CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Large Language Model Empowered Deep Reinforcement Learning to Support Prescriptive Maintenance

Master's Thesis in Software Engineering and Technology

Jinxia Luo

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's Thesis 2025

Large Language Model Empowered Deep Reinforcement Learning to Support Prescriptive Maintenance

Jinxia Luo

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Large Language Model Empowered Deep Reinforcement Learning to Support Prescriptive Maintenance
Jinxia Luo

Supervisor: Siyuan Chen, Department of Industrial and Materials Science
Examiner: Ebru Turanoglu Bekar, Department of Industrial and Materials Science

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 736582601

Typeset in LaTeX
Gothenburg, Sweden 2025

Jinxia Luo
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

To enhance the reliability and efficiency of complex industrial systems, it is critical to adopt approaches that transcend traditional reactive and preventive maintenance methods. Prescriptive maintenance, by leveraging advanced artificial intelligence (AI) techniques to recommend and optimize specific maintenance actions before failures occur, has emerged as a vital strategy for modern industries to reduce downtime and operational costs. In this context, Deep Reinforcement Learning (DRL) has demonstrated significant potential in prescriptive maintenance tasks, enabling autonomous optimization of maintenance strategies through self-learning and adaptive decision-making. However, in real-world factory maintenance scenarios, DRL models frequently face challenges like cold-start problems, imprecise judgments, and a lack of flexibility in complicated dynamic contexts. These challenges prevent them from being widely used in complicated and highly dependable systems.

To address these limitations, this paper proposes an innovative smart maintenance framework that effectively combines cutting-edge Large Language Models (LLMs) with DRL approaches. The framework leverages the powerful capabilities of LLMs in domain knowledge comprehension, semantic reasoning, and knowledge transfer to convert complicated, real-time, structured maintenance data into high-dimensional state representations that DRL models can easily use. The LLMs provide the DRL agents with rich previous knowledge and decision support by automatically extracting important attributes and identifying possible failure patterns.

We conducted a case study at a lab-scale drone manufacturing plant, and the results showed that the proposed framework significantly improved the accuracy of smart maintenance decisions. Compared with previous studies[1], the average reward of traditional deep reinforcement learning algorithms was 0.65, while after introducing large language models, the average reward steadily increased to 0.82. In addition, fluctuations in the early stages of learning were markedly reduced, resulting in a smoother overall training process. Furthermore, we designed multiple sets of cross-experiments with different DRL algorithms and LLM models, all of which outperformed those in previous studies[1], further validating the effectiveness and generalizability of the proposed framework.

The LLM-DRL integrated smart maintenance framework presented in this study not only refines and improves smart maintenance decision-making but also aligns with Industry 4.0's broader objectives of operational efficiency, flexibility, and sustainability. Our research provides new theoretical and technological perspectives for advancing maintenance intelligence, enabling greater autonomy, improved generalization, and broader industrial applications.

Keywords: large language models, deep reinforcement learning, digital twin, prescriptive maintenance, decision-making, state representation

Acknowledgements

I would like to thank my supervisor, Siyuan Chen, for his essential guidance, patience, and expert advice during the project. His encouragement and support have been crucial to my development. I am particularly grateful to my examiner, Ebru Turanoglu Bekar, for her insightful remarks and ideas that contributed significantly to the project's success.

Jinxia Luo, Gothenburg, June 2025

Contents

List of Figures

1

Introduction

1.1 Problem Description

As Industry 4.0 progresses, the maintenance paradigm in manufacturing is increasingly evolving toward smart solutions. In this context, prescriptive maintenance has become an important area of exploration in modern manufacturing. Unlike traditional preventive or predictive maintenance, prescriptive maintenance not only predicts potential failures but also provides specific recommendations and maintenance actions to prevent them. This proactive approach is crucial for minimizing downtime, reducing operational costs, and extending equipment lifespan[3].

To fully harness the potential of prescriptive maintenance, there is a growing demand for AI-driven digital twin solutions[4]. These systems integrate artificial intelligence and machine learning to transform digital twins from static models into dynamic, self-optimizing frameworks. AI techniques enable digital twins to learn from observed behavior and historical data, dramatically improving data analysis efficiency and prediction accuracy[5]. These intelligent systems can process large volumes of real-time data, learn from it, and autonomously generate actionable maintenance strategies based on the current state of equipment and production environments. By integrating AI with digital twins, manufacturers can achieve a closed-loop maintenance system that continuously adjusts and improves maintenance decisions, ensuring optimal production efficiency in complex industrial settings[1].

Deep Reinforcement Learning (DRL) serves as a pivotal AI methodology within digital twin ecosystems, enabling dynamic optimization of maintenance workflows. DRL combines reinforcement learning—where agents learn behaviors through iterative trial-and-error interactions with dynamic environments—with deep neural networks to handle complex state spaces and learn policies for sequential decision tasks. Through continuous interaction with digital twin simulations, DRL agents leverage adaptive reward mechanisms to prioritize maintenance tasks, demonstrating significant improvements over traditional empirical decision-making approaches[1]. Furthermore, DRL models can accurately predict the remaining useful life of critical components, demonstrating superior performance compared to other state-of-the-art methods[6]. The integration of DRL with digital twins creates a self-improving system where maintenance strategies continuously evolve based on real-world equip-

ment operational patterns, providing dynamic optimization capabilities for industrial intelligence strategies.

Meanwhile, AI technology frameworks are undergoing revolutionary advancements, particularly marked by breakthroughs in the field of generative AI. The development of LLMs, like Google's BERT and OpenAI's GPT series, represents a significant advancement in the field of natural language processing (NLP). With the aid of increasingly potent computational resources and sophisticated algorithms, LLMs have proven to be exceptionally proficient at contextual comprehension, question answering, and content generation[7]. With their strong logical reasoning, knowledge transfer, and text-processing abilities, LLMs provide new opportunities for optimizing processes, enhancing efficiency, and driving innovation in manufacturing, greatly facilitating the intelligent transformation of the sector[8]. At present, LLMs have been widely integrated into Industrial Internet of Things (IIoT) and automation systems to achieve knowledge-driven automation and intelligence in all aspects of production, operations and management, bringing higher efficiency and innovation to manufacturing[9].

However, AI-driven digital twin architectures still face many significant challenges in their application to the manufacturing industry[10]. This is because the manufacturing environment is often highly specialized, complex, and constantly changing, which places higher demands on the model's comprehension, self-learning, and adaptive optimization capabilities[11]. Current solutions often struggle to efficiently integrate real-time operational data with advanced AI technologies (e.g., LLMs, DRL, etc.) within a coherent and scalable framework that enables seamless collaboration between physical and virtual systems[10]. Although LLMs show great potential in semantic understanding, knowledge reasoning, and multimodal information integration, how to efficiently and effectively integrate these models with digital twins to fully utilise the potential of digital twins in intelligent monitoring, predictive maintenance, and complex decision-making support is still an important topic that needs to be addressed[12, 13].

1.2 Purpose and Aim

This study focuses on the critical challenge of efficiently leveraging real-time operational data and professional expertise to achieve prescriptive maintenance in complex industrial environments. Traditional methods combining DRL and digital twins struggle to dynamically integrate structured sensor data with unstructured expert knowledge for formulating reliable maintenance strategies. Furthermore, such systems frequently face cold-start problems and exhibit low learning efficiency. This research aims to bridge the gap between data-driven analytics and human-centric decision-making. The purpose is to construct an adaptive system capable of continuously transforming multimodal inputs, including real-time sensor data, maintenance logs, and expert recommendations, into optimized prescriptive maintenance actions, thereby overcoming existing limitations.

The core objective of this study is to propose and validate an LLM-driven DRL-digital twin decision framework to enhance prescriptive maintenance capabilities. Building upon conventional DRL-digital twin systems, this framework leverages the natural language processing capabilities of LLMs to convert unstructured expert knowledge, maintenance manuals, and historical reports into actionable insights for DRL agents, enabling context-aware decision-making. Through a case study conducted at a lab-scale drone manufacturing plant, the effectiveness of this approach is demonstrated. The framework achieves superior performance compared to traditional DRL-digital twin systems, delivering reliable intelligent decisions and establishing a novel methodology for prescriptive maintenance practices.

1.3 Research Question

This thesis addresses the following key research questions:

RQ 1: How can LLMs convert real-time maintenance reports into valuable state representations suitable for DRL algorithms?

This question explores how to effectively leverage state-of-the-art LLMs (such as LLaMA, DeepSeek, or Phi) to process streaming structured text data (e.g., maintenance reports containing human feedback) and transform them into interpretable numerical representations. Key considerations include how to design high-quality prompt templates to effectively guide LLMs in analyzing report content; how to design state variables that reflect the complex conditions of equipment; and how to feed the generated results into the DRL agents.

RQ 2: How do different DRL models perform when trained with high-dimensional state representations?

This question investigates the training performance and adaptability of mainstream DRL models (such as DDQN, SAC, and PPO) when handling highly condensed state representation data. Key considerations include analyzing each model's capacity to extract important features from the state representations; comparing the convergence speed and sample efficiency of various models under high-dimensional inputs; and evaluating decision-making accuracy when influenced by information generated from different LLMs.

1.4 Scope and Delimitation

This study focuses on the application of LLMs and DRL models in prescriptive maintenance for manufacturing systems. We conducted experiments in a drone assembly laboratory and proposed an intelligent decision-making strategy for determining the maintenance priorities of multiple assembly machines along the production line. The digital twin model was implemented using factory simulation software, creating a virtual environment based on real snapshots from the drone assembly lab. During

the experiments, the simulation software generated virtual equipment fault reports in real time. The LLM extracted the latest dozens of entries from these reports for processing, which were then used to enhance the DRL model. The generated report information included equipment fault data automatically produced by the digital twin model, combined with pre-collected fault resolution information and human feedback. By integrating these diverse data sources with intelligent algorithms, this study aims to further explore the topic of intelligent maintenance prioritization within the field of prescriptive maintenance.

This study was conducted solely in a virtual simulation environment of a drone assembly laboratory and does not include data from real factories or other types of manufacturing systems. The experimental subjects are limited to certain assembly devices on the production line and do not cover all equipment or other stages of the production process. All LLMs used in this study are open-source models from Huggingface (https://huggingface.co/models), and proprietary models such as ChatGPT and Gemini are not included. This study focuses only on smart maintenance decision-making in manufacturing and does not cover other aspects such as economic benefits, environmental protection, or ethical issues.

1.5 Thesis Outline

The structure of this report is as follows: Section 2 presents the theoretical background, introducing the main technologies used in this study and their underlying principles. It also discusses the relevant literature that supports the thesis. Section 3 provides a detailed description of the design of the algorithms used in this study, along with the experimental methods adopted. Section 4 presents the research findings obtained during the study and offers a discussion of these results. Finally, the report concludes with a summary of the experimental outcomes and an outlook on future research directions.

2

Theoretical Background

2.1 Maintenance in Manufacturing

With the evolution of industry from the early days of Industry 1.0 to today's era of automation and data-intensive Industry 4.0, maintenance practices have also advanced significantly. Starting from reactive maintenance, the field has gradually progressed to preventive maintenance, evolved into predictive maintenance, and more recently, has seen the growing application of prescriptive maintenance. Figure 4.3 shows the whole process of maintenance development.

2.1.1 Reactive Maintenance

In the early stages of industrial development, maintenance was primarily passive, a practice known as reactive maintenance[14]. Its central feature is that maintenance action is taken after a problem has arisen or the equipment has ceased to operate, with the aim of restoring the equipment to its normal operating condition as soon as possible. Such maintenance does not require the investment of maintenance resources in advance and is suitable for non-core equipment[15].

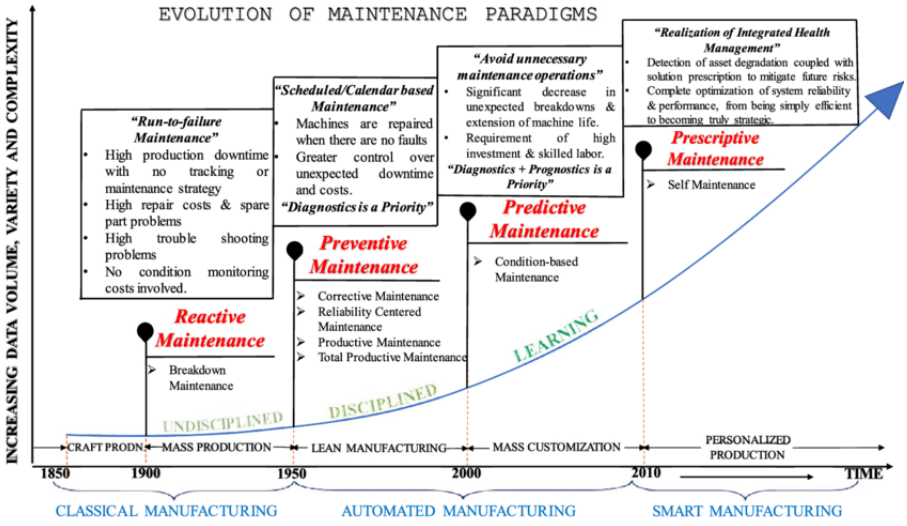

Figure 2.1: The evolution of maintenance paradigms[**?**].

2.1.2 Preventive Maintenance

Preventive maintenance is a proactive and systematic approach to preventing unplanned breakdowns of equipment, machinery or assets and reducing unplanned downtime through regular, planned inspections, maintenance and repairs. Specific activities of preventive maintenance may include cleaning, lubrication, calibration, parts replacement, and general inspections. These tasks are typically scheduled based on time intervals (time-based maintenance), usage (usage-based maintenance), or a specific operating condition (condition-based maintenance)[16]. By implementing preventive maintenance, companies can reduce emergency repair costs, minimise production interruptions, improve workplace safety and maintain consistent product quality[17].

2.1.3 Predictive Maintenance

During the transition from Industry 3.0 to Industry 4.0, the continuous expansion of industrial scale and rapid technological advancements have given rise to predictive maintenance (PdM). Unlike traditional maintenance, which relies on pre-scheduled inspections and servicing at fixed intervals, predictive maintenance seeks to intervene before failures occur, thereby preventing unexpected downtime and potential losses. The core idea of PdM is to collect operational data over time, monitor equipment condition, and identify correlations and patterns that help predict and ultimately prevent failures[18].

Most predictive maintenance applications utilize a variety of sensors and Industrial Internet of Things (IIoT) devices to continuously gather and analyze data generated during equipment operation. These data include, but are not limited to, vibration patterns, temperature, oil condition, eddy current analysis, and operational logs. By monitoring and analyzing these data streams in real time, the system can detect early signs of performance degradation[18]. In some cases, sensor data may be unavailable or insufficient for effective prediction; in such situations, event logs generated by the equipment can be analyzed. These logs record various status changes and anomalies throughout the equipment's operation, providing valuable information for researchers to predict failures and optimize maintenance schedules[19].

With the advancement of AI and machine learning technologies, predictive maintenance has become increasingly intelligent and reliable. Advanced deep learning algorithms can process high-dimensional, nonlinear time series data, capturing the complex dynamics of equipment operation and identifying precursors to failures even without explicit rules. Studies have shown that machine learning–enabled predictive maintenance systems can achieve failure prediction accuracies exceeding 95% [18], leading to a 30–50% reduction in unplanned downtime and a 20–40% decrease in maintenance costs[20]. The adoption of predictive maintenance allows maintenance teams to shift from traditional reactive or calendar-based interventions to more efficient, condition-based proactive strategies. This not only helps reduce failure rates and maximize asset up-time by improving reliability, but also optimizes operational costs by lowering maintenance expenses and maximizing productive time.

2.1.4 Prescriptive Maintenance

Industry 4.0 has not only greatly accelerated the widespread adoption of predictive maintenance but has also driven the emergence of prescriptive maintenance[21, 22]. As the fourth stage in the evolution of maintenance strategies, prescriptive maintenance is characterized by a paradigm shift from passive response to proactive intervention. Unlike earlier approaches, prescriptive maintenance goes beyond fault detection: it leverages data analytics and optimization algorithms to actively recommend or automatically execute the most effective maintenance decisions[16, 22]. Implementing prescriptive maintenance requires the integration of reinforcement learning (RL), multi-objective optimization, and digital twin technologies[23].

A typical prescriptive maintenance system consists of three core modules: the data acquisition layer collects real-time equipment status data via IIoT devices, the analysis and decision-making layer uses RL agents to predict equipment states and applies operational optimization algorithms to generate optimal maintenance solutions, and the execution layer translates generated decisions into machine-understandable instructions and immediately collects feedback data for further model refinement and optimization[11]. This closed-loop architecture enables continuous improvement in decision quality, fostering a virtuous cycle of knowledge accumulation within the system. For example, in the sheet metal glue line application, unsupervised deep learning models are used to analyze images of sheet metal glue lines, enabling effective detection and localization of anomalies, thereby significantly reducing false alarms and downtime of the gluing machine.[24].

The emergence of prescriptive maintenance marks a shift, from a focus on prediction to a focus on decision-making and optimization, in maintenance management. The scope of decision-making expands from single equipment to entire production systems[25]; the processing efficiency advances from offline analysis to real-time optimization[23]; and the goal evolves from simple cost control to the pursuit of sustainable development.

2.2 Deep Reinforcement Learning

Reinforcement Learning (RL) is a major branch of machine learning and is currently one of the most widely applied methods for intelligent decision-making and control. Traditional supervised learning relies on labeled datasets to construct input-output mappings; for example, in image classification tasks such as ImageNet, each sample is paired with an explicit class label[26]. Unsupervised learning, on the other hand, focuses on uncovering the intrinsic structure within data, such as revealing consumer behavior patterns through clustering analysis[27]. In contrast, RL emphasizes the discovery of optimal problem-solving strategies. In RL, an agent interacts with its environment, continuously optimizing its action policy to maximize cumulative rewards. When the accumulated reward converges to a maximum and stabilizes, it indicates that the agent has learned a globally or locally optimal policy[28]. Through this iterative loop of trial, feedback, and adjustment, the agent

gradually masters strategies that maximize cumulative rewards. RL is not only applicable to traditional control systems and games, such as AlphaGo defeating human champions[29], but also demonstrates powerful capabilities in complex domains such as robotic control[30], intelligent manufacturing[31], autonomous driving[32], and financial investment[14].

The development of RL has gone through several key stages. In its early days, tabular methods such as Q-Learning[33] and SARSA[34] enabled RL to achieve optimal decision-making in small-scale, finite state spaces. With the emergence of function approximation techniques, such as linear function approximation[35, 36] and radial basis functions[37], RL algorithms became capable of handling larger and continuous state spaces. In the 21st century, the ongoing advancement of neural network technologies has fostered the integration of deep learning with RL, giving rise to DRL.

At the heart of RL lies the interaction between the agent (the learner or decision-maker) and the environment (the external system with which the agent interacts), as shown in Figure 2.2. As the agent observes the environment, it selects actions and receives reward signals based on feedback from the environment. In this process, the agent's decision-making and learning rely on the following three core components:

- Policy: The mechanism that determines which action the agent takes in each state; it serves as the guiding principle for the agent's behavior.
- Value Function: An estimate of the expected cumulative reward, used to evaluate the quality of a particular state or state-action pair and provide a basis for decision-making.
- Model: The agent's internal representation of the environment's dynamics, describing the probabilities of state transitions and the distribution of rewards. The model typically needs to address two fundamental issues: (1) predicting the probability distribution of the next state given the current state and action, and (2) estimating the immediate reward expected for a given state-action pair.

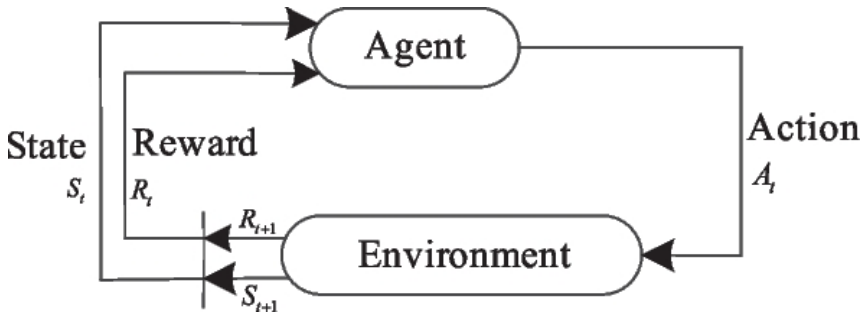

Figure 2.2: The agent-environment interaction in Markov Decision Process

In a typical RL framework, the environment is modeled as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ), where S represents the set of possible states, A denotes the set of available actions, P is the state transition probability

function, R is the reward function, and γ is the discount factor. At each discrete time step t, the agent observes the current state $s_t \in S$, selects an action $a_t \in A$, and transitions to a new state s_{t+1} according to the transition dynamics P. Simultaneously, the agent receives a reward $r_t = R(s_t, a_t)$ as feedback on the quality of the chosen action. The agent's objective is to find a policy $\pi(a|s)$, a mapping from states to actions, that maximizes the expected cumulative discounted return over time:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, \pi\right]$$

Based on the learning mechanisms of agents, RL agents can be classified as follows:

- Value-Based Agents: These agents maintain only a value function over states or state-action pairs and derive the optimal policy indirectly from the value function. Representative methods include Q-Learning[33] and DQN[38].
- Policy-Based Agents: These agents directly optimize the policy function without explicitly maintaining a value function. Usually these agents apply techniques like policy gradient search for the optimal solution. Examples include Policy Gradient[28] and PPO[39].
- Actor-Critic Agents: These agents combine both policy and value functions, using the value function to evaluate the current policy and guide its improvement. This synergistic approach is exemplified by algorithms such as A3C[40] and SAC[41].

In addition, RL agents can be classified into model-free and model-based agents based on whether they learn an explicit model of the environment or not. Model-based agents maintain an internal model of their environment and make decisions based on their understanding of the model. Learned models can be used for planning, policy improvement, or to generate simulated experiences to accelerate learning[42]. Model-free agents learn optimal policies or value functions directly from their interactions with the environment without building explicit models of the environment's dynamics. Currently, most DRL methods employ model-free agents, due to the fact that in most cases in current research, the environment is static and describable, and the state of the intelligences is discrete and observable[43].

2.2.1 Value-based Methods

Value-based methods are among the earliest and most systematic approaches in RL. Their core idea is to estimate a value function that evaluates the environment, thereby indirectly guiding the agent to take optimal actions. Value-based methods typically use the Bellman Equation for recursive estimation and optimization of the value function.

Bellman's dynamic-programming framework[44] models a sequential-decision task as an Markov decision processes with well-defined states $\mathcal{S}$, actions $\mathcal{A}$, rewards $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and state-transition probabilities π. The optimal value function

obeys the Bellman equation

$$Q(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q(s',a') \mid s_t = s, a_t = a\right]$$

which formalities the principle of optimality and underpins value-iteration, policy-iteration, and Q-learning algorithms. However, the Bellman equation assumes a known environment model—specifically, prior knowledge of $P(s' \mid s,a)$. This requirement limits its applicability to unknown environments, and its computational complexity grows explosively with the size of the state space.

To eliminate the requirement for a complete model or full returns, research shifted toward model-free methods that learn directly from experience. Monte Carlo methods estimate value by sampling entire trajectories: $V(s) \approx \frac{1}{N}\sum_{i=1}^{N} G_i(s)$, where $G_i(s) = \sum_{t=0}^{T} \gamma^t r_t$). But updates can be performed only after the episode terminates. Temporal-difference (TD) learning[45, 35] combines Monte Carlo sampling with dynamic-programming bootstrapping, enabling single-step, online updates: $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$. This algorithm updates online and therefore does not have to wait until an episode ends, but it introduces estimation bias.

Extending TD learning, Q-learning[33] directly learns an action-value function Q(s,a) and has become a cornerstone of modern RL. Q-learning employs an off-policy update that decouples the target policy from the behaviour policy:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

This rule guarantees asymptotic convergence to the optimal policy. In theory, a greedy target policy drives value maximisation, while an ε-greedy or similar behaviour policy ensures exploration. The resulting separation naturally complements mechanisms such as experience replay. In contrast, SARSA[34] performs an on-policy update: $Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma Q(s',a') - Q(s,a)\Big]$. Although this formulation yields lower update variance, policy improvement remains constrained by the exploration strategy currently in use.

Deep Q-Network (DQN)[38] is a foundational algorithm in DRL that combines traditional Q-learning with deep neural networks to enable decision-making in high-dimensional, discrete action spaces. It approximates the optimal action-value function using a neural network and employs experience replay and target networks to stabilize training. The Q-function is modeled by a deep neural network parameterized by $\theta : Q(s,a;\theta)$. The training objective is to minimize the temporal-difference (TD) error:

$$L(\theta) = \mathbb{E}(s,a,r,s')\left[(y - Q(s,a;\theta))^2\right]$$

where

$$y = r + \gamma \max a' Q(s',a';\theta^-)$$

where θ^- denotes the parameters of the target network. The transitions (s, a, r, s') generated from the agent's interactions with the environment are stored in a replay buffer. Mini-batches are then randomly sampled from this buffer for network training. This approach breaks the correlations between samples and improves data efficiency. A separate set of Q-network parameters is maintained for computing the TD targets. The parameters of the target network are periodically updated by copying the current Q-network parameters, which helps to mitigate instability caused by shifting targets during training.

Double Deep Q-Network (DDQN)[46] is an extension of the standard DQN designed to address a known issue in value-based RL algorithms: the overestimation bias caused by using the same Q-network to both select and evaluate actions during the update. Double Q-Learning decouples action selection and action evaluation: the online network is used to select the best action at the next state, and the target network is used to evaluate the value of that action. In DDQN, the target value is modified as follows:

$$y_t^{\text{DDQN}} = r_t + \gamma Q_{\theta^-}\left(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a')\right)$$

Where θ are the parameters of the online Q-network (used for action selection), θ^- are the parameters of the target Q-network (used for action evaluation), $r_t$is the reward at time t and γ is the discount factor. The DDQN loss function minimizes the mean squared error between the target and predicted Q-values:

$$L(\theta) = \mathbb{E}(s_t, a_t, r_t, st+1) \sim \mathcal{D}\left[\left(y_t^{\text{DDQN}} - Q_\theta(s_t, a_t)\right)^2\right]$$

This objective is used to update the parameters $\theta$of the online Q-network using stochastic gradient descent.

Dueling DQN[47] is an improved variant of the standard DQN architecture. Its core motivation lies in the observation that, in many states, the choice of action does not significantly affect the overall value of the state—that is, the advantages of different actions may be quite similar. Traditional DQN struggles to distinguish between "critical decisions" and "uninformative decisions" in such cases. Dueling DQN addresses this by decoupling the estimation of the state value from the advantages of individual actions, thereby enhancing both the representational capacity and the learning efficiency of Q-value estimation. This structure is especially beneficial in environments with large action spaces or many "irrelevant" actions. The output of the Dueling DQN is divided into two streams:

- State Value Stream: Represents the intrinsic value of a given state, denoted as $V(s; \theta, \beta)$.
- Advantage Stream: Represents the relative advantage of each action in a given state, denoted as $A(s, a; \theta, \alpha)$.

These two streams are combined to reconstruct the Q-value function as follows:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$

The significance of this architecture is that even when all actions in a given state have similar (possibly low) advantages, the network can still accurately learn the underlying value of the state itself. This leads to improved learning efficiency and stronger generalization in policy performance.

2.2.2 Policy-based Methods

Policy-based methods are an important class of reinforcement learning approaches whose core idea is to directly model and optimize the policy function, rather than indirectly deriving the policy through a value function as in value-based methods. The policy $\pi(a|s)$ defines a probability distribution over actions given a particular state. Policy-based methods are typically well-suited for continuous or high-dimensional action spaces, as well as scenarios where the policy needs to be represented as a probability distribution.

The objective of policy-based methods is to find an optimal policy that maximizes the expected cumulative return. Mathematically, this objective can be expressed as:

$$J(\theta) = \mathbb{E}\pi\theta \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

where θ denotes the parameters of the policy function.

REINFORCE[48] is one of the earliest and most fundamental Monte Carlo policy gradient algorithms, and serves as a representative example of baseline-free policy gradient methods. It directly models the policy function and updates its parameters by sampling complete episodes and estimating the expected return based on the actual rewards received. In the REINFORCE algorithm, the policy function $\pi_\theta(a|s)$ is parameterized by θ, and can be any differentiable probability distribution. The goal is to maximize the expected cumulative reward:

$$J(\theta) = \mathbb{E}\pi\theta \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

According to the policy gradient theorem, the update direction for the policy parameters is:

$$\nabla_\theta J(\theta) = \mathbb{E}\pi\theta \left[\nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right]$$

where G_t is the actual (discounted) return from time t until the end of the episode, and γ is the discount factor. For each time step t in each episode, the policy parameters are updated using:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

This algorithm is simple to implement and applicable to any differentiable policy parameterization, and can handle both continuous and discrete action spaces.

Proximal Policy Optimization (PPO)[39] is a family of policy gradient algorithms that strike a balance between implementation simplicity, sample efficiency, and training stability. The core idea of PPO is to constrain the magnitude of each policy update in order to prevent performance degradation caused by excessively rapid changes in the policy (known as policy collapse). Specifically, PPO incorporates a term into the objective function that directly limits policy changes, ensuring that the KL divergence or the likelihood ratio between the new and old policies does not deviate too much. There are two primary variants of PPO: PPO-Penalty and PPO-Clip.

PPO-Penalty constrains the magnitude of policy updates by employing a KL penalty term. Its objective function is:

$$L^{KLPEN}(\theta) = \mathbb{E}t \left[r_t(\theta)\hat{A}t - \beta \mathrm{KL}\left[\pi\theta old(\cdot|s_t), \pi_\theta(\cdot|s_t)\right] \right]$$

where $\mathrm{KL}[P, Q]$ denotes the KL divergence between the old and new policies, and β is the penalty coefficient (which can be adjusted adaptively).

PPO-Clip improves upon traditional policy gradient methods by introducing a clipped surrogate objective function, which constrains the policy updates to remain within a trusted region, thereby preventing drastic policy changes that can destabilize training. The objective function is

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the probability of taking action a_t at state s_t in the current policy divided by the previous one. Due to its simple implementation, lack of need for fine-tuning hyperparameters, and stable training performance, this variant has become the mainstream choice.

2.2.3 Actor-Critic Methods

Actor-Critic methods are hybrid policy optimization approaches in RL that combine value-based and policy-based methods. The core idea of Actor-Critic methods is to train two models simultaneously:

- Actor: Responsible for decision-making, representing the policy, and outputting either the probability of taking each action in a given state or directly outputting the action itself.

- Critic: Responsible for evaluation, representing the value function, and estimating the value (or action value/advantage) of the current policy in a given state.

Actor-Critic methods achieve synergistic progress by having the actor update the policy parameters while the critic evaluates the current policy and provides gradient signals for policy improvement.

One notable variant, Soft Actor-Critic (SAC)[41], is characterized by entropy regularization. The objective of SAC is to maximize the entropy of the policy in addition to the expected cumulative reward. A high-entropy policy encourages the agent to continuously explore various actions, helping to avoid getting stuck in local optima and improving adaptability in complex or uncertain environments. SAC simultaneously learns a stochastic policy and two Q-functions, combining the benefits of actor-critic methods with a strong exploration bias. The objective function can be written as:

$$J(\pi) = \sum_t \mathbb{E}(s_t, a_t) \sim \rho\pi \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

where $\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a_t \sim \pi}[\log \pi(a_t|s_t)]$ denotes the entropy of the policy at state s_t, and α is a tunable temperature coefficient that balances the contribution of reward and entropy. By incorporating key components such as the maximum entropy framework, double Q-networks, experience replay, and automatic temperature tuning, SAC achieves high sample efficiency, strong stability, and excellent policy generalization. As a result, it has become one of the most representative DRL algorithms for continuous action space problems. SAC has been widely applied in real-world scenarios such as robotic control and autonomous driving.

2.3 Generative AI

Generative AI is a major branch within the field of AI, referring to systems that can generate entirely new content based on existing data. Unlike traditional machine learning models, which typically focus on classification or regression of input data, generative models learn the underlying data distribution and generate similar data accordingly—including text, images, audio, and more[49, 50]. The roots of this field can be traced back to the 1980s and 1990s with latent variable models such as Gaussian Mixture Models, Hidden Markov Models, and Naive Bayes[51], though these early approaches were limited in expressive power and struggled with modeling high-dimensional data. In the early 21st century, with the advent of neural networks and deep learning, generative models experienced explosive progress.

In 2014, Ian Goodfellow and colleagues proposed the Generative Adversarial Network (GAN)[52], which consists of two components: a generator and a discriminator. These two models are trained in a zero-sum game framework: the generator tries to produce realistic samples to fool the discriminator, while the discriminator aims to distinguish between real and generated samples. Through this adversarial process, both models improve over time, and the generator eventually produces highly

realistic samples. That same year, the introduction of the Variational Autoencoder (VAE)[50] further advanced unsupervised generative modeling. VAEs are probabilistic generative models that maximize the marginal likelihood of the data in an unsupervised manner. They employ an encoder-decoder structure to map input data to a latent space distribution and sample from this distribution to generate new data. By introducing variational inference, VAEs enable efficient Bayesian inference and data generation. Subsequently, various autoregressive models (such as LSTM[53]) demonstrated outstanding performance in sequence generation tasks. Autoregressive models decompose the high-dimensional joint distribution into a chain of conditional probabilities, generating data step by step. For example, in text generation, the model first generates the initial word, then generates each subsequent word conditioned on the previous ones[54].

In 2017, Google introduced the Transformer architecture[55], which is based on the self-attention mechanism. Transformers can efficiently model long-range dependencies without relying on recurrence or convolution, greatly advancing natural language processing and multimodal generation. At the end of 2022, OpenAI released ChatGPT[56], bringing generative AI into the public spotlight and sparking a worldwide wave of AI-driven applications and innovation.

2.3.1 Transformer Framework

The Transformer model represents a major breakthrough in deep learning for sequential data, particularly in natural language processing (NLP). Unlike traditional recurrent or convolutional architectures, the Transformer relies entirely on an attention mechanism, specifically self-attention, to capture dependencies between input and output elements, regardless of their distance in the sequence. This innovation dramatically improves both the parallelizability and effectiveness of deep sequence modeling. Figure 2.3 shows the complete structure of Transformer model.
At the heart of the Transformer is the self-attention mechanism (also called scaled dot-product attention), which enables the model to weigh the relevance of different elements in a sequence when encoding a given element. To allow the model to jointly attend to information from different representation subspaces, the Transformer uses multi-head attention. The attention mechanism is run multiple times in parallel (with separate parameters), and the outputs are concatenated and linearly transformed. Since the Transformer lacks any recurrence or convolution, it needs a way to capture the order of sequence elements. This is achieved by adding positional encodings (either fixed or learned) to the input embeddings. The standard Transformer is composed of an encoder and a decoder. The encoder in the paper consists of 6 encoder layers stacked on top of each other, as does the decoder. The encoder layer consists of a multi-head self-attention sub-layer (with residual connection and layer normalization) and a position-wise feed-forward neural network (with residual connection and layer normalization). The decoder layer also contains the two layers of the network mentioned by the encoder layer, but there is an attention layer in the middle of these two layers that helps the current node to get the current focus of what it needs to focus on. The input passes through several stacked encoder

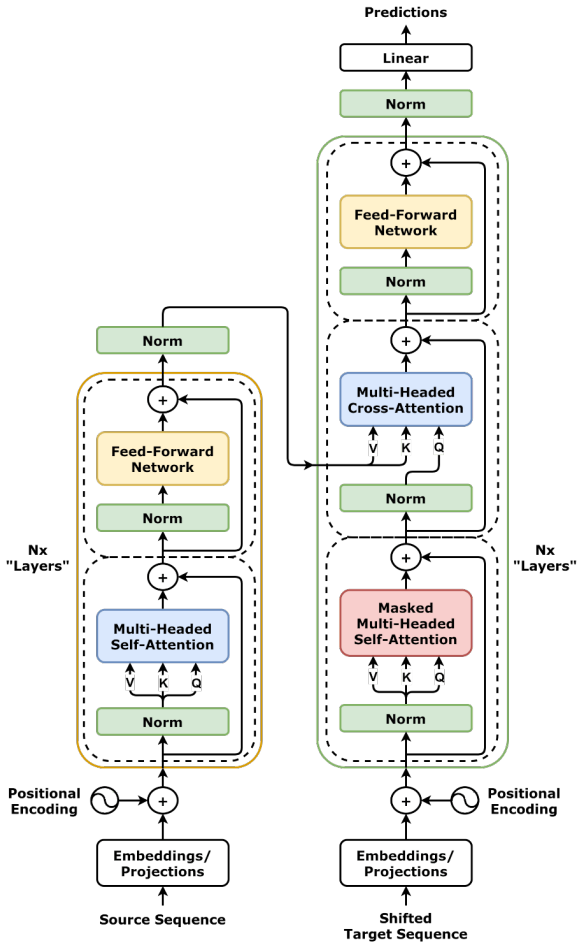

Figure 2.3: A standard Transformer architecture.

layers, then the decoder attends to both its own outputs and the encoder outputs to generate the final sequence[55].

Since its introduction, the Transformer has become the backbone of most state-of-the-art models in NLP and beyond. Variants such as BERT[57], GPT[58], and Vision Transformer[59] extend the original architecture to diverse applications including language understanding, text generation, computer vision, and multi-modal tasks.

2.3.2 Large Language Model

Since the release of OpenAI's GPT-3 in 2020, the field of LLMs has experienced explosive growth. This wave of innovation has been driven by advancements in model architecture (particularly the Transformer), the availability of massive datasets, and breakthroughs in training techniques. GPT-3, with 175 billion parameters, set new benchmarks in language understanding and generation[60]. Its successor, GPT-3.5, powered the original ChatGPT and demonstrated strong abilities in conversational AI, code generation, and multi-turn dialogue[56, 61].

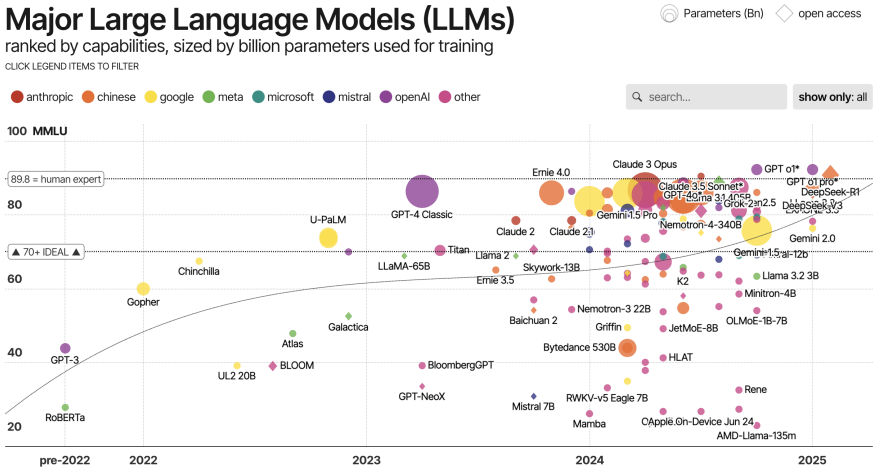

Figure 2.4: A visualization of major LLMs, ranked by performance[2].

Subsequently, Google's Pathways Language Model (PaLM)[62, 63] and the recently released Gemini[64] are large-scale, multi-modal models capable of reasoning across text, images, audio, and video. They demonstrate state-of-the-art performance on benchmarks, featuring powerful reasoning and multi-modal capabilities. In particular, Gemini integrates text, images, and audio into a unified model. Meta released LLaMA (Large Language Model Meta AI) in 2023 as a competitive open-source alternative to GPT models[65]. LLaMA-2 and LLaMA-3 further improved performance, openness, and efficiency[66]. LLaMA is renowned for its efficient scalability, strong performance on language tasks, and widespread adoption within the open-source community. Anthropic's Claude series emphasizes constitutional AI and alignment, aiming to produce safer and more controllable AI assistants[67]. The Claude models excel at maintaining helpfulness and safety and are optimized for responsible conversational design. Alibaba's Qwen models were released in 2023 as Chinese-centric open-source LLMs with multilingual capabilities. Some Qwen models, such as Qwen-VL, support multi-modal tasks and can process both text and images simultaneously[68]. DeepSeek is an emerging player in the LLM space, backed by a prominent research lab with a focus on open, scalable, and practical language models. DeepSeek released its main model series, including DeepSeek-LLM and DeepSeek-VL, in 2023–2024[69, 70]. On key benchmarks for reasoning, language understanding, and code, DeepSeek models are competitive with, and sometimes surpass, models such as GPT-3.5 and LLaMA-2/3[71]. Figure 2.4 shows the latest LLMs and their performance ranking.

2.4 State of The Art

Several studies have demonstrated the significant value of DRL in maintenance due to its ability to dynamically optimize schedules and prioritize tasks using real-time data. By integrating DRL with digital twins and customizing reward functions based on production cost models, maintenance task sequencing is dynamically optimized through real-time data analytics. A lab-scale drone factory case study

validated this approach, achieving an average reward of 0.67 and demonstrating that AI-enhanced digital twins significantly improve operational efficiency[1]. Utilizing Multi-State Degradation Models (MSDM) synergized with DRL, sewer pipe maintenance strategies are optimized to generate intelligent, cost-saving policies[72]. Research also confirms that DRL-driven digital twins reduce railway infrastructure maintenance activities by 21% and lower defect occurrences by 68% through real-time prioritization[73].

Recent research has demonstrated the value of LLMs in maintenance applications, particularly for processing unstructured data and extracting actionable insights. For example, an industrial case study at Ericsson AB validated that multi-agent LLM architectures automate software test maintenance by analyzing code changes and predicting required updates, significantly improving efficiency in industrial settings[74]. In the domain of bug triaging, models like SevPredict (based on GPT-2) enhance defect severity prediction accuracy in software maintenance, effectively reducing manual inspection burdens and boosting automated triage capabilities[75]. Furthermore, LLMs have been applied to aircraft propulsion diagnostics, where they analyze telemetry data to identify anomalies and generate maintenance guidelines. Experimental results confirm these models significantly improve reliability even under suboptimal data-quality conditions, demonstrating robust performance in critical systems[76].

Recent research has shown significant advancements in integrating LLMs with DRL for industrial maintenance optimization. The TranDRL framework exemplifies this progress by combining Transformer-based LLMs for remaining useful life prediction with DRL for maintenance action optimization, resulting in a 15–28% reduction in mean absolute error compared to traditional methods and more adaptive scheduling on benchmark datasets[77]. Similarly, the LLM-Empowered State Representation approach addresses sample inefficiency by using LLMs to autonomously generate task-specific state representations and intrinsic reward functions, which not only reduces training iterations by 50% but also improves policy robustness across various equipment failure scenarios[78]. The Latent Reward method further enhances DRL by employing LLMs as intelligent observers to evaluate and distribute rewards at the action level in sparse-reward environments, thereby accelerating DRL convergence by a factor of three and improving exploration in complex, multi-stage maintenance tasks[79]. In dynamic network maintenance, LLM-Augmented Deep Reinforcement Learning for O-RAN Network Slicing introduces learnable prompts that optimize both semantic clustering and RL objectives, structuring unorganized network feedback into latent representations and achieving faster convergence and higher rewards in wireless network maintenance scenarios[80].

In summary, these studies demonstrate that LLMs can significantly enhance DRL by providing semantic contextualization of unstructured data, shaping rewards for complex tasks, and refining state representations to reduce sample complexity, thereby enabling practical implementation of prescriptive maintenance applications in industrial settings.

3

Methodology

3.1 Overview of Framework

This study adopts a four-layer collaborative architecture (as shown in the Figure 3.1), utilizing the DRL-digital twin system proposed by Chen et al.[1], combined with LLM, to implement a closed-loop system for smart maintenance decision-making.

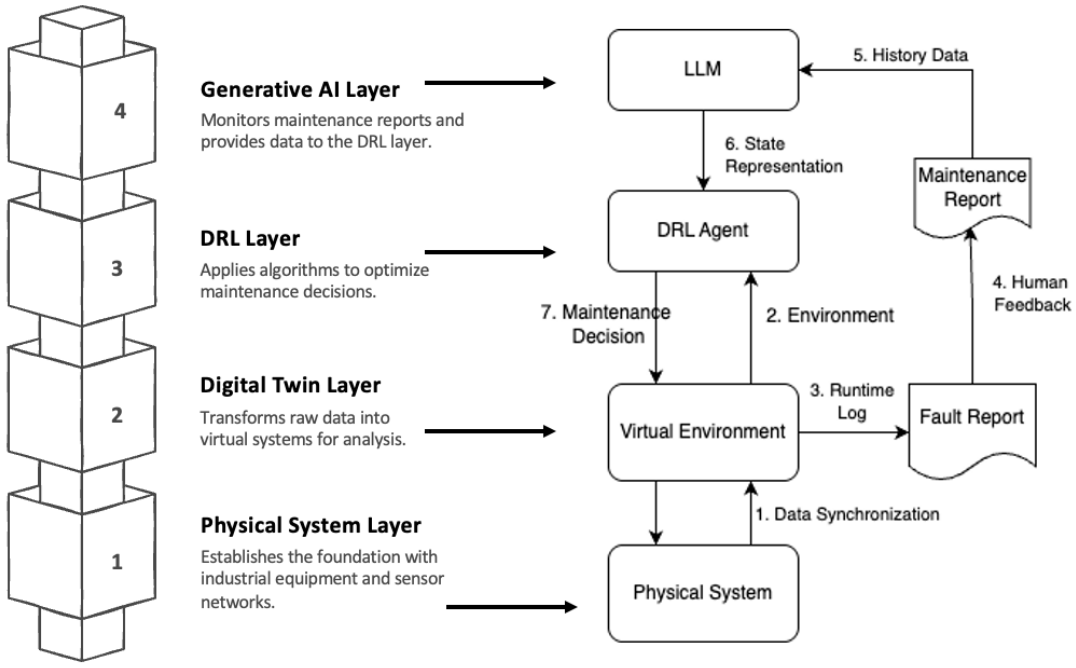

Figure 3.1: Overview of the four-layer architecture.

As the physical foundation of the architecture, the physical system layer consists of clusters of industrial equipment and distributed sensor networks, which continuously collect multimodal data including operational status, process parameters, and environmental indicators. The digital twin layer employs dynamic modeling techniques to transform the raw data from the physical layer into a computable virtual system, thereby providing structured data support for decision analysis. The Generative AI layer monitors fault reports generated by the digital twin layer and provides state representation data to the DRL layer. The DRL layer utilizes advanced algorithms

to continuously train and optimize the policy network based on system data from the digital twin layer and state representation data from the LLM in Generative AI layer, generating optimal maintenance decisions.

In the digital simulation environment, an efficient bidirectional data interaction mechanism is established between the digital twin layer and the DRL layer, achieving dynamic coordination between the virtual model and the intelligent decision-making system. After analyzing real-time machine operating status obtained from the digital twin layer, the DRL layer utilizes state-of-the-art machine learning architectures to process and analyze high-dimensional industrial input data. This enables simulation of diverse production scenarios and generation of proactive predictive maintenance strategies for the digital twin layer. Upon executing corresponding actions, the digital twin layer produces updated machine states, which are subsequently acquired by the DRL layer for continuous learning iterations. Real-time, accurate data transmission is facilitated through standardized application programming interfaces (APIs).

Meanwhile, the simulation system continuously records equipment failures and abnormal events in real-time fault logs. Maintenance teams monitor these log data in real-time, enabling immediate and precise intervention in the physical system upon anomaly detection. For each fault log entry, maintenance personnel can add feedback comments, generating corresponding analyzable maintenance logs. Upon detecting log updates, the LLM in Generative AI layer automatically triggers to generate updated system state representation vectors based on new inputs. These vectors are transmitted in real-time to the DRL layer's state space, providing rich maintenance context that supports continuous optimization of decision policies through a closed feedback loop.

This iterative and tightly integrated feedback loop enables the continuous evolution of maintenance policies. By leveraging real-time data, simulated outcomes, human expertise, and LLMs, the framework fosters a self-improving system that enhances the reliability, efficiency, and responsiveness of industrial maintenance operations.

3.2 Physical Layer

This layer generally refers to the machines and equipment in a modern industrial environment. A contemporary production workshop typically contains hundreds of machines that cooperate with one another to carry out manufacturing tasks. The experiment is conducted at the SII Laboratory in Lindholmen, Gothenburg, which serves as Sweden's national industrial digitalization testbed. Figure 3.2 shows the stations in laboratory. The laboratory features a drone assembly line, comprising a main conveyor belt (with a transport precision of ± 0.5 mm), four assembly stations (responsible for fuselage assembly, power system integration, flight control module installation, and outer shell packaging, respectively), and a quality inspection station. The main conveyor belt transports the drones sequentially from the first assembly station through the remaining three for assembly, and finally to the quality inspection terminal. Each machine is equipped with an IIoT sensor network

that captures key performance indicators in real time, including overall equipment effectiveness, individual cycle times, tool wear conditions, and unplanned downtime events. The resulting industrial big data are transmitted through an enterprise-grade IIoT platform to the simulation system.

Figure 3.2: Drone assembly stations at the SII laboratory.

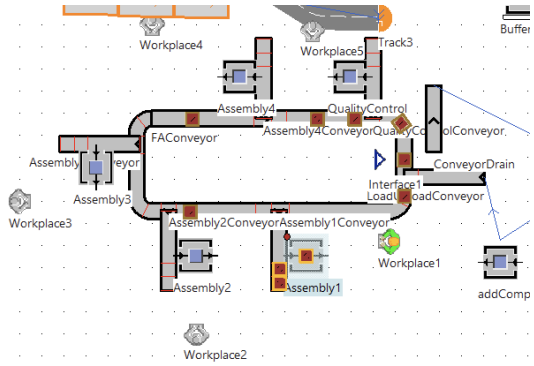

Figure 3.3: A screenshot of Siemens Plant Simulation Platform

3.3 Digital Twin Layer

Siemens Plant Simulation is employed to accurately replicate the 3D layout of the physical workshop and the dynamic interactions between machines. The Siemens simulation model is synchronized with the physical system via the ThingWorx IIoT platform. Figure 3.3 shows the simulation model we built on Siemens Plant Simulation platform, which serves as the virtual environment of our experiment. The model includes three assembly stations and one quality inspection station, corresponding to the physical system layer.

3.4 Deep Reinforcement Learning Layer

Building upon the physical layer and the digital twin layer, DRL layer is further introduced to enable intelligent maintenance decision-making by continuously interacting with the digital simulation platform.

3.4.1 Markov Decision Process

Before we dive into the DRL algorithms, the maintenance problem is modeled into Markov Decision Process (MDP)[81], which can be defined by a tuple (S, A, R, P, γ), where:

- S denotes the state space of the system with $S = \{s_1, s_2, s_3, \cdots, s_n\}$ consisting all possible n states that the system can take at time t.
- A denotes the maintenance action space with $A = \{a_1, a_2, a_3, \cdots, a_m\}$ if we denote the number of possible maintenance actions by m.

- $P(s_j^{t+1}|s_i^t, a_k^t)$ represents the dynamic transition function, defining the probability of transitioning from state s_i^t to state s_j^{t+1} after taking action a_k at time t.

- $R(s_i^t, a_k^t, s_j^{t+1})$ represents the reward function, specifying the immediate reward received after transitioning from state s_i^t to state s_j^{t+1} after taking action a_k^t. The expected reward for taking action a_k^t within state s_i^t is given by $R(s_i^t, a_k^t) = \sum_{j=1}^{n} P(s_j^{t+1}|s_i^t, a_k^t) R(s_i^t, a_k^t, s_j^{t+1})$.

- $\gamma \in [0, 1]$ is the discount factor.

A policy function π is a (probability) mapping from state space to action space. The goal of MDP is to find an optimal policy $\pi^*(a|s)$ that maximizes the expected cumulative discounted reward. If we assume the value of a state s_i is under policy π is $V^\pi(s_i) = R(s_i, \pi(s_i)) + \gamma \sum_{j=1}^{n} P_{i,j}^{\pi(s_i)} V^\pi(s_j)$, then the value of a state under an optimal policy π^* obeys this recursive Bellman optimality equation $V^{\pi^*}(s_i) = max_{a \in A}\Big[R(s_i, a) + \gamma \sum_{j=1}^{n} P_{i,j}^{a} V^{\pi^*}(s_j)\Big]$.

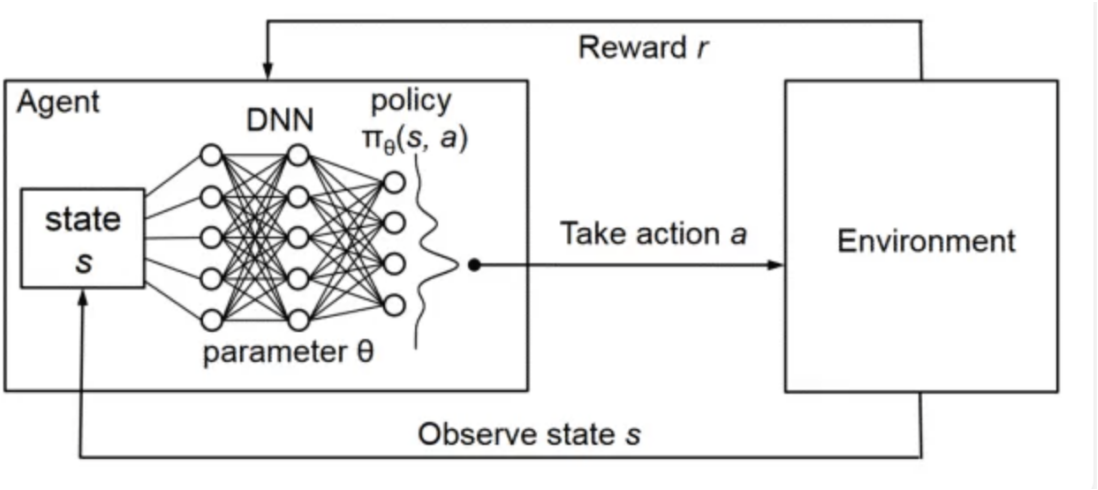

Figure 3.4: A schematic diagram of DRL.

As for the proposed framework for intelligent maintenance, the problem can be formulated by MDP as the subsequent discussions:

- States: The state space contains both the real-time maintenance status of equipments and key performance metrics of the system. The state at time t can be interpreted as $s^t = \{Ava^t, CT^t, TPH^t, AS1^t, AS3^t, AS4^t\}$, where Ava^t and CT^t are relative to system performance, which stand for the availability of the system and the cycle time, which is the total time taken to process a product from start to finish, respectively. TPH^t, throughput per hour, measures the number of units completed within one hour, which is used to measure the efficiency and output capacity of a system, process, or piece of equipment. $AS1^t, AS3^t, AS4^t$ indicate the maintenance of three assembly stations (AS1, AS3, AS4). These three variables are floating-point numbers ranging from 0 to 10, where smaller values indicate that the equipment is more unstable and

requires greater attention. The data is derived from the historical operational status of the corresponding equipment, including runtime, failure events, and maintenance records.

- Actions: The action space is a discrete and finite set consisting of 24 distinct actions, representing all possible permutations of maintenance priorities among the three assembly stations (AS1, AS3, AS4) and the quality control station (QC), calculated as 4!. Each element $a_i \in A$ takes a value from the set $1, 2, 3, 4$, representing the assigned priority level of the i_{th} workstation. A higher value indicates greater urgency for maintenance.

- Reward Model: The reward function is strategically derived by selecting key elements from a sophisticated reward function[82] that comprehensively considers both the detailed costs of the production environment and broader sustainability objectives. Given the partial observability of data in the model, this function selectively aggregates empirically measurable elements. To ensure dimensional consistency among different contributing factors and to prevent any single factor from disproportionately influencing the results due to its numerical scale, each contributing factor is normalized to the $[0, 1]$ range. Consequently, the immediate reward R_t at time t, Rt , is computed as a weighted sum of these normalized factors:

$$R_t = w_1 \cdot v_{OPE} + w_2 \cdot v_{CT} + w_3 \cdot v_{SR} + w_4 \cdot v_{TPH}$$

  where $v_{OPE} = \frac{OPE}{max(OPE)}$ denotes the normalized overall process effectiveness, $v_{CT} = \frac{max(CT)-CT}{max(CT)-min(CT)}$ denotes the normalized cycle time, $v_{SR} = \frac{max(SR)-SR}{max(SR)}$ denotes the normalized scrap ratio, and $v_{TPH} = \frac{TPH-min(TPH)}{max(TPH)-min(TPH)}$ denotes the normalized throughput per hour. w_1,w_2,w_3, and w_4 are the weights with constraint $w_1 + w_2 + w_3 + w_4 = 1$. This balances the reward function between sustainability and productivity, guiding the RL agent to take optimal actions that enhance production efficiency, improve cycle time efficiency, minimize defect rates, and maximize throughput.

In selecting appropriate DRL techniques for this study, since our application involves a discrete and finite action space, we adopted value-based algorithms from the DQN family, with a particular focus on Double DQN (DDQN), which represents one of the most advanced and widely recognized value-based DRL approaches. Additionally, among policy-based methods, we selected Proximal Policy Optimization (PPO), especially the PPO-Clip variant, which is a policy-gradient algorithm that has been extensively validated and widely applied in experimental research. The detailed structure of DRL shown in Figure 3.5.

3.4.2 DDQN Algorithm

This framework adopts the DDQN algorithm because the strategy used by the traditional DQN algorithm to estimate Q-values tends to produce overestimation bias[46], which can negatively impact performance. DQN uses the same network both to

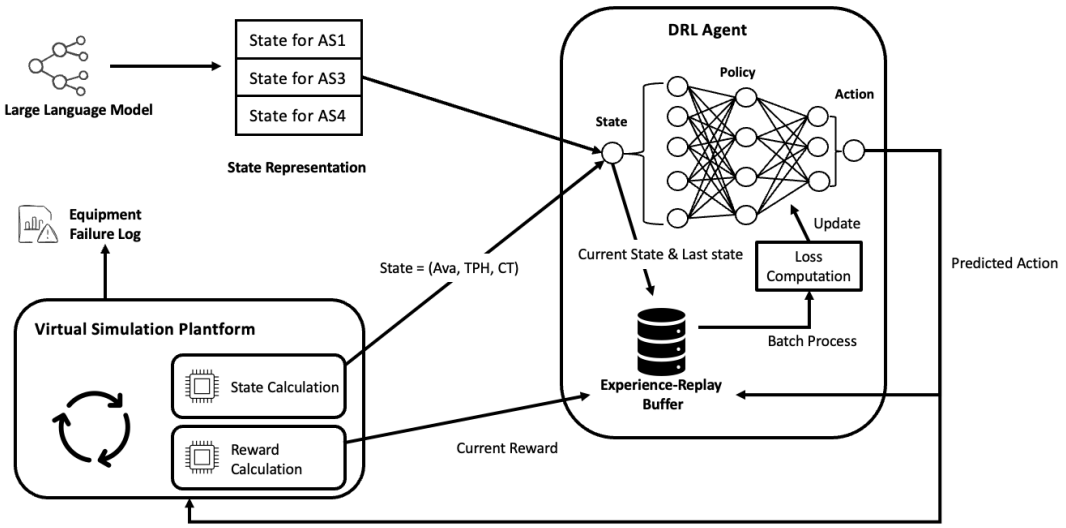

Figure 3.5: The detailed structure of DRL agent.

choose the action (using max operation) and to evaluate it, since the parameters θ^- of target network are copied every τ steps from the online network. The target become

$$Y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q_{\theta_t^-}(s_{t+1}, a)$$

The use of maximum operation introduces overestimation bias in both action selection and value estimation. The DDQN deals with this coupling problem by using two separate Q-value estimators, one online Q-table for action selection and one target Q-table for value evaluation of the chosen action a. The target is rewritten as

$$Y_t^{\text{DDQN}} \equiv R_{t+1} + \gamma Q_{\theta_t^-}\Big(s_{t+1},\ \arg\max_a Q_{\theta_t}(s_{t+1}, a)\Big)$$

The corresponding action a is chosen with the parameters θ in the online network but the value is locked up with the target value θ^- in the target network. The target network is then updated at certain intervals. The application of DDQN can help the framework find the optimal maintenance action in the current equipment state, ensuring robustness and adaptability during training.

The proposed algorithm of DRL agent adopts a Dueling DQN architecture[47] for both online network Q_θ and target network Q_{θ^-}. The Dueling DQN factories the Q function into a state-value stream $V_\psi(s)$ and an advantage stream $A_\phi(s, a)$, and then recombines them to produce the final Q-values.

$$Q_\theta(s, a) = V_\psi(s) \ + \ \Big(A_\phi(s, a) - \tfrac{1}{|\mathcal{A}|} \textstyle\sum_{a'} A_\phi(s, a')\Big)$$

This design improves sample efficiency and reduces unnecessary gradient variance, which empirically accelerates convergence in maintenance-scheduling tasks where many actions share similar returns.

During each training step the agent adopts the ϵ-greedy policy: with probability ϵ the agent samples from a soft distribution that favors under-explored actions;

Algorithm 1 DDQN Algorithm for Optimizing Maintenance Priority

Initialize online network Q_θ with parameters θ
Initialize target network Q_{θ^-} with parameters $\theta^- = \theta$
Initialize replay buffer $\mathcal{D}$
for each episode **do**
    Reset environment and observe initial state s
    while current state is not the goal state **do**
        Select a random action a_i based on the $\epsilon-$greedy policy
        Execute action a_i, observe reward r_i and next state s'_i
        Store transition (s_i, a_i, r_i, s'_i) in experience reply $\mathcal{D}$
        Set $s \leftarrow s'$
        if the size of $\mathcal{D} \geq$ threshold **then**
            Sample random mini-batch of transitions (s_i, a_i, r_i, s'_i) from $\mathcal{D}$
            Compute target for each sampled transition
            Perform a gradient descent step with respect to Q_θ
            Periodically update target network $\theta^- \leftarrow \theta$
        end if
    end while
end for

otherwise it exploits the arg-max Q-value. The chosen action is executed in the simulation model, yielding the reward r and next state s'. Interaction tuples (s, a, r, s') are stored in an experience-replay buffer and sampled in mini-batches; the online network parameters θ are updated by minimizing the mean-squared error $\mathcal{L}(\theta)$ via back-propagation and Adam optimization[83].

$$\mathcal{L}(\theta) = \frac{1}{B}\sum_{i=1}^{B}\Big[y_i - Q_\theta(s_i, a_i)\Big]^2$$

To ensure learning stability, a hard synchronization from the online network to the target network is performed only after every $N = update\_interval$ gradient updates.

3.4.3 PPO Algorithm

This study also adopts PPO, which is a typical on-policy policy gradient algorithm. Policy-gradient methods are vary sensitive to the choice of step size: an update that is too small slows convergence, whereas an overly aggressive update drives the new policy far from the old one and destabilizes learning[84]. PPO alleviates this sensitivity by optimizing a surrogate objective that can be evaluated on mini-batches and iterated several times per batch. There are two primary variants of PPO: PPO-Penalty, which introduces a Kullback–Leibler (KL) penalty via a Lagrange multiplier which is updated online to keep the KL divergence near a target value[85], and PPO-Clip, which clips the probability ratio so that successive policies cannot deviate beyond a fixed threshold[86]. PPO-Clip was chosen because the maintenance-priority task has a small, discrete action space, and in our preliminary experiments the clip variant consistently converged faster than the penalty-based

alternative.

PPO-clip updates policy by maximizing a clipped surrogate objective that constrains how much the new policy can deviate from the old policy at each update.

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} [L(s, a, \theta_k, \theta)]$$

where the clipped surrogate objective L is given by

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \;\; \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right)$$

where ϵ is a small hyperparamete, which roughly says how far away the new policy is allowed to go from the old, and $A^{\pi_{\theta_k}}(s, a)$ is the estimated advantage[39]. The *min* and *clip* functions ensure that the policy update stays within a safe range.

The PPO model adopts a structure comprising an actor network and a critic network, both comprise three hidden layers nestled between the input and output layers. The hidden layers use Rectified Linear Unit (ReLU) activation and parameters are updated with the Adam optimization. This hierarchy extracts increasingly abstract features from the maintenance state space, enabling the actor to output a priority distribution and the critic to estimate the state value.

3.4.4 SAC algorithm

SAC[41] is a hybrid method that combines policy optimization with Q-learning. On one hand, it trains a Q-network (called the "critic") using a loss function based on the Bellman equation. At the same time, it improves the policy (the "actor") by minimizing a loss function that helps to maximize the expected reward. SAC builds on ideas from earlier algorithms, such as using double Q-learning and target networks. But the main innovation of SAC is the use of entropy regularization in reinforcement learning. Entropy regularization measures how random the policy is. This encourages the policy to stay random during training, which helps avoid getting stuck in local optima.

To make the SAC algorithm work with discrete action spaces, two main equations need to be adjusted: the loss function for the policy π (the actor), with parameters θ; the loss function for the Q-function (the critic), with parameters ϕ. These equations come from the definitions of the optimal policy and the Q-function. With the usual actor-critic method, the goal is to find a policy that maximizes the total discounted reward over time:

$$\pi^* = argmax_\pi \mathbb{E}_\pi [\sum_{t=0}^{t} \gamma^t (R(s_t, a_t, s_{t+s}) + \alpha \mathcal{H}(\pi(\cdot|s_t)))]$$

where π is the policy ,$\mathcal{H}(\pi(\cdot|s)) = -\sum_a \pi(a|s) \log \pi(a|s)$ denotes the entropy of the policy, and α is a temperature, which determines the trade-off between the obtained reward and the randomness of the policy. $\pi(\cdot|s_t)$ denotes the probability density

Algorithm 2 PPO Algorithm for Optimizing Maintenance Priority

```
Initialize actor network
Initialize critic network
Initialize empty PPO memory
for each episode do
    Reset environment and observe current state s
    while s is not the goal state do
        Observe current state s
        Obtain action a and π(a|s) based on the current policy θ with st as input
        Obtain value V(s) from critic network with s as input
        Execute action a and choose maintenance priority for stations
        Get reward r and next state s' from environment
        Store the trajectory (s, a, r, V(s), π(a|s)) into memory buffer
        for each epoch do
            Get samples from memory buffer
            for each step in the trajectory do
                Compute advantage Â_t using Generalized Advantage Estimation(GAE)
            end for
            for each mini-batch do
                Compute probability ratio
                Compute the surrogate objective L
                Calculate actor loss and critic loss
                Update actor parameters and critic function parameters
            end for
        end for
        Clear memory buffer
    end while
end for
```

function (pdf) of the action distribution given by the policy in state s. The policy $\pi_\phi(a|s)$ is parameterized by a neural network that outputs the probability of each discrete action (typically using a softmax layer). A double Q-network architecture is usually adopted, i.e., two Q-networks Q_{θ_1} and Q_{θ_2}, to reduce overestimation bias. For each state s and action a, the networks output the Q-value $Q_{\theta_i}(s, a)$. The actor is updated by minimizing the following objective:

$$J_\pi(\phi) = \mathbb{E}s \sim \mathcal{D}\left[\sum a \in \mathcal{A}\pi_\phi(a|s)\left(\alpha \log \pi_\phi(a|s) - Q_{\min}(s, a)\right)\right]$$

where $Q_{\min}(s, a) = \min(Q_{\theta_1}(s, a), Q_{\theta_2}(s, a))$. This encourages the policy to assign higher probability to actions with higher Q-values, while also maintaining sufficient policy entropy. For each (s, a, r, s') tuple, the Q-networks are optimized to minimize:

$$J_Q(\theta) = \mathbb{E}(s, a, r, s') \sim \mathcal{D}\left[\left(Q\theta(s, a) - y\right)^2\right]$$

where $y = r + \gamma V_{\text{target}}(s')V_{\text{target}}(s') = \sum_{a'} \pi_\phi(a'|s')\left[Q_{\text{target}}(s', a') - \alpha \log \pi_\phi(a'|s')\right]$ That is, the target value is the expected soft value over all possible actions in the

next state. α is updated to match a target entropy:

$$J(\alpha) = \mathbb{E}_{a\sim\pi\phi(\cdot|s),s\sim\mathcal{D}}\left[-\alpha\left(\log\pi_\phi(a|s) + \mathcal{H}\text{target}\right)\right]$$

where $\mathcal{H}$target is typically set to $-\log\frac{1}{|\mathcal{A}|}$. Then the target network is updated by $\theta_{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta_{\text{target}}$.

The adopted SAC algorithm employs Double Q-learning, where two Q-networks are trained simultaneously, and the smaller of the two predicted values is used to prevent overestimated Q-values due to gradient-based optimization. Therefore, the algorithm includes four Q-networks in total: two for the critic and two as target networks. All of these use two-layer feedforward neural network architectures. Additionally, there is an actor network that outputs the probability distribution over actions, as well as a Replay Buffer. The temperature parameter is also optimized during training using a dedicated loss function.

Algorithm 3 SAC Algorithm for Optimizing Maintenance Priority

```
Initialize actor network
Initialize critic networks
Initialize reply buffer
for each episode do
    Reset environment and observe current state s
    while s is not the goal state do
        Observe current state s
        Obtain action a based on the exploration strategy
        Execute action a and choose maintenance priority for stations
        Get reward r and next state s' from environment
        Store the trajectory (s, a, r, V(s), π(a|s)) into reply buffer
        if the size of reply buffer > batch size then
            Get samples from reply buffer
            Calculate policy distribution and log probability
            Compute two Q networks
            Compute the actor loss
            Update actor network
            Update temperature parameters
            Calculate target Q value
            Calculate critic loss and backpropagate
            Soft update target Q Network
        end if
        Clear memory buffer
    end while
end for
```

3.5 Generative AI Layer

Although DRL algorithms excel at continuously learning and making optimal decisions in complex industrial environments, providing a powerful tool for maintenance

tasks, they consistently suffer from a lack of explainability and reliability, which can make their decisions difficult to trust. Since the expertise and judgment of maintenance technicians are highly valuable and can serve as a strong basis for maintenance decision-making, incorporating their insights is crucial for achieving truly intelligent maintenance[77]. Therefore, we propose a solution that integrates human feedback on machine conditions into existing DRL algorithms, enabling the DRL agent to make more informed and trustworthy decisions regarding maintenance priorities.

Transformer-based LLMs have now become powerful tools for time series analysis tasks, owing to their proficiency in understanding complex temporal sequences. Therefore, we leverage popular open-source LLMs to convert formatted maintenance reports, including technician feedback, into numerical states. These generated states serve directly as the machine state variables within the DRL agent's state space, enabling the DRL algorithm to make more intelligent and informed maintenance decisions.

3.5.1 State Representation

To determine the state of a equipment, it is essential to analyze its historical maintenance records. These records should include the failure start and end times, the type of fault, the repair solution, and technician feedback. Such structured data can be directly integrated into a predefined prompt template, allowing the LLMs to generate specific results. However, to ensure that the LLMs provide more reliable assessments, it is beneficial to include a portion of historical data, such as the ten most recent maintenance records for the current equipment, and to specify clear benchmarks in the prompt, for example, indicating that frequent failures within a short interval signify poor machine condition. This approach enables the language model to produce outputs that are more reasonable and aligned with real-world expectations.

3.5.2 Large Language Models

DeepSeek[69], Llama[66], and Phi[87] are leading open-source LLMs, providing full transparency and the ability to customize or fine-tune models for specific industrial maintenance scenarios. This openness is particularly valuable in the industrial domain, where privacy, security, and domain-specific requirements often demand tailor-made AI solutions. These models have demonstrated state-of-the-art performance on a variety of natural language understanding and summarization tasks, which is critical for accurate extraction and abstraction of valuable information from maintenance reports.

In our proposed framework, it is crucial for the LLM to generate results quickly, as the primary task is to distill and summarize input data. Since higher model parameters tend to require more extensive and expensive computational resources, Lightweight LLMs can be optimal in this context, as they maintain strong performance despite their smaller size. For example, MetaAI uses pruning techniques to

reduce model size by removing unnecessary components, and applies knowledge distillation so that smaller models can "learn" from their larger counterparts. As a result, the Llama-3.2 1B and 3B models demonstrate impressive capabilities. With their ultra-fast processing speed, these models can provide instant responses, making them ideal for deployment in time-sensitive industrial environments.

For DeepSeek, the DeepSeek-R1-Distill-Qwen-1.5B model is specifically selected for its advanced distillation pipeline, which effectively transfers the knowledge and reasoning capabilities of larger Qwen models into a compact and computationally efficient 1.5B-parameter network. The latest Llama-3.2-3B model is chosen due to its unique combination of instruction tuning, model compression, and open-ended reasoning capabilities. The Phi-3.5-mini model is selected because of its cutting-edge performance in lightweight, real-time reasoning and instruction following. All models are accessed through the official Hugging Face Model Hub since it offers robust APIs, a rich repository of cutting-edge pretrained language models[88], which allows for efficient experimentation, fine-tuning, and comparative evaluation in a unified environment.

To improve response efficiency, we adopted a batch processing approach using mul-

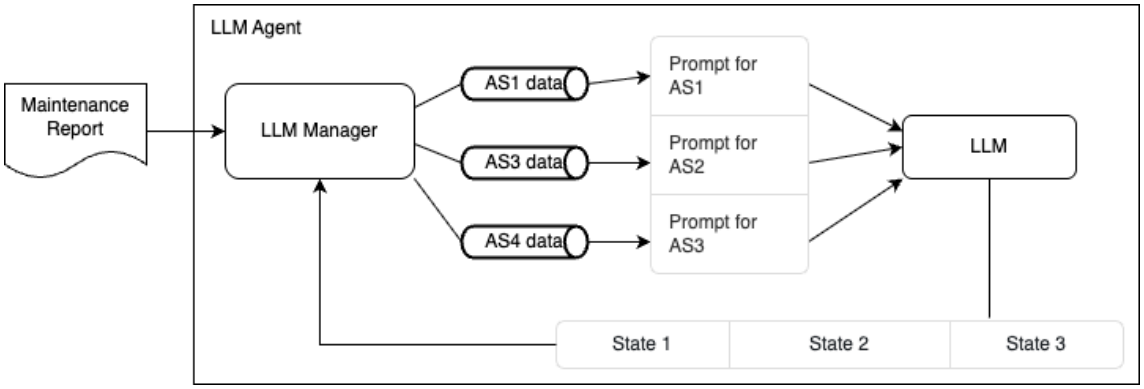

Figure 3.6: The detailed structure of LLM agent.

tiple prompts simultaneously. After receiving the reports, the LLM agent groups them by model name, for example, there are three different models in our lab. The grouped maintenance data are organized and concatenated into corresponding prompt templates. These prompts are then fed to the tokenizer and model in batches to generate feedback for all models at once. This design significantly accelerates the process of state representation. In particular, when conducting experiments on simulation platforms, the ability to rapidly generate state values is crucial for keeping pace with the consumption rate of the DRL agent.

3.6 Case Study

In this section, we present a case study of our proposed framework.

3.6.1 Data

The simulation platform at the digital twin layer can generate real-time reports in CSV format under simulation mode, capturing details such as the failure start time, end time, failure type, and the predicted time until the next failure. With machine availability set at 60%, the simulation platform is capable of producing approximately 190 failure records per second during simulation.

The corresponding maintenance data, including failure causes, repair methods, and human feedback, are collected from the lab's historical maintenance records. In total, 15 types of failure causes and 17 repair methods have been identified. Among these, two types of failures are self-recoverable after a period of time, and four types of failures can each be addressed by two alternative repair methods.

3.6.2 Experiment Design

The failure report produced by simulation platform is monitored by a seperate thread in the produce every 0.1 second. Then corresponding solution information is appended and send to LLm agent.

In the digital simulation environment, certain operating parameters were hypothetically adjusted to create a more controlled experimental setting. The mean time to repair (MTTR) for each workstation was set to 30. The availability of each assembly station was assumed to be 60%, which is deliberately lower than the actual availability to simulate a stress-testing scenario. The target state was defined as achieving a reward value greater than 0.85.

Table 3.1 shows the main parameter settings for PPO, DDQN, and SAC.

PPO uses the PPO-Clip variant. The advantage function is implemented with GAE, which improves sample efficiency and stability. Entropy regularization is supported to encourage exploration and avoid getting stuck in local optima. Both the Actor and Critic are implemented as multi-layer fully connected neural networks. As an on-policy algorithm, PPO trains only on the most recently collected data. The implementation supports multiple epochs of updates to further improve data utilization.

DDQN adopts the Dueling Q-Network architecture, which separates the value and advantage streams to enhance the modeling of state values and action advantages. It maintains two neural networks: an online network and a target network, with weights periodically synchronized. An experience replay buffer is used to shuffle the sampling order and improve the independence of training samples. An epsilon-greedy exploration strategy is supported, with the exploration rate gradually decaying during training.

SAC uses a discrete-action variant, where the policy network outputs a probability distribution rather than mean and variance. There are two independent Critic net-

works, each with its own target network. The implementation features an automatic temperature parameter (alpha) adjustment mechanism, dynamically balancing reward maximization and policy entropy. The network structure consists of multi-layer fully connected neural networks. An experience replay buffer is used for storage and sampling, enabling off-policy training. The target networks are updated using a soft update strategy, which slowly synchronizes parameters to improve training stability.

Table 3.1: Comparison of main hyperparameters for PPO, DDQN, and SAC agents

Parameter	**PPO**	**DDQN**	**SAC (Discrete)**
Learning rate	alpha (1e-4, actor/critic)	lr (1e-5, Q-net)	alpha (1e-4, actor); beta (1e-4, critic)
Discount factor (γ)	0.99	0.99	0.99
Batch size	64	64	64
Buffer size	–	1,000,000	1,000,000
Target update	–	every 10 steps	τ (0.005, soft update)
Epsilon params	–	start: 1.0; end: 0.05; decay: 0.99	–
Policy clip	0.2	–	–
GAE lambda	0.95	–	–
Entropy coeff/target	0.01	–	target: $-\log(1/n)$; auto α
Network hidden size	256/256/256 (actor/critic)	256/256/256 (Q network)	256/256/256 (actor/critic)
Optimizer	Adam	Adam	Adam
Replay buffer	On-policy memory	Yes	Yes

The LLMs agent takes as input a DataFrame of historical failure records for several machine models, processes the latest 10 failure events for each model, and uses an LLM to assess the current stability of each model. The output is a list of numerical stability scores, one for each machine model under consideration. The tokenizer's padding token is set to the end-of-sequence (EOS) token, with left-side padding. The maximum number of tokens generated per inference call is limited to 1,000 in case excessive generation length. Inference is conducted using half-precision arithmetic (torch.float16) and sampling is disabled (do_sample=False), with all decoding proceeding via greedy selection to guarantee that identical inputs yield identical outputs. Figure 3.7 shows the prompt template used to in LLMs.

We conducted comprehensive comparative experiments on the selected LLM agents and DRL agents using both horizontal and vertical comparisons. The horizontal comparison evaluated the training performance of different DRL agents when

Prompt Template

```
{
    {
        "role": "system",
        "content": '"You are a systems engineer. Based on historical machine failure records and expert feedback,
            you must assess the current health of the machine and output a numerical stability score between 0.0 (very
            unstable) to 10.0 (very stable).\n
            Some guidance:\n
            - If the number of reports is too few to determine a trend, rely more on whether it can be automatically fixed and
            the human feedback to assess the condition.\n
            - If more failure reports exist, and the duration from failure start to failure end is getting shorter over time, the
            condition is becoming **more stable**, otherwise, it is becomming **less stable**\n
            - If the predicted time interval until the next failure occurs is getting **longer** as repair records accumulate, the
            machine is also becoming **more stable**.\n
            - If, as maintenance records accumulate, some records indicate that the predicted interval to the next failure has
            become significantly shorter compared to previous intervals, this suggests that the machine is also becoming
            **less stable**.\n
            Return the result in the following format: 'Output: The score I suggest is x.x', where 'x.x' is a **single float
            number** between 0.0 and 10.0. Pilease limit your response within 100 words.'"
    },
    {
        "role": "user",
        "content": f"Here is the failure history and feedback for {model_name}:\n{history_text}"
    }
}
```

Figure 3.7: Prompt template.

trained with the same LLM agent. In the vertical comparison, we introduced a baseline experiment that generates state representations based on a random algorithm in order to assess the training performance of the same DRL agent when paired with different LLM agents. Evaluation metrics include average reward per episode and speed of convergence. All of the agents' training parameters remained the same throughout these comparative tests.

4

Results

Our experiment was conducted in three phases: First, we implemented a simulation model in the digital twin layer to automatically generate fault reports in a simulated environment. Then, we investigated how to use LLMs to analyze these reports and produce usable digital representations. Finally, we tested three different DRL algorithms in a complete closed-loop experiment.

4.1 Generation of Maintenance Report from Simulation Model

Figure 4.1 shows the way we implemented the automatic export of fault reports. The lab currently has three machines, all generating error logs during simulation in the same standardized format. Each log entry includes: the faulty machine identifier, error type, fault start time, fault end time, and the platform's predicted time interval until the next occurrence of the same fault. Figure 4.2 displays a screenshot of the

Figure 4.1: The screenshot of the method to export the fault report.

output fault report. A Python daemon thread persistently monitors the target file, fetching incremental updates from the last recorded cursor position when new data becomes available. The system then automatically annotates the data with diagnostic causes and repair solutions, emulating human troubleshooting workflows. The resulting complete maintenance report is presented in Figure 4.3.

failures.csv > data

```
model_name,failure_profile,start_time,end_time,failure_interval
*.Models.Model.Assembly1,FailureA,1:25:26.3092,1:45:37.5492,1:00:00.0000
*.Models.Model.Assembly1,FailureA,2:07:47.5811,2:29:36.0375,1:00:00.0000
*.Models.Model.Assembly4,FailureA,2:25:44.5912,2:54:16.3028,3:36:37.8242
*.Models.Model.Assembly3,FailureA,2:50:46.1443,3:17:57.5290,24:06.3070
*.Models.Model.Assembly1,FailureA,3:08:47.7267,3:40:01.7915,1:00:00.0000
*.Models.Model.Assembly4,Failure1,3:33:06.5943,3:49:38.4411,3:36:37.8242
*.Models.Model.Assembly3,FailureA,3:42:10.3444,4:10:36.7210,17:51.4314
*.Models.Model.Assembly1,FailureA,3:49:12.0532,4:31:58.7694,1:00:00.0000
*.Models.Model.Assembly3,FailureA,4:37:31.1378,5:12:08.1544,26:40.4903
*.Models.Model.Assembly1,FailureA,4:35:26.4480,5:34:26.2129,1:00:00.0000
*.Models.Model.Assembly1,FailureA,5:37:44.2303,5:55:56.7815,1:00:00.0000
*.Models.Model.Assembly1,FailureA,6:08:50.2347,6:24:16.3377,1:00:00.0000
*.Models.Model.Assembly3,FailureA,6:41:28.9722,6:57:12.2429,3:13.1278
*.Models.Model.Assembly1,FailureA,6:45:09.5793,7:12:17.3459,1:00:00.0000
*.Models.Model.Assembly4,Failure1,7:05:00.0627,7:22:54.3940,3:36:37.8242
*.Models.Model.Assembly3,FailureA,7:00:26.9978,7:41:52.5114,40:25.2761
```

Figure 4.2: The screenshot of fault report.

ilures_updated.csv > data

```
model_name,failure_profile,start_time,end_time,failure_interval,failure_type,solution_method,is_fixable,repair_feedback
*.Models.Model.Assembly1,FailureA,1:25:26.3092,1:45:37.5492,1:00:00.0000,8,6,0,Recommendation: Schedule a comprehensive power supply u
*.Models.Model.Assembly1,FailureA,2:07:47.5811,2:29:36.0375,1:00:00.0000,13,5,0,Recommend replace the faulty actuator component as soo
*.Models.Model.Assembly4,FailureA,2:25:44.5912,2:54:16.3028,3:36:37.8242,2,15,1,Recommend: Implement a lubrication schedule to prevent
*.Models.Model.Assembly3,FailureA,2:50:46.1443,3:17:57.5290,24:06.3070,1,0,0,Recommend: Perform an immediate thorough diagnostic inves
*.Models.Model.Assembly1,FailureA,3:08:47.7267,3:40:01.7915,1:00:00.0000,7,10,0,Recommend check and replace the RFID sensor as soon as
*.Models.Model.Assembly4,Failure1,3:33:06.5943,3:49:38.4411,3:36:37.8242,12,4,0,Perform an immediate review of the machine's mechanica
*.Models.Model.Assembly3,FailureA,3:42:10.3444,4:10:36.7210,17:51.4314,5,8,1,Recommend: Perform immediate cooling system inspection an
*.Models.Model.Assembly1,FailureA,3:49:12.0532,4:31:58.7694,1:00:00.0000,10,3,0,Recommend conduct a thorough inspection of all connect
*.Models.Model.Assembly3,FailureA,4:37:31.1378,5:12:08.1544,26:40.4903,6,9,0,Recommendation: Perform an immediate thorough inspection
*.Models.Model.Assembly1,FailureA,4:35:26.4480,5:34:26.2129,1:00:00.0000,1,0,0,Immediately investigate and repair or replace the fault
*.Models.Model.Assembly1,FailureA,5:37:44.2303,5:55:56.7815,1:00:00.0000,11,1,1,Recommendation: Immediately implement regular monitori
*.Models.Model.Assembly1,FailureA,6:08:50.2347,6:24:16.3377,1:00:00.0000,7,10,0,Recommend perform a thorough inspection and calibratio
*.Models.Model.Assembly3,FailureA,6:41:28.9722,6:57:12.2429,3:13.1278,7,10,0,**Recommendation:** Check and replace the RFID sensor to
*.Models.Model.Assembly1,FailureA,6:45:09.5793,7:12:17.3459,1:00:00.0000,9,12,1,Recommend inspect and repair or replace the motor and
*.Models.Model.Assembly4,Failure1,7:05:00.0627,7:22:54.3940,3:36:37.8242,13,5,0,Recommendation: Perform a thorough inspection of the a
*.Models.Model.Assembly3,FailureA,7:00:26.9978,7:41:52.5114,40:25.2761,8,6,0,Recommend: Run a thorough diagnostic test to identify and
*.Models.Model.Assembly1,FailureA,7:46:44.7912,8:10:01.8407,1:00:00.0000,6,9,0,Monitor the machine's airflow pressure continuously to
*.Models.Model.Assembly3,FailureA,8:22:35.8790,8:39:01.0507,18:27.1201,11,1,1,Maintenance Recommendation: Run a thorough diagnostic te
*.Models.Model.Assembly3,FailureA,8:57:41.3758,9:15:43.9077,25:30.7564,13,5,0,Recommend: Conduct a thorough diagnostic analysis of the
```

Figure 4.3: The screenshot of maintenance report.

4.2 Output from Generative AI Layer

We employed three different LLM models for our experiments, all utilizing the smallest available versions of each model. This decision was made not only due to computational resource constraints but also because our task was relatively simple - the input data contained only 9 columns, and for each component we sampled only the most recent 10 entries for training. Even the smallest LLM models proved sufficient to validate our proposed framework. All three models were trained using identical parameters and prompts. For each training batch, we compiled historical data from the three machines into arrays containing three prompts. These were then tokenized collectively before being processed by the large language models for analysis. The expected output is shown in Figure 4.4.

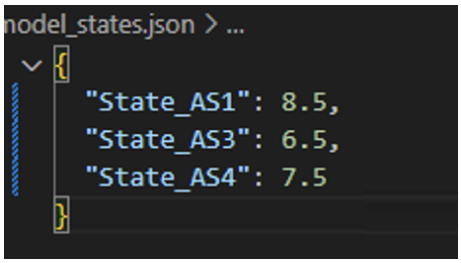

Figure 4.4: The output of LLMs.

4.3 Output from Deep Reinforcement Learning Layer

To evaluate the training performance of different DRL agents under the outputs of various LLMs, we also established a baseline model. This model generates machine state values based on experience for the DRL to read. Serving as a control, the baseline model helps quantify the performance differences of DRL agents influenced by different LLM outputs.

We compared their performances during training over 500 episodes, each with a maximum of 100 steps. We evaluated the average reward per episode, which reflects the overall performance and learning effectiveness, as well as the number of steps taken to reach the goal state in each episode, which indicates the convergence speed.

Experimental results show that, in the horizontal comparison experiments, the Phi model demonstrated significant advantages across all three DRL agents, with especially outstanding performance when paired with the PPO agent—the average reward during training was noticeably higher than with other models (as shown in Figure 4.9). In the vertical comparison experiments, PPO outperformed DDQN and SAC by a large margin in the maintenance prioritization task, achieving higher rewards and a more stable training process. After introducing the Phi model, PPO's training performance was further improved, with both the obtained rewards and the average number of steps required to reach the goal state becoming more optimal and stable, as shown in Figure 4.11. In contrast, DDQN and SAC agents did not show significant improvements in reward after integrating LLMs, and the overall gains were limited. By averaging the average reward, as shown in the Table 4.1, the combination of the PPO and Phi models achieves the best performance, with highest score 0.829.

Notably, all models demonstrate stable learning trajectories after the initial data-scarce phase. The observed early-stage volatility rapidly subsides, with policies converging to steady-state performance within 100 training epochs. This stabilization confirms that LLM integration effectively resolves the cold-start problem inherent in conventional DRL-digital twin systems.

	Llama-3.2-1b	Phi-3.5-mini	DeepSeek-R1-Distill-Qwen-1.5B
DDQN	0.817	0.818	0.816
SAC	0.785	0.788	0.783
PPO	0.819	0.829	0.818

Table 4.1: Performance comparison across different LLMs and RL algorithms.

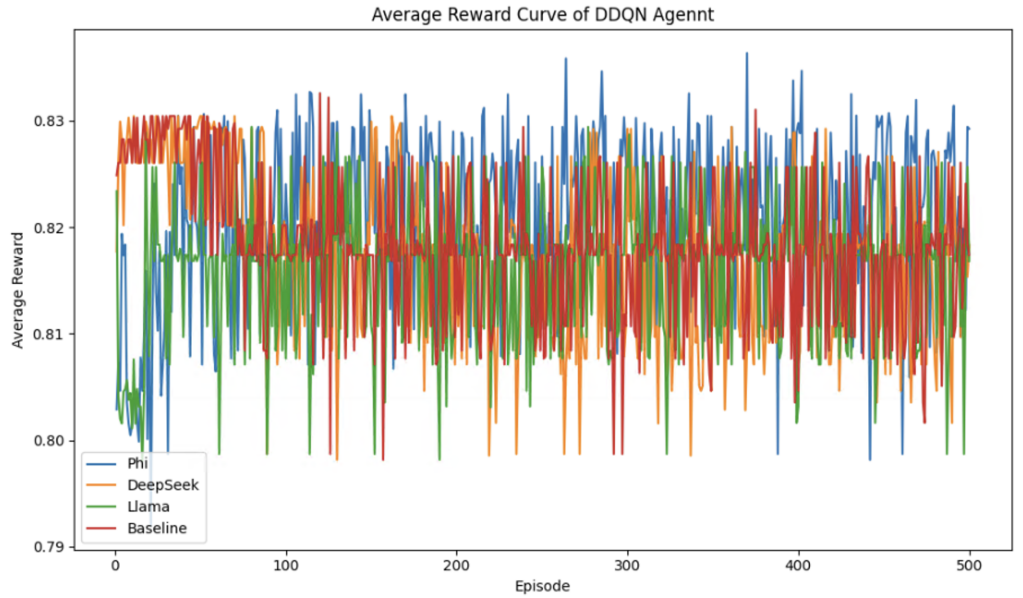

Figure 4.5: The figure shows the average reward and the train steps per episode during training for the DDQN agent when interacting with three different LLM models and the baseline model.

4.4 Reflection

This experiment aims to support prescriptive maintenance decisions in industrial environments by dynamically optimizing maintenance workflows through AI-driven insights. The DRL agent generates maintenance priority actions based on real-time LLM state representations and equipment operating conditions. After execution, the digital twin's reward function calculates a reward value reflecting the virtual model's performance post-action. Higher rewards indicate greater DRL learning efficiency, more reliable decision outcomes, and closer alignment with prescriptive maintenance objectives.

Experimental results show SAC maintained stable performance at 0.78 reward but gained no improvement from LLM-enhanced states, indicating its inability to leverage semantic representations. DDQN showed marginal gains when combined with the Phi model, yet lacked decisive advantages over baselines, suggesting LLMs only minimally enhanced its learning efficiency. In contrast, PPO demonstrated significant improvement when integrated with LLM state representations, particularly with the Phi model, achieving a substantially higher average reward of 0.829. This

Figure 4.6: The figure shows the number of steps per episode taken to reach the goal state during training for the DDQN agent when interacting with three different LLM models and the baseline model.

confirms PPO's policy-gradient architecture uniquely harnesses the semantic richness of LLM outputs, enabling faster convergence, more robust learning, and superior decision quality in complex industrial settings.

The three algorithms employed in this experiment demonstrated relatively stable average performance over 500 training episodes. This stability indicates that the selected hyperparameter configurations operate within robust operational ranges for each algorithm, avoiding extreme fluctuations due to minor adjustments.

For DDQN, key hyperparameters include the learning rate, target network update interval, and epsilon parameters governing exploration. A higher learning rate accelerates convergence but often causes significant reward volatility, while a lower rate enhances training stability at the cost of slower progress. The target network update frequency is equally critical: infrequent updates slow learning, whereas excessive updates induce instability. Epsilon parameters balance exploration and exploitation, slow decay prolongs suboptimal exploration, while rapid decay risks inadequate early exploration and persistent randomness, reducing average rewards.

In the discrete-action SAC implementation, sensitivity primarily manifests in the entropy target, temperature coefficient α, soft update coefficient τ, and learning rates for actor/critic networks. Misconfiguration of entropy targets or automatic entropy adjustment mechanisms may yield overly random policies or insufficient exploration. The soft update coefficient τ dictates how quickly target networks track primary networks, directly influencing training stability and adaptability.

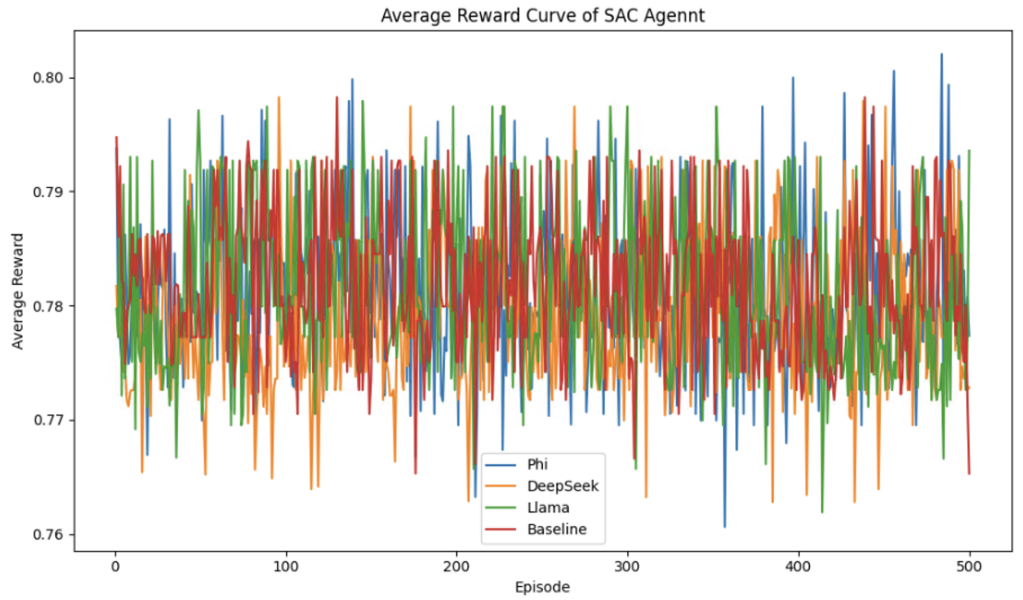

Figure 4.7: The figure shows the average reward during training for the SAC agent when interacting with three different LLM models and the baseline model.

PPO exhibits significant sensitivity to the policy clip range, entropy coefficient, learning rate, and training iterations per batch. The clip range constrains divergence between old and new policies to ensure stable updates; improper settings may suppress policy improvement or cause divergence. The entropy coefficient directly controls exploration: high values encourage exploration but hinder convergence, while low values promote premature exploitation of suboptimal policies. Learning rates and training iterations per batch require careful balancing to avoid underfitting or overfitting.

Despite the algorithms reporting similar average rewards in this experiment, we observed that even slight deviations from empirically optimal hyperparameter ranges could cause substantial performance degradation or training instability.

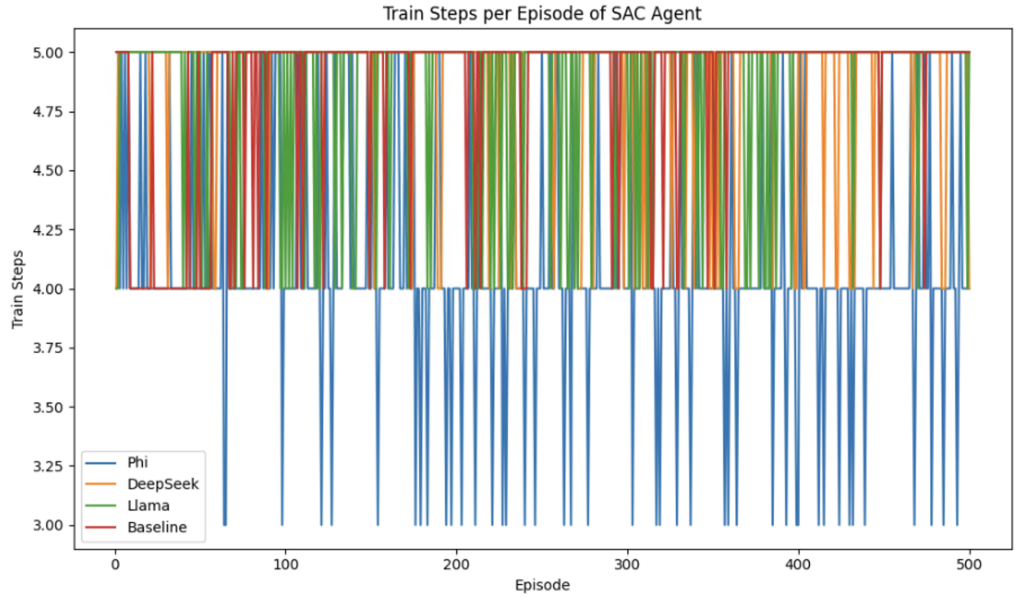

Figure 4.8: The figure shows the number of steps per episode taken to reach the goal state during training for the SAC agent when interacting with three different LLM models and the baseline model.

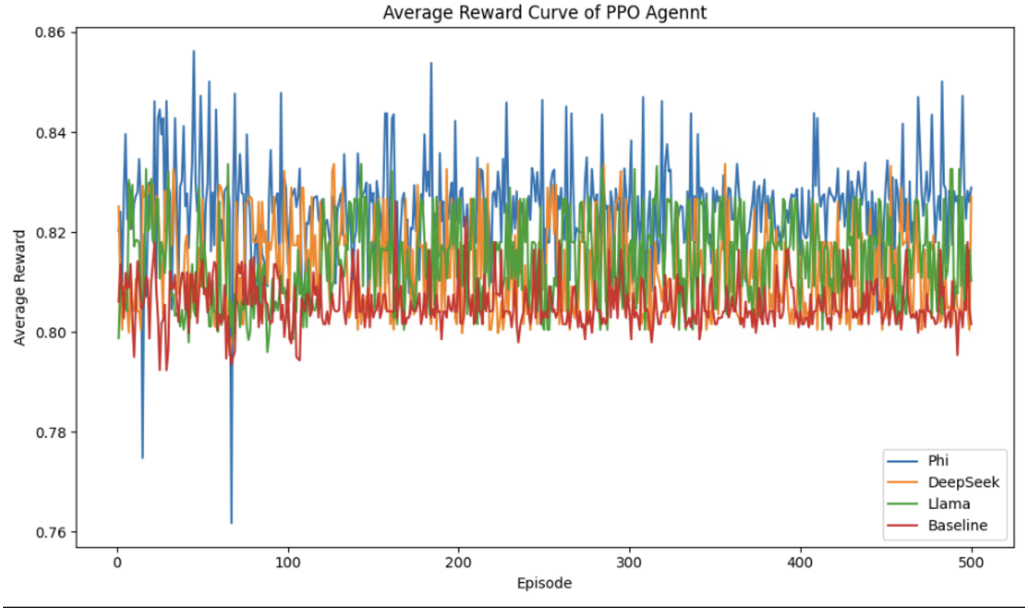

Figure 4.9: The figure shows the average reward during training for the PPO agent when interacting with three different LLM models and the baseline model.

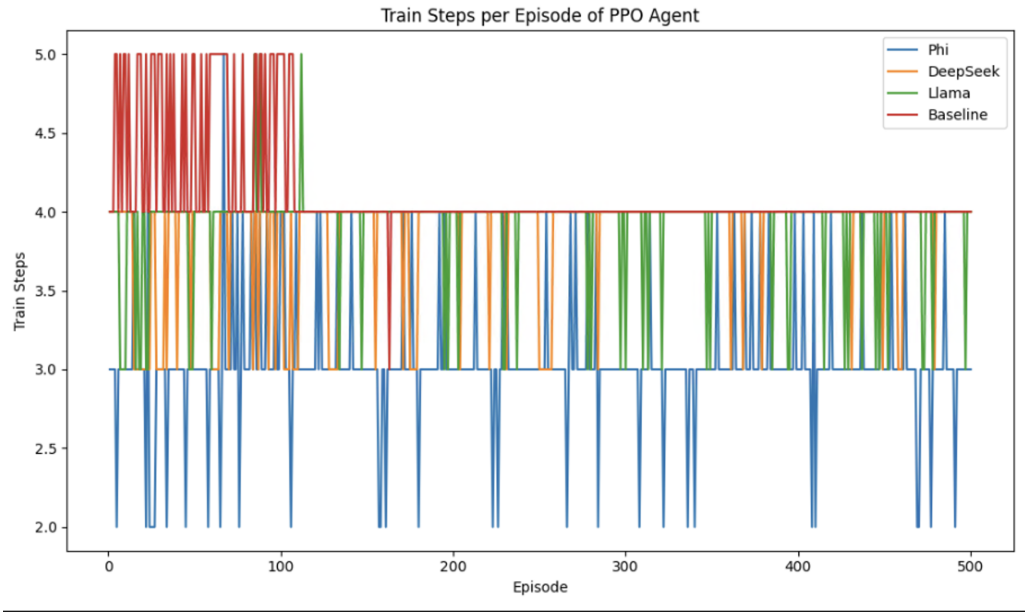

Figure 4.10: The figure shows the number of steps per episode taken to reach the goal state during training for the PPO agent when interacting with three different LLM models and the baseline model.

Figure 4.11: The figure shows the average reward during training for the Phi model when interacting with three different DRL agents.

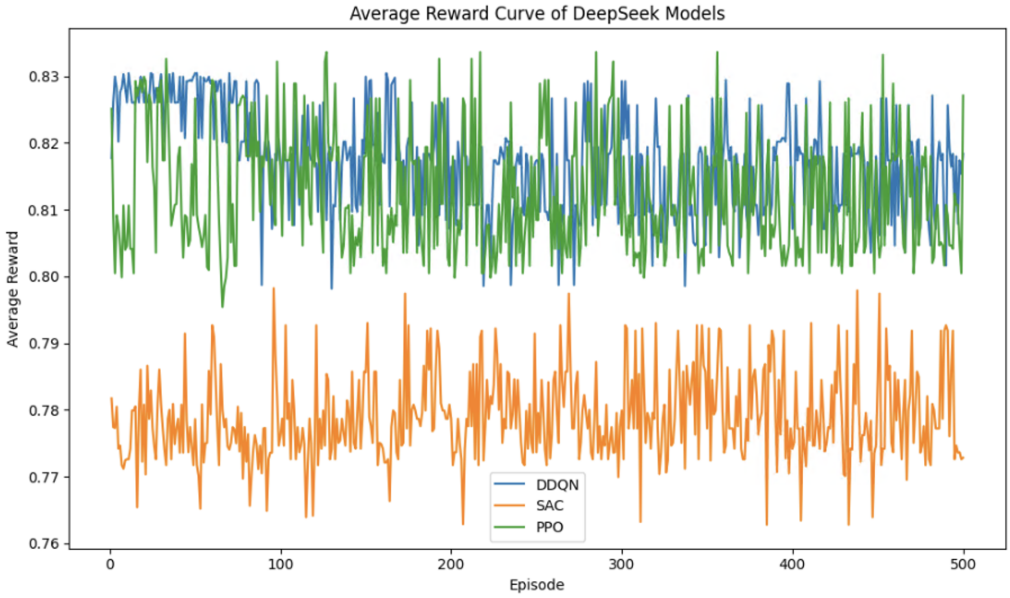

Figure 4.12: The figure shows the average reward during training for the DeepSeek model when interacting with three different DRL agents.

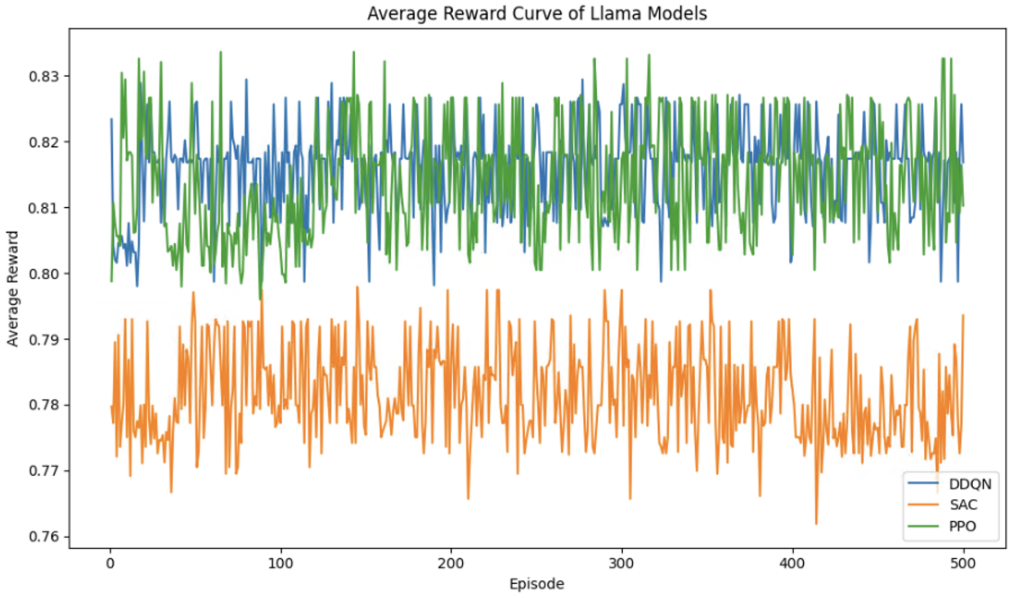

Figure 4.13: The figure shows the average reward during training for the Llama model when interacting with three different DRL agents.

5

Discussion

5.1 Answers to Research Question

RQ 1: How can LLMs convert real-time maintenance reports into valuable state representations suitable for deep reinforcement learning algorithms?

In our experimental scenario, the data includes both structured fields (such as equipment start time, end time, and operational status—which require the LLM to possess certain logical reasoning capabilities) and unstructured textual feedback from frontline engineers and maintenance experts. Directly inputting structured data into the LLM often results in inaccurate comprehension of field relationships and may even cause parsing errors when processing formats like CSV. To address this issue, we adopted a schema-guided prompting method to preprocess and format input data, enhancing the LLM's ability to understand structured information and improving output accuracy.

To ensure the reliability of decision-making, we first standardize the raw data and clearly define the criteria for performance evaluation. We then reconstruct the structured data into natural language prompts—for example, converting the "start time" and "end time" fields into descriptions like "The equipment malfunctioned at [time] and was repaired at [time]." These natural language prompts are then combined with synthetic data generated by the virtual simulation platform (such as "predicted time to next failure") and input into the LLM. Then LLM must accomplish two key tasks: first, to reason about the variation trends in the time intervals between start and end times across different records, thereby assisting in the assessment of equipment operating cycles and fault patterns; and second, to extract critical information from expert notes and other unstructured texts, such as hardware fault descriptions or abnormal alarms, further determining whether the equipment is in a stable or abnormal state.

The LLM-empowered state representation method we adopt theoretically aligns closely with the LESR (LLM-Empowered State Representation) framework discussed in related literature[78]. Both approaches address the challenge of transforming heterogeneous industrial data into representations usable by reinforcement

learning agents. LESR pioneered the autonomous generation of state encodings and intrinsic rewards using language models, while our schema-guided prompting technique builds upon this foundation with customized extensions tailored to the specific data constraints of manufacturing. The core commonality between the two lies in viewing LLMs as 'semantic interpreters' rather than mere feature extractors, bridging the gap between raw industrial data and decision-ready representations.

RQ 2: How do different DRL models perform when trained with high-dimensional state representations?

Compared to prior research where PPO integrated with digital twins achieved an average reward of 0.6[1], our experimental results demonstrate significantly superior performance. The LLM-empowered PPO-digital twin system consistently attained average rewards exceeding 0.8. These findings further validate that PPO's on-policy, policy-gradient architecture uniquely leverages LLM-enhanced state representations to deliver exceptional outcomes in high-dimensional industrial environments.

In our experiments, we additionally implemented two other algorithms, DDQN and SAC, which are off-policy methods, for comparative analysis. PPO consistently outperformed both DDQN and SAC, which aligns with the TranDRL framework's conclusions where PPO similarly emerged as the optimal algorithm for LLM-enhanced reinforcement learning[77]. This superiority stems from PPO's on-policy, policy-gradient methodology, which updates policies exclusively using the most recently collected data. This enables superior adaptation to high-dimensional, discrete, and dynamically changing state spaces. Conversely, value-based off-policy methods like DDQN and SAC are more susceptible to value estimation bias and training instability in high-dimensional environments.

Additionally, the Phi model with 3.82B parameters demonstrates significantly better training performance compared to the 1.24B-parameter llama-3.2-1b model and the 1.78B-parameter DeepSeek-R1-Distill-Qwen-1.5B model. This advantage is mainly due to Phi's larger parameter scale, as well as its use of a high-quality, large-scale training dataset (up to 3.3 trillion tokens), and rigorous data selection and multi-stage optimization during training and safety alignment. As a result, the Phi model delivers superior training and inference performance.

Experimental results demonstrate that the Phi model provides PPO with state inputs richer in semantic information and structural features, enabling faster and more accurate learning of optimal policies. However, the study did not individually optimize DRL hyperparameters for different LLMs, which would be a significant limitation given reinforcement learning's sensitivity to parameter configuration. This lack of personalized tuning risks underestimating the capabilities of larger LLMs[89].

5.2 Contribution of This Study

This study explores methods for LLM-empowered DRL state representation. By LLMs, the proposed approach efficiently transforms multi-dimensional machine states, including both structured operational parameters and unstructured human feedback, into semantically rich state representations interpretable by DRL agents. This approach significantly enhances DRL learning efficiency, addressing issues of insufficient state space representation, unreliable learning outcomes, and cold-start problems. The introduction of LLMs effectively bridges the gap between historical data, expert knowledge, and executable maintenance strategies, providing a feasible exploration for truly smart decision-making systems.

In comparative experiments with different types of DRL models, we found that on-policy methods like PPO perform better in high-dimensional, semantically complex state spaces. PPO can continuously optimize its policy using the most recently collected data, allowing it to better adapt to the dynamic state distributions generated by LLMs, resulting in more stable training processes and higher reward scores. In contrast, off-policy methods such as DDQN and SAC are more susceptible to value estimation bias and slower adaptation, making them less effective than PPO in complex environments. In addition, lightweight LLMs like Phi, which have more training parameters, can provide PPO with richer and more accurate state representations, further enhancing PPO's performance in complex tasks.

5.3 Limitation

First, the LLMs selected in our current experiments are lightweight versions with a limited number of parameters. Although previous research has shown that larger models possess stronger knowledge representation and reasoning abilities, due to resource constraints, we have not yet verified whether DRL's learning efficiency and decision accuracy would be further enhanced with larger-scale LLMs (such as those with billions or tens of billions of parameters) that have greater knowledge reserves[90].

Second, due to computational and platform limitations, the number of DRL training episodes in this study is relatively small and does not cover larger-scale training scenarios. Previous literature has pointed out that sample efficiency and training scale have a significant impact on DRL performance [91]. In the future, it would be valuable to increase the number of training episodes to 100,000 or more to observe whether the average return and policy convergence speed can be further improved, providing a more comprehensive evaluation of the sample efficiency and generalization ability of the integrated model.

Third, our experiments did not perform hyperparameter optimization specifically tailored for different LLM architectures within the DRL framework. Given reinforcement learning's well-established sensitivity to parameter configurations—where even

minor adjustments can cause up to a 50% reward reduction in LLM-enhanced systems, this oversight risks weakening knowledge retention capabilities[89]. However, since the experiments primarily used small models with similar parameter scales, the resulting performance impact was not substantial. Nevertheless, when integrating larger LLMs (such as 70B+ parameter models), which fundamentally differ from lightweight architectures like Phi in gradient complexity, output stability, and representation density, personalized tuning of learning rates, entropy coefficients, and clipping thresholds becomes essential.

5.4 Future Work

The representation of action spaces will emerge as one of the primary research directions in the future. This innovative approach can effectively address the action dimensionality challenges faced by traditional reinforcement learning in complex industrial scenarios, while providing new theoretical frameworks and technical pathways for optimizing intelligent maintenance systems.

Furthermore, the integration of multi-agent collaborative architectures enables effective handling of complex decision-making problems involving multi-equipment coordination in production lines. Within such frameworks, each agent specializes in learning action representations for specific equipment, while achieving information exchange through dedicated communication protocols. This allows for intelligent maintenance operations in sophisticated manufacturing environments.

Finally, incorporating Retrieval-Augmented Generation (RAG) technology facilitates the development of specialized knowledge bases for maintenance prioritization decisions. These knowledge repositories further enhance LLMs' capability to generate more precise and accurate state-action representations.

6

Conclusion

Given the exceptional capabilities of LLMs in processing textual data, combined with the widespread adoption of DRL and digital twin technologies in intelligent manufacturing, this paper proposes an LLM-driven DRL digital twin system solution. This approach effectively integrates unstructured expert feedback with structured equipment operational data, transforming them via LLM into state representation data usable by DRL, successfully addressing the challenge faced by traditional DRL digital twin systems that cannot directly process unstructured human feedback data.

Based on our experimental results, on-policy methods like PPO demonstrate outstanding performance in high-dimensional, semantically complex state spaces, especially when integrated with high-parameter, lightweight LLMs such as Phi, achieving faster convergence and better decision-making outcomes. The proposed approach not only effectively utilizes real-time data and human feedback to achieve excellent decision quality but also addresses the cold-start problem faced by traditional DRL-digital twin systems, enabling rapid convergence and stable learning processes.

The LLM-DRL integrated solution proposed in this study can be further validated in the future by incorporating larger-scale, multi-source industrial data from more complex environments. This will verify the system's decision-making robustness and generalization capabilities in actual dynamic industrial settings. Simultaneously, systematic exploration of LLMs with different parameter scales can be conducted to assess their impact on state representation quality and intelligent decision-making, while investigating the relationship between model complexity and performance improvement. Building on the excellent results achieved in state representation, future work could leverage LLMs' natural language generation capabilities to expand the maintenance action space. When combined with LLM-driven adaptive reward mechanisms, this enables dynamic policy optimization. Furthermore, by integrating domain-specific knowledge graphs, the system can generate more fine-grained and targeted fault resolution solutions, thereby enhancing decision reliability and transparency.

Bibliography

[1] S. Chen, P. V. Lopes, S. Marti, M. Rajashekarappa, S. Bandaru, C. Windmark, J. Bokrantz, and A. Skoogh, "Enhancing digital twins with deep reinforcement learning: A use case in maintenance prioritization," in *Proceedings of the Winter Simulation Conference*, ser. WSC '24. IEEE Press, 2025, p. 1611–1622.

[2] Information is Beautiful, "The rise of generative ai large language models (llms) like chatgpt," https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/, 2024, accessed: 2025-05-22.

[3] B. A. Adaramola, J. F. Kayode, S. I. Monye, and S. A. Afolalu, "Overview of maintenance management strategies in the industry," in *2024 International Conference on Science, Engineering and Business for Driving Sustainable Development Goals (SEB4SDG)*, 2024, pp. 1–11.

[4] B. Meindl and J. Mendonça, "Mapping industry 4.0 technologies: From cyber-physical systems to artificial intelligence," 2021. [Online]. Available: https://arxiv.org/abs/2111.14168

[5] Z. Huang, Y. Shen, J. Li, M. Fey, and C. Brecher, "A survey on ai-driven digital twins in industry 4.0: Smart manufacturing and advanced robotics," *Sensors (Basel, Switzerland)*, vol. 21, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:238639686

[6] Q. Hu, Y. Zhao, Y. Wang, P. Peng, and L. Ren, "Remaining useful life estimation in prognostics using deep reinforcement learning," *IEEE Access*, vol. 11, pp. 32 919–32 934, 2023.

[7] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," 2022. [Online]. Available: https://arxiv.org/abs/2206.07682

[8] J. Lee and H. Su, "A unified industrial large knowledge model framework in industry 4.0 and smart manufacturing," *International Journal of AI for Materials and Design*, vol. 1, no. 2, p. 41, Jul. 2024. [Online]. Available:

http://dx.doi.org/10.36922/ijamd.3681

[9] R. Aghaei, A. A. Kiaei, M. Boush, J. Vahidi, Z. Barzegar, and M. Rofoosheh, "The potential of large language models in supply chain management: Advancing decision-making, efficiency, and innovation," 2025. [Online]. Available: https://arxiv.org/abs/2501.15411

[10] W. Jiang, Y. Zhang, Q. Li, and X. Wang, "Leveraging large language models for enhanced digital twin modeling: Trends, methods, and challenges," *arXiv preprint arXiv:2503.02167*, 2025. [Online]. Available: https://arxiv.org/abs/2503.02167

[11] J. Leng, H. Zhang, D. Yan, Q. Liu, X. Chen, and D. Zhang, "Digital twin-driven manufacturing cyber-physical system for parallel controlling of smart workshop," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 1155 – 1166, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:67333956

[12] K. Y. H. Lim, P. Zheng, and C.-H. Chen, "A state-of-the-art survey of digital twin: techniques, engineering product lifecycle management and business innovation perspectives," *J. Intell. Manuf.*, vol. 31, no. 6, p. 1313–1337, Aug. 2020. [Online]. Available: https://doi.org/10.1007/s10845-019-01512-w

[13] R. Rayhana, L. Bai, G. G. Xiao, M. Liao, and Z. Liu, "Digital twin models: Functions, challenges, and industry applications," *IEEE Journal of Radio Frequency Identification*, vol. 8, pp. 282–321, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:269122652

[14] S. I. Monye, S. A. Afolalu, S. L. Lawal, O. A. Oluwatoyin, and A. G. Adeyemi, "Overview and impact of maintenance process in 4th industrial revolution," *E3S Web of Conferences*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263823594

[15] WorkTrek, "7 types of reactive maintenance," https://worktrek.com/blog/reactive-maintenance-types/, 2025, accessed: 2025-05-22.

[16] Y. Ran, X. Zhou, P. Lin, Y. Wen, and R. Deng, "A survey of predictive maintenance: Systems, purposes and approaches," *ArXiv*, vol. abs/1912.07383, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:209376485

[17] A. C. D. Peña, "Optimization of preventive maintenance procedures for aircraft systems in an educational laboratory setting," *International Journal of Research and Scientific Innovation*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:273301519

[18] A. P. Kane, A. S. Kore, A. N. Khandale, S. S. Nigade, and P. P. Joshi, "Predictive maintenance using machine learning," 2022. [Online]. Available: https://arxiv.org/abs/2205.09402

[19] A. Guillaume, C. Vrain, and E. Wael, "Predictive maintenance on event logs: Application on an atm fleet," 2021. [Online]. Available: https://arxiv.org/abs/2011.10996

[20] H. H. Hansen, M. Kulahci, and B. F. Nielsen, "A primer on predictive maintenance: Potential benefits and practical challenges," *Quality Engineering*, vol. 36, pp. 638 – 649, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:268698600

[21] R. Shahriar, "Towards a cyber-physical manufacturing cloud through operable digital twins and virtual production lines," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226624200

[22] C. P. X. P. Isaac, "Cloud digital twins: Redefining enterprise infrastructure management with predictive analytics and automation," *World Journal of Advanced Engineering Technology and Sciences*, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:278024523

[23] H. Mao, Z. Liu, C. Qiu, Y. Huang, and J. Tan, "Prescriptive maintenance for complex products with digital twin considering production planning and resource constraints," *Measurement Science and Technology*, vol. 34, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260643795

[24] S. Chen, S. Bandaru, S. Marti, E. Turanoglu Bekar, and A. Skoogh, "Comparison of unsupervised image anomaly detection models for sheet metal glue lines," *Engineering Applications of Artificial Intelligence*, vol. 153, p. 110740, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197625007407

[25] Q. Lorente, É. Villeneuve, C. Merlo, G. A. Boy, and F. Thermy, "Development of a digital twin for collaborative decision-making, based on a multi-agent system: application to prescriptive maintenance," *INCOSE International Symposium*, vol. 32, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246582710

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[27] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol.

529, no. 7587, pp. 484–489, 2016.

[30] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[31] Y. Zhang, S. Wang, C. Wang, and J. Li, "Deep reinforcement learning for intelligent manufacturing: A review," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6523–6534, 2020.

[32] A. Kendall, J. Hawke, D. Janz, J. Mazur, I. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[33] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[34] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:59872172

[35] R. S. Sutton, C. Szepesvari, and H. R. Maei, "A convergent o(n) temporal-difference algorithm for off-policy learning with linear function approximation," in *Neural Information Processing Systems*, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:46601168

[36] J. A. Boyan, "Least-squares temporal difference learning," in *International Conference on Machine Learning*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:6988226

[37] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, no. 3, pp. 321–355, 1988.

[38] V. Mnih and K. Kavukcuoglu, "Methods and apparatus for reinforcement learning," Jun. 13 2017, uS Patent 9,679,258.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:28695052

[40] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *International conference on machine learning*, pp. 1928–1937, 2016.

[41] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290

[42] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient meth-

ods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[43] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[44] R. Bellman, "Dynamic programming," *Science*, vol. 153, pp. 34 – 37, 1957. [Online]. Available: https://api.semanticscholar.org/CorpusID:271544899

[45] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988. [Online]. Available: https://api.semanticscholar.org/CorpusID:3349598

[46] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.06461

[47] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016. [Online]. Available: https://arxiv.org/abs/1511.06581

[48] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992. [Online]. Available: https://api.semanticscholar.org/CorpusID:19115634

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[50] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[51] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[52] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[53] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[54] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," in *Journal of Machine Learning Research*, vol. 3, 2003, pp. 1137–1155.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[56] OpenAI, "Chatgpt: Optimizing language models for dialogue," https://openai.com/blog/chatgpt, 2022, accessed: 2025-05-22.

[57] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[58] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI Blog*, 2018, https://www.openai.com/research/language-unsupervised.

[59] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[60] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[61] OpenAI, "Gpt-4 technical report," https://cdn.openai.com/papers/gpt-4.pdf, 2023, accessed: 2025-05-22.

[62] A. Chowdhery, S. Narang, J. Devlin *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[63] R. Anil *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[64] G. DeepMind, "Gemini: A family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2024.

[65] H. Touvron, L. Martin, K. Stone *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[66] M. AI, "Introducing meta llama 3: The most capable openly available llm," https://ai.meta.com/blog/meta-llama-3/, 2024, accessed: 2025-05-22.

[67] I. Bhatia *et al.*, "Ai governance and accountability: An analysis of anthropic's claude," *arXiv preprint arXiv:2407.01557*, 2024.

[68] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, H. Zhong, Y. Zhu, M. Yang, Z. Li, J. Wan, P. Wang, W. Ding, Z. Fu, Y. Xu, J. Ye, X. Zhang, T. Xie, Z. Cheng, H. Zhang, Z. Yang,

H. Xu, and J. Lin, "Qwen2.5-vl technical report," 2025. [Online]. Available: https://arxiv.org/abs/2502.13923

[69] D. Research, "Deepseek llm: Scaling open-source language models with longtermism," *arXiv preprint arXiv:2401.02954*, 2024.

[70] H. Lu, W. Liu, B. Zhang, B. Wang, K. Dong, B. Liu, J. Sun, T. Ren, Z. Li, H. Yang, Y. Sun, C. Deng, H. Xu, Z. Xie, and C. Ruan, "Deepseek-vl: Towards real-world vision-language understanding," 2024. [Online]. Available: https://arxiv.org/abs/2403.05525

[71] O. Kilic *et al.*, "Generative ai in academic writing: A comparison of deepseek, qwen, chatgpt, gemini, llama, mistral, and gemma," *arXiv preprint arXiv:2503.04765*, 2025.

[72] L. A. Jimenez-Roa, T. D. Simão, Z. Bukhsh, T. Tinga, H. Molegraaf, N. Jansen, and M. Stoelinga, "Maintenance strategies for sewer pipes with multi-state degradation and deep reinforcement learning," *PHM Society European Conference*, vol. 8, no. 1, p. 14, Jun. 2024. [Online]. Available: http://dx.doi.org/10.36001/phme.2024.v8i1.4091

[73] J. Sresakoolchai and S. Kaewunruen, "Railway infrastructure maintenance efficiency improvement using deep reinforcement learning integrated with digital twin based on track geometry and component defects," *Scientific Reports*, vol. 13, no. 1, p. 2439, feb 2023. [Online]. Available: https://doi.org/10.1038/s41598-023-29526-8

[74] L. Lemner, L. Wahlgren, G. Gay, N. Mohammadiha, J. Liu, and J. Wennerberg, "Exploring the integration of large language models in industrial test maintenance processes," 2024. [Online]. Available: https://arxiv.org/abs/2409.06416

[75] M. A. Arshad, A. Riaz, R. Fatima, and A. Yasin, "Sevpredict: Exploring the potential of large language models in software maintenance," *AI*, vol. 5, no. 4, pp. 2739–2760, 2024. [Online]. Available: https://www.mdpi.com/2673-2688/5/4/132

[76] M. Klekowicki, G. Szymański, M. Waligórski, and W. Misztal, "Application of large language models in diagnostics and maintenance of aircraft propulsion systems," *Advances in Science and Technology Research Journal*, vol. 19, pp. 304–320, 2024. [Online]. Available: https://doi.org/10.12913/22998624/196264

[77] Y. Zhao, J. Yang, W. Wang, H. Yang, and D. Niyato, "Trandrl: A transformer-driven deep reinforcement learning enabled prescriptive maintenance framework," 2024. [Online]. Available: https://arxiv.org/abs/2309.16935

[78] H. Yu *et al.*, "Llm-empowered state representation for reinforcement learning,"

arXiv preprint arXiv:2407.13237, 2024.

[79] Y.-G. Kwon *et al.*, "Latent reward: Llm-empowered credit assignment in episodic reinforcement learning," *arXiv preprint arXiv:2312.06718*, 2023.

[80] F. Lotfi, H. Rajoli Nowdeh, and F. Afghah, "Llm-augmented deep reinforcement learning for dynamic o-ran network slicing," 01 2025.

[81] R. BELLMAN, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[82] J.-E. Ståhl and C. Windmark, *Hållbara Produktionssystem.* Lund, Sweden: Studentlitteratur AB, 2022, artikelnummer: 39931-01.

[83] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, 2015, arXiv:1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980

[84] M. Pirotta, M. Restelli, and L. Bascetta, "Adaptive step-size for policy gradient methods," in *Neural Information Processing Systems*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:18363270

[85] Y. Wang, H. He, X. Tan, and Y. Gan, "Trust region-guided proximal policy optimization," *ArXiv*, vol. abs/1901.10314, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:59336220

[86] N.-C. Huang, P.-C. Hsieh, K.-H. Ho, and I.-C. Wu, "Ppo-clip attains global optimality: Towards deeper understandings of clipping," *ArXiv*, vol. abs/2312.12065, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:266362674

[87] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. D. Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H. S. Behl, X. Wang, S. Bubeck, R. Eldan, A. T. Kalai, Y. T. Lee, and Y. Li, "Textbooks are all you need," 2023. [Online]. Available: https://arxiv.org/abs/2306.11644

[88] J. C. e. a. Fernández, "Analyzing the evolution and maintenance of ml models on hugging face," *arXiv:2311.13380*, 2023.

[89] S. Ghosh, C. K. R. Evuru, S. Kumar, R. S, D. Aneja, Z. Jin, R. Duraiswami, and D. Manocha, "A closer look at the limitations of instruction tuning," 2024. [Online]. Available: https://arxiv.org/abs/2402.05119

[90] Z. Hou, P. Du, Y. Niu, Z. Du, A. Zeng, X. Liu, M. Huang, H. Wang, J. Tang, and Y. Dong, "Does rlhf scale? exploring the impacts from data, model, and method," 2024. [Online]. Available: https://arxiv.org/abs/2412.06000

[91] F. E. Dorner, “Measuring progress in deep reinforcement learning sample efficiency,” 2021. [Online]. Available: https://arxiv.org/abs/2102.04881