows for the independent development and deployment of specific components. This ensures that the implementation process can be tailored to the unique requirements of varied industrial settings. This scalability is particularly advantageous in dynamic manufacturing environments where the scale and complexity of operations may vary. This not only streamlines the initial implementation of self-diagnosis services but also positions the system for ongoing adaptability.
- The comprehensive incorporation of LLMs improves the intelligence and adaptive capabilities of the system. This integration marks a shift towards the FD 2.0 era. It enables LLMs to process unstructured and noisy data, integrate heterogeneous information, and deliver context-aware insights. They can automate data preprocessing, infer missing values, guide model selection, act as controllers for prediction tasks, refine causal inferences for RCA, provide decision support, and generate knowledge-driven contingency plans. Furthermore, they play a crucial role in action validation, refinement, and policy evolution for self-correction. Their integration can also aid in reasoning with unstructured data from older, less digitised legacy systems, bridging interoperability gaps.
- The architecture explicitly maps its layers to the RAMI 4.0 hierarchy levels and layers. The mapping demonstrates alignment with a key Industry 4.0 reference architecture. The adoption of AASs in the asset layer further facilitates semantic interoperability and standardised data access following RAMI 4.0 principles. This directly addresses legacy system compatibility. Furthermore, the structured KB allows for the formalisation and reuse of knowledge from various sources, including legacy systems.
- The methodology presented in this study provides the steps to map other systems to the architecture and requirements of self-diagnosis services. This not only facilitates a streamlined integration process but also enhances the interoperability of diverse systems within an overarching framework. In addition, this systematic approach contributes to the scalability and universality of the self-diagnosis service, potentially extending its application across various industrial contexts.

The insights derived from the OptiTwin case study further established the value of the proposed architecture and requirements. Unlike existing research focusing on specific aspects like data collection or fault detection, this work provided a holistic guideline that considers all stages, from initial data acquisition to knowledge management and process improvement. This comprehensive approach can ensure a more systematic and effective implementation of self-diagnosis functionalities within industrial environments. By detailing aspects from connectivity to knowledge management, the architecture and requirements of self-diagnosis services serve as a valuable roadmap for researchers, engineers, and practitioners in the field, facilitating the adoption of CPS in industrial environments and contributing to the advancement of production improvement.

In this work, LLM functionalities are positioned as optional or secondary enhancements to the proposed architecture and requirements. They are intended to complement, rather than replace, established diagnostic and reasoning techniques, enabling more flexible and interpretable self-diagnosis in advanced implementations. As such, their absence in a given system, such as in the OptiTwin case study, does not preclude meeting the core requirements defined in this study, but may represent an opportunity for future capability expansion.

6.3. Alignment with industry 5.0

The proposed guideline is intrinsically aligned with the emerging Industry 5.0 paradigm, which prioritises human-centricity, resiliency, and sustainability as core values [63]. This guideline supports this shift towards a value-oriented manufacturing system through its comprehensive set of requirements (R1–R6) and the adoption of advanced capabilities, such as LLMs.

Resiliency is fundamentally addressed by the self-diagnosis behaviours that underpin the architecture. The Planning (R4) and Executing (R5) requirements focus on developing and implementing contingency plans, enabling fault correction and recovery strategies. The capacity of self-diagnosis services for autonomous adjustment supports robust system behaviour and resilience against unexpected disturbances.

Sustainability is supported by the core function of the system, i.e., minimising production downtime and optimising resource use. The Monitoring and Analysis requirement (R3) stipulates the establishment of KPIs, which include monitoring energy efficiency and throughput, directly contributing to resource optimisation and long-term competitiveness.

Human-centricity is integrated through the use of structured KBs and explainable components (R6). The Executing requirement (R5) mandates the provision of operator guidance and alerts via secure dashboards and HMIs. The integration of LLMs in the Advanced tier further enhances this human–machine collaboration. LLMs act as autonomous co-pilots by interpreting complex diagnostic results and providing actionable recommendations. They can generate these insights in natural language responses, thereby making the underlying predictive models and causal inferences (R3) transparent and explainable to human operators. This aims to generate trust in users and effective decision-making in the self-diagnosis service. This integration ensures that autonomous capabilities serve to augment human roles rather than replace them entirely, aligning precisely with the human-centric principles of Industry 5.0.

6.4. Limitations

While the proposed guideline and architecture offer a structured foundation for implementing self-diagnosis services in industrial CPS, several limitations must be acknowledged. First, the proposed guideline and architecture presume the availability of high-quality, continuous data streams from multiple heterogeneous sources. In practice, sensor failures, network disruptions, or legacy system limitations may compromise data integrity and disrupt real-time diagnostic functions.

Second, seamless integration relies on interoperability among devices, platforms, and data formats. While the proposed design adopts open standards (e.g., REST APIs and MQTT), many industrial environments remain fragmented or dependent on proprietary protocols. This necessitates custom adaptations that may reduce the ease of deployment, especially with legacy systems. However, the KB-centric approach and its capacity to leverage LLMs for processing unstructured data from diverse sources provide a foundational pathway for systematically incorporating information from these older systems, mitigating the impact of such fragmentation.

Third, the application of the guideline is demonstrated through a single case study focused on TCM in a CNC machining context. However, broader applicability remains to be established. Requirements, such as self-explanatory results, may carry different importance in other industrial domains, including assembly lines or continuous-process industries. Future evaluations across diverse contexts are therefore necessary to consolidate the readiness matrix with additional empirical mappings.

Finally, while the modular, multi-layered structure of the architecture supports vertical scaling (from SMEs to large-scale facilities) and horizontal reuse across domains, broader standardisation and further empirical testing are required. In particular, phased implementation templates and integration guidelines should be developed to facilitate adaptation under varying levels of technical maturity and resource availability.

The lessons learned from OptiTwin highlight several key aspects of the proposed architecture. Tiered classification proved useful. A layered design offered flexibility for integrating legacy systems. Structured knowledge bases supported explainability and built operator trust. However, these strengths also revealed challenges. The modularity that enables incremental adoption places high demands on organisational readiness. It also requires strong cross-disciplinary collaboration, particularly when implementing advanced features such as LLM-driven reasoning.

Despite these limitations, the proposed guideline addresses a critical gap in the literature. It provides formalised requirements and a robust reference architecture for self-diagnosis services in CPS. The proposed modular composition and service-oriented design allow tailoring of functionalities to sector-specific constraints and regulatory environments. This invites adaptation without redesigning the core architecture.

7. Conclusions

This paper presents a novel guideline for implementing self-diagnosis services in industrial settings. It combines a multi-layered architecture with a comprehensive set of functional requirements. Informed by an extensive review of existing research and grounded in practical experience, the guideline offers practitioners a clear road map for designing, deploying, and maintaining self-diagnosis services in real-world applications.

A central guideline component is the proposed architecture, which adopts a multi-layered design comprising asset, edge, fog, and cloud layers. This structure supports seamless communication from sensor-level data acquisition to high-level analytics, promoting scalability, adaptability, and autonomous decision-making. In addition, the architecture demonstrates formal alignment with the RAMI 4.0 reference model, making it suitable for broader applicability to other CPSs. This aligns with previous work on enabling interoperability in autonomous industrial environments [30].

The OptiTwin project, a digital twin platform for tool wear diagnosis, was used as a demonstrator to apply the proposed guideline. The system was mapped against both architectural and functional requirements to illustrate how readiness can be quantified. Results showed a Core-tier architectural completeness of 88.89% and a global score of 47.06%. Functionally, OptiTwin reached 85.71% at the Core tier and 42.42% overall, classifying it as *Baseline ready* within the proposed readiness matrix.

The results of the case study highlight a solid foundation for self-diagnosis while also revealing gaps in higher tiers (Secondary 12.5% and Advanced 0%). This demonstrates the value of tiered classification. Core functionalities, such as data acquisition and fault detection, can be prioritised first. Thereafter, advanced features, including recovery planning and LLM-driven reasoning, can be developed incrementally.

This work extends beyond technical specifications, offering a practical road map and methodology for implementing self-diagnosis services in industrial settings. This approach, which emphasises modularity, scalability, and incremental functionality, caters to researchers, engineers, and practitioners. By facilitating phased implementations, the methodology allows for early benefits and iterative improvements, adapting to evolving needs and technological advancements. Furthermore, the methodology enables the seamless integration of diverse systems within a unified framework, enhancing interoperability and potentially expanding its application across broader industrial landscapes.

Future work will focus on refining and extending the proposed guideline for self-diagnosis services based on other real-world use cases. Priorities include improving the clarity and granularity of individual requirements, particularly those identified as unmet with medium or high severity. Additionally, the guideline will be reviewed to enhance its generalisability across diverse industrial contexts, incorporating feedback from additional case studies. These improvements aim to support more accurate assessments of implementation readiness, promote consistent interpretation of requirements, and facilitate the broader adoption of self-diagnosis services in industrial CPS.

Furthermore, given the increasing role of LLMs in self-diagnosis, future research will address their limitations, such as "hallucination" and sensitivity to noisy industrial data, by developing domain-specific LLMs and dedicated datasets. Performance metrics to assess the reliability and cost-effectiveness of LLM-enabled modules will be incorporated. Key indicators include hallucination rate, RCA accuracy, and system downtime reduction. Additionally, practical deployment guidelines will specify edge-compute thresholds suitable for SMEs (e.g., minimum CPU/GPU configurations) to ensure that resource demands remain proportionate to operational constraints. Such metrics will allow benchmarking of self-diagnosis services under realistic industrial conditions.

CRediT authorship contribution statement

José Joaquín Peralta Abadía: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Fabio Marco Monetti:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Conceptualization. **Sylvia Nathaly Rea Minango:** Writing – original draft, Visualization, Validation, Methodology, Investigation, Conceptualization. **Angela Carrera-Rivera:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Conceptualization. **Miriam Ugarte Querejeta:** Writing – original draft, Visualization, Validation, Methodology, Investigation, Conceptualization. **Mikel Cuesta Zabaljauregui:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition. **Felix Larrinaga Barrenechea:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition. **Miren Illarramendi Rezabal:** Writing – review & editing, Resources, Project administration, Funding acquisition. **Antonio Maffei:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 814078 and by the Department of Education, Universities and Research of the Basque Government under the projects Ikerketa Taldeak (Grupo de Ingeniería de Software y Sistemas IT1519-22 and Grupo de investigación de Mecanizado de Alto Rendimiento IT1443-22).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jmsy.2025.11.010.

References

[1] Colombo AW, Karnouskos S, Kaynak O, Shi Y, Yin S. Industrial Cyberphysical Systems: A Backbone of the Fourth Industrial Revolution. IEEE Ind Electron Mag 2017;11(1):6–16. http://dx.doi.org/10.1109/MIE.2017.2648857.

[2] Ferrer BR, Mohammed WM, Martinez Lastra JL, Villalonga A, Beruvides G, Castano F, Haber RE. IEEE 16th International Conference on Industrial Informatics (INDIN). IEEE, Porto 2018;2018:792–9. http://dx.doi.org/10.1109/INDIN.2018.8472061.

[3] Lozano CV, Vijayan KK. Literature review on cyber physical systems design. In: Procedia manufacturing, 45 of CLF. Graz, AT: Elsevier; 2020, p. 295–300. http://dx.doi.org/10.1016/j.promfg.2020.04.020.

[4] Al-Ali R, Bulej L, Kofroň J, Bureš T. A guide to design uncertainty-aware self-adaptive components in Cyber–Physical Systems. Future Gener Comput Syst 2022;128:466–89. http://dx.doi.org/10.1016/j.future.2021.10.027.

[5] Xu X. From cloud computing to cloud manufacturing. Robot Comput-Integr Manuf 2012;28(1):75–86. http://dx.doi.org/10.1016/j.rcim.2011.07.002.

[6] Mell P, Grance T, et al. The NIST definition of cloud computing. National institute of science and technology; 2011, special publication.

[7] Mell P, Grance T. Perspectives on cloud computing and standards. National institute of science and technology; 2009, special publication.

[8] Kephart JO, Chess DM. The vision of autonomic computing. Computer 2003;36(1):41–50. http://dx.doi.org/10.1109/MC.2003.1160055.

[9] Salehie M, Tahvildari L. Self-adaptive software: Landscape and research challenges. ACM Trans Auton Adapt Syst 2009;4(2):1–42. http://dx.doi.org/10.1145/1516533.1516538.

[10] Sardinha LF, Sousa Á, Leite E, Carvalho A. Self-diagnosis tool for time management: Proposal and validation. Int J Adv Appl Sci 2023;10(5):183–94.

[11] Zhong RY, Xu C, Chen C, Huang GQ. Big Data Analytics for Physical Internet-based intelligent manufacturing shop floors. Int J Prod Res 2017;55(9):2610–21. http://dx.doi.org/10.1080/00207543.2015.1086037.

[12] Lee J, Bagheri B, Kao H-A. A cyber–physical systems architecture for industry 4.0-based manufacturing systems. Manuf Lett 2015;3:18–23. http://dx.doi.org/10.1016/j.mfglet.2014.12.001.

[13] Caggiano A. Cloud-based manufacturing process monitoring for smart diagnosis services. Int J Comput Integr Manuf 2018;31(7):612–23. http://dx.doi.org/10.1080/0951192X.2018.1425552.

[14] Kusiak A. Smart manufacturing must embrace big data. Nature 2017;544(7648):23–5. http://dx.doi.org/10.1038/544023a.

[15] Mulet Alberola JA, Carrera-Rivera A, Ugarte Querejeta M, Ngoc Nguyen H, Estrada-Jimenez LA, Patil T, Onori M. DiManD work package 3.1b – full analysis and final set of requirements. 2021.

[16] Estrada-Jimenez LA, Mulet Alberola JA, Carrera-Rivera A, Ugarte Querejeta M, Ngoc Nguyen H, Patil T. DiManD work package 3.2 – cyber physical systems architecture. 2022.

[17] Peralta Abadia JJ, Larrinaga Barrenechea F, Cuesta Zabaljauregui M, Badiola Aiestaran X, Duo Zubiaurre A, Olalde Mendia G, Pérez Lázare T. OptiTwin: Data-driven machining process optimization platform for SMEs. In: 29th IEEE International Conference on Emerging Technologies and Factory Automation. Padova, Italy: IEEE; 2024, http://dx.doi.org/10.1109/ETFA61755.2024.10711032.

[18] Maffei A. evolvable production systems: foundations for new business models, skolan för industriell teknik och management. Kungliga Tekniska högskolan; 2010.

[19] Lopes de Sousa Jabbour AB, Jabbour CJC, Godinho Filho M, Roubaud D. Industry 4.0 and the circular economy: a proposed research agenda and original roadmap for sustainable operations. Ann Oper Res 2018;270:273–86. http://dx.doi.org/10.1007/s10479-018-2772-8.

[20] Rahatulain A, Lawson H. Towards life cycle management of industrial manufacturing systems: A systems perspective. In: 9th annual IEEE international systems conference (sysCon). (13-16):BC, Canada: Vancouver; 2015, p. 2015.

[21] Van Brussel H, Wyns J, Valckenaers P, Bongaerts L, Peeters P. Reference architecture for holonic manufacturing systems. PROSA Comput. ind. 1998;37(3):255–74.

[22] Bi ZM, Lang SY, Shen W, Wang L. Reconfigurable manufacturing systems: the state of the art. Int J Prod Res 2008;46(4):967–92. http://dx.doi.org/10.1080/00207540600905646.

[23] Onori M, Lohse N, Barata J, Hanisch C. The IDEAS project: plug & produce at shop-floor level. Assem Autom 2012;32(2):124–34. http://dx.doi.org/10.1108/01445151211212280.

[24] Zhang Q, Xu C, Li J, Sun Y, Bao J, Zhang D. LLM-TSFD: An industrial time series human-in-the-loop fault diagnosis method based on a large language model. Expert Syst Appl 2025;264:125861. http://dx.doi.org/10.1016/j.eswa.2024.125861.

[25] Choudhury MD, Blincoe K, Dhupia JS. An overview of fault diagnosis of industrial machines operating under variable speeds. Acoustics Australia 2021;49(2):229–38. http://dx.doi.org/10.1007/s40857-021-00236-3.

[26] Frei R, McWilliam R, Derrick B, Purvis A, Tiwari A, Di Marzo Serugendo G. Self-healing and self-repairing technologies. The International Journal of Advanced Manufacturing Technology 2013;69(2013):1033–61. http://dx.doi.org/10.1007/s00170-013-5070-2.

[27] Leitao P, Karnouskos S, Ribeiro L, Lee J, Strasser T, Colombo AW. Smart agents in industrial cyber–physical systems. Proc IEEE 2016;104(5):1086–101. http://dx.doi.org/10.1109/JPROC.2016.2521931.

[28] Frei R. Self-organisation in evolvable assembly systems. 2010.

[29] Onori M, Semere D, Lindberg B. Evolvable systems: an approach to self-X production. Int J Comput Integr Manuf 2011;24(5):506–16. http://dx.doi.org/10.1080/0951192X.2011.566282.

[30] Mulet Alberola JA, Carrera-Rivera A, Ugarte Querejeta M, Ngoc Nguyen H, Estrada-Jimenez LA, Patil T, Onori M. Dimand work package 3.1a overall requirements and preliminary selection. 2021.

[31] Mühl G, Werner M, Jaeger MA, Herrmann K, Parzyjegla H. On the definitions of self-managing and self-organizing systems. In: Communication in Distributed Systems-15. ITG/GI Symposium. IEEE; 2007, p. 1–11.
[32] Shin S-J, Kim Y-M, Meilanitasari P. A holonic-based self-learning mechanism for energy-predictive planning in machining processes. Processes 2019;7(10):739.
[33] Christiansen L, Fay A, Opgenoorth B, Neidig J. Improved diagnosis by combining structural and process knowledge. In: ETFA2011. IEEE; 2011, p. 1–8. http://dx.doi.org/10.1109/ETFA.2011.6059056.
[34] Kalgren PW, Byington CS, Roemer MJ, Watson MJ, phm Defining. Defining phm, a lexical evolution of maintenance and logistics. In: 2006 IEEE autotestcon. IEEE 2006;2006:353–8. http://dx.doi.org/10.1109/AUTEST.2006.283685.
[35] Barata J, Ribeiro L, Onori M. 2007 IEEE international symposium on industrial electronics. In: Diagnosis on evolvable production systems. IEEE; 2007, p. 3221–6. http://dx.doi.org/10.1109/ISIE.2007.4375131.
[36] Fries TP. International conference on integration of knowledge intensive multi-agent systems. IEEE 2007;2007:168–73. http://dx.doi.org/10.1109/KIMAS.2007.369804.
[37] Jeong Y, Son S, Jeong E, Lee B. An integrated self-diagnosis system for an autonomous vehicle based on an iot gateway and deep learning. Appl Sci 2018;8(7):1164. http://dx.doi.org/10.3390/app8071164.
[38] Gatica CP, Boschmann A. Enabling self-diagnosis of automation devices through industrial analytics. 2018, Springer; 2018, p. 107–15. http://dx.doi.org/10.1007/978-3-662-58485-9_12.
[39] Long J, Chen Y, Yang Z, Huang Y, Li C. A novel self-training semi-supervised deep learning approach for machinery fault diagnosis. Int J Prod Res 2023;61(23):8238–51. http://dx.doi.org/10.1080/00207543.2022.2032860.
[40] Lee J, Ni J, Djurdjanovic D, Qiu H, Liao H. Intelligent prognostics tools and e-maintenance. Comput Ind 2006;57(6):476–89. http://dx.doi.org/10.1016/j.compind.2006.02.014.
[41] Yu R, Iung B, Panetto H. A multi-agents based e-maintenance system with case-based reasoning decision support. Eng Appl Artif Intell 2003;16(4):321–33. http://dx.doi.org/10.1016/S0952-1976(03)00079-4.
[42] Feldmann K, Göhringer J. Internet based diagnosis of assembly systems. CIRP Ann 2001;50(1):5–8. http://dx.doi.org/10.1016/S0007-8506(07)62058-7.
[43] Zhou S, Zhang J, Wang S. Fault diagnosis in industrial processes using principal component analysis and hidden markov model. In: Proceedings of the 2004 American control conference. vol. 6, IEEE; 2004, p. 5680–5. http://dx.doi.org/10.23919/ACC.2004.1384761.
[44] Ploix S, Gentil S, Lesecq S. Isolation decision for a multi-agent-based diagnostic system. In: IFAC proceedings. vol. 36, (5):2003, p. 465–70. http://dx.doi.org/10.1016/S1474-6670(17)36535-7.
[45] Pinto R, Reis J, Sousa V, Silva R, Gonçalves GM. Self-diagnosis and automatic configuration of smart components in advanced manufacturing systems. In: INTELLI 2015, the fourth international conference on intelligent systems and applications. 2015.
[46] Barata J, Ribeiro L, Onori M. Diagnosis on evolvable production systems. In: 2007 IEEE international symposium on industrial electronics. 2007, p. 3221–6. http://dx.doi.org/10.1109/ISIE.2007.4375131.
[47] Frei R, Ribeiro L, Barata J, Semere D. Evolvable assembly systems: Towards user friendly manufacturing. In: 2007 IEEE international symposium on assembly and manufacturing. 2007, p. 288–93. http://dx.doi.org/10.1109/ISAM.2007.4288487.
[48] Iglesia DGDL, Weyns D. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. ACM Trans Auton Adapt Syst 2015;10(3). http://dx.doi.org/10.1145/2724719.
[49] Schweichhart K. Reference architectural model industrie 4.0 (rami 4.0). Tech. rep., Plattform Industrie 4.0.
[50] Qi Q, Tao F. A smart manufacturing service system based on edge computing, fog computing, and cloud computing. IEEE Access 2019;7:86769–77. http://dx.doi.org/10.1109/ACCESS.2019.2923610.
[51] Wu D, Terpenny J, Zhang L, Gao R, Kurfess T. Fog-enabled architecture for data-driven cyber-manufacturing systems. In: Materials; biomanufacturing; properties, applications and systems; sustainable manufacturing. vol. 2, American Society of Mechanical Engineers; 2016, http://dx.doi.org/10.1115/MSEC2016-8559, V002T04A032.
[52] Skala K, Davidovic D, Afgan E, Sovic I, Sojat Z. Scalable distributed computing hierarchy: Cloud, fog and dew computing. Open J Cloud Comput (OJCC) 2015;1(2):16–24. http://dx.doi.org/10.19210/1002.2.1.16.
[53] for Standardization International Organization. Automation systems and integration – digital twin framework for manufacturing. 2021, https://www.iso.org/standard/75066.html, iSO 23247.
[54] Melo V, Barbosa J, Mota G, de La Prieta F, Leitao P. Design of an iso 23247 compliant digital twin for an automotive assembly line. In: 2024 IEEE 7th international conference on industrial cyber-physical systems. ICPS, IEEE; 2024, p. 1–6. http://dx.doi.org/10.1109/ICPS59941.2024.10640052.
[55] Boonmee A, Wongsuwan K, Sukjai P. Consultation on industrial machine faults with large language models. 2024, http://dx.doi.org/10.48550/arXiv.2410.03223, arXiv preprint arXiv:2410.03223.
[56] Baldea M, Georgiou AT, Gopaluni B, Mercangöz M, Pantelides CC, Sheth K, Zavala VM, Georgakis C. From automated to autonomous process operations. Comput Chem Eng 2025;196:109064. http://dx.doi.org/10.1016/j.compchemeng.2025.109064.
[57] Shridhar K, Stolfo A, Sachan M. Distilling reasoning capabilities into smaller language models. 2022, http://dx.doi.org/10.48550/arXiv.2212.00193, arXiv preprint arXiv:2212.00193.
[58] Vyas J, Mercangöz M. Autonomous industrial control using an agentic framework with large language models. 2024, http://dx.doi.org/10.48550/arXiv.2411.05904, arXiv preprint arXiv:2411.05904.
[59] Ochoa W, Larrinaga F, Perez A, Cuenca J. Enhancing flexibility in industry 4.0 workflows: A context-aware component for dynamic service orchestration. Procedia Comput Sci 2024;232:1503–12. http://dx.doi.org/10.1016/j.procs.2024.01.148.
[60] Yin RK. Case study research and applications. CA: Sage Thousand Oaks; 2018, p. 6.
[61] Peralta Abadía J-J, Cuesta Zabaljauregui M, Larrinaga Barrenechea F. Mondragon unibertsitatea face-milling dataset for smart tool condition monitoring. Sci Data 2025;12(855). http://dx.doi.org/10.1038/s41597-025-05168-5.
[62] Monostori L. Cyber-physical systems. In: CIRP encyclopedia of production engineering. Springer; 2018, p. 1–8.
[63] Leng J, Sha W, Wang B, Zheng P, Zhuang C, Liu Q, Wuest T, Mourtzis D, Wang L. Industry 5.0: Prospect and retrospect. J Manuf Syst 2022;65:279–95. http://dx.doi.org/10.1016/j.jmsy.2022.09.017.