ML Pipeline Insights Service for Rule-Based Assessment of Training Practices in Reinforcement Learning

Evangelos Ntentos[*][0000−0002−7997−905X], Francesco Urdih[†][0009−0000−3507−5043], and Uwe Zdun[*][0000−0002−6233−2591]

[*]Faculty of Computer Science, Research Group Software Architecture, University of Vienna, Vienna, Austria
[†]UniVie Doctoral School Computer Science (DoCS), University of Vienna, Vienna, Austria
firstname.lastname@univie.ac.at

Abstract. As artificial intelligence continues to advance, Reinforcement Learning (RL) has established itself as a core approach for developing intelligent agents that make decisions over time. As RL systems grow in complexity, the need for standardized training practices becomes critical. This paper introduces a rule-based assessment approach to enforce best practices in RL training. We define a comprehensive set of architectural rules focused on RL pipeline practices, models versioning, multi-agents deployment and managing models in inference. Our methodology integrates Large Language Models (LLMs) and custom-based code detectors to ensure compliance with these best practices across diverse RL systems. We developed a *ML pipeline insights service* to automatically validate RL training practices directly from the source code. We validate our approach by applying it in a large-scale industrial case study and sixteen open-source case studies. Our evaluation showed that custom-based detectors achieved near-perfect precision and recall ($F_1 \approx 0.98$), while LLM-based detectors provided scalable validation with moderate F_1 scores (0.67–0.71), demonstrating the hybrid approach's strength in balancing accuracy and automation. The results demonstrate our tool's accuracy in identifying and enforcing best practices with high precision and recall rates, highlighting its practical applicability and automation feasibility.

Keywords: Reinforcement Learning, Best Practices, Machine Learning, Architecture Rules, Case Studies

1 Introduction

Reinforcement Learning has emerged as a major paradigm for training intelligent agents to make sequential decisions in the dynamic field of artificial intelligence and machine learning. As RL architectures become increasingly complex and scalable, practices regarding training strategies and their impact on software architecture grow more intricate. Several studies have aimed to document patterns and best practices in training strategies within RL architectures [9,1,11].

Reliable RL pipelines go beyond training, they require careful planning for development, management, and deployment [2]. Key stages include data collection, training, evaluation, and deployment, with automation ensuring efficiency and consistency. In

RL, frequent model updates make version tracking essential for reproducibility and collaboration.

Multi-agent reinforcement learning (MARL) adds complexity due to agent interactions. Developers address this with modular pipelines, communication strategies, and tools like graph neural networks for coordination and hierarchical frameworks for scalable learning [21,14].

To address the complexities and variations in RL training practices, we propose a method, based on our previous approach [13], which investigates how well LLMs can identify patterns and practices, to automatically evaluate RL training-related practices directly from source code. This involves expanding our prior set of architectural rules to a comprehensive collection of thirty-two rules and the corresponding detectors. The framework is designed as an ML pipeline insights service, leveraging a service-based architecture to provide real-time insights into ML training and inference processes.

To validate our approach's applicability and the performance of the defined rules, we performed a large-scale industrial case study of applying RL in a cyber-physical system for production automation and sixteen open-source case studies of RL systems. The industrial case study uses the insights service to observe multiple CI/CD pipelines across production facilities. This comprehensive evaluation allows us to assess the practical implementation of our rules across diverse RL environments and training practices, demonstrating our approach's practical applicability and automation feasibility.

The paper is organized as follows: Section 2 introduces RL practices. The methodology and specifics of automatic rule checking are discussed in Section 3. Case studies are presented in Section 4. The quantitative evaluation and case study are discussed in Section 5. Section 6 interprets the results, while Section 7 addresses potential validity threats. Related work is reviewed in Section 8, and Section 9 concludes the paper.

2 Background on Reinforcement Learning Practices

This section provides background information on the RL practices in the focus of this work.

RL Pipeline Implementation To ensure everything runs properly, it is important to follow a certain process: starting with collecting the right data, then training the model, testing its performance, and finally putting it into action. By following these steps, RL workflows are more scalable, easy to reproduce, and dependable [2]. One key part of building an effective pipeline is automating the repetitive tasks.

Model Versioning A key practice is that in RL versioning models is used to ensure that experiments are reproducible, changes are traceable, and collaborative development runs smoothly [8]. Since RL models are frequently updated in response to new data or policy changes, having a structured way to track different versions becomes very important. Version control tools such as Git make it easier to log changes to models and training data, which is critical for comparing performance across different iterations [2].

Multi-Agent Aspects and Deployment Practices MARL adds an extra layer of complexity because it involves several autonomous agents interacting within the same environment. To make MARL work well in practice, it needs certain strategies for things such as coordination, communication, and scaling, especially since the environment can change over time and agents often do not have full visibility of it [21]. One of the biggest factors in successful MARL systems is strong communication between agents. Recent techniques using graph neural networks [12] have advanced the field by modeling how agents relate to each other, which helps improve cooperation in tasks like multi-agent collaborations.

Managing Models in Inference Managing models efficiently during inference is critical when RL systems are deployed in the real world scenario. For maintaining speed and responsiveness, recent practices focus on reducing the computation, for instance, using sparse computation techniques [3]. These practices help RL models make quick decisions even when resources are limited. Another aspect that is important to understand is why a model makes a certain decision. This is relevant to interpretability. By linking RL with probabilistic inference frameworks, it can make the decision-making process more transparent [10].

3 Approach

This section describes the research methods followed in this study and our rule-based assessment approach to ensure best practices. The data used in and produced as part of this study have been made available online for reproducibility[1].

3.1 Research Methods

We initially reviewed various knowledge sources on RL-specific best practices, including practitioner books, blogs, scientific literature ([18,16,19,4,7]), and open-source repositories. Subsequently, we conducted a qualitative analysis using Grounded Theory (open and axial coding), to analyze the collected data and extract relevant practices. We then formulated thirty-two rules and developed two types of source code detectors, custom-based and LLM-based detectors, to automate compliance with each practice described in Section 2. Next, we selected a number of case studies, analyzed them, and used for evaluation. The main steps after analysis are:

Architectural Rule Definitions We defined key rules to improve RL training reliability and performance across four areas: pipeline design, model versioning, multi-agent deployment, and inference. Pipeline rules ensure modularity and reproducibility. Versioning rules promote traceable, automated model management using registries and tools like Kubernetes or Helm. For multi-agent setups, rules address coordination, scalability, and conflict handling. Inference rules support versioned models, and real-time monitoring.

[1] https://doi.org/10.5281/zenodo.15487181

Development of Detectors and Insights Service In the next step, we developed a prototype tool with source code detectors and an insights service for automating the validation of RL training practices to be in compliance with the specified rules. This includes writing advanced code detectors that met the rules requirements. The detectors were integrated into the RL systems, enabling them to parse source code and generate outputs in JSON format with the detected patterns and compliance values.

Application to Case Studies and Validation The final step of our approach involves the usage of the insights service for evaluating different RL systems. This involved thorough validation in different scenarios, including one industrial case study and sixteen open-source RL systems. This evaluation showed the weaknesses and strengths of the RL training practices.

3.2 Architecture of the ML Pipeline Insights Service

The *ML pipeline insights service* architecture consists of several key modules, each playing a crucial role in the analysis and validation process. The major components of the architecture are as follows: The *Service* orchestrates the analysis workflow by taking project source code as input and running rule-based detectors. These detectors, which check for best practices and issues in RL code, are split into two types: built-in rules for common patterns and custom rules tailored to specific project needs. Rule-based detectors are integrated in a *Code Analyzer*, which parses RL source code and checks for rule compliance. It identifies violations and passes the findings to the *Result Generation* component, which outputs detailed JSON reports.

Figure 1 provides an overview of the architecture of the ML pipeline insights service. The service runs within the *MLOps pipeline* of the ML project or as a standalone service. It provides its reports to the *Web-based Insights UI*, enabling a continuous feedback loop with the ML specialists and software architects working on the project.

3.3 Architecture of LLM-Based Detection Approach

The *LLM-based detection framework* is crafted as a modular and intelligent system for automating the discovery of best practices within RL-based software architectures. Figure 1 shows the architecture of the framework.

The main component of the system is the *LLM Code Flow Executor*, which runs full validation workflows or targeted checks. These workflows, defined in the *LLM Code Flows* module, specify rule-aligned detection steps. The *LLM* module abstracts different language models, while the *Conversation* class handles structured interactions, allowing detectors to query and interpret LLM responses within context. The interactions between the detection logic and the LLM are managed by the *Conversation* class within the *LLM* module. It maintains conversational state, enabling detectors to pose structured queries and interpret the LLM's contextual responses as part of the detection process. The *LLM-based Detectors* module implements the detection routines responsible for verifying the architectural rules. Each detector receives a shared Data Transfer Object, the *LLM Code Flow DTO*, which encapsulates critical context such as project metadata, source file paths, and accumulated validation results. As the object moves through the detector pipeline, it gradually builds a compliance report.

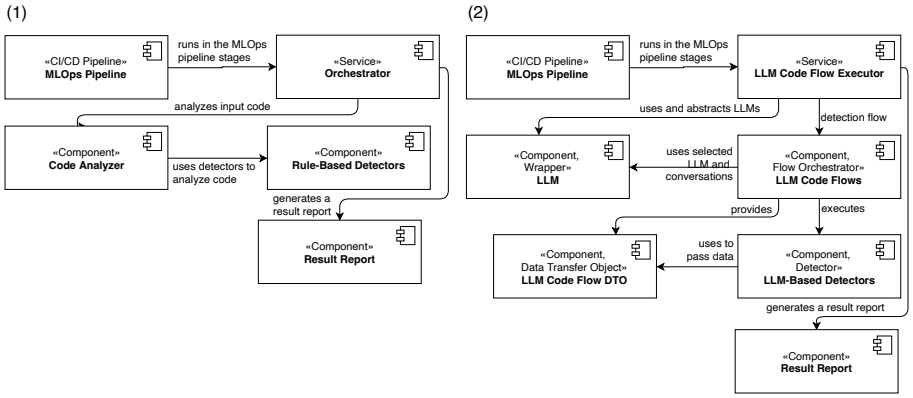

Fig. 1. (1): Architecture of the ML Pipeline Insights Service, (2): Architecture of the LLM-Based Detectors

3.4 Rules and Detectors

Rules Tables 1 2 3 4 present a binary classification of architectural rules used in this study. A rule is considered true when the associated architectural principle is evidently implemented in the source code. Conversely, a rule is false if the implementation either contradicts or lacks the required elements.

For example, in the evaluation of CS11, the rule *R11: Maintain a centralized model registry* holds true because all trained RL models are saved in uniquely named directories, with comprehensive metadata including training level, mode, and performance logs, as implemented across files. Similarly, the rule *R13: Implement versioning semantics* is also supported: while semantic versioning is not explicitly used, the code applies a structured naming convention with incremental epoch tracking and level-based model restoration, ensuring backward compatibility and traceability.

Rule-Based Detectors The rule-based detectors consist of specialized functions that analyze source code to determine rule compliance. These detectors use regular expressions, pattern matching, and parsing techniques to identify relevant implementation patterns.

Custom rule-based detectors directly analyze functional roles. For example, in CS12, detectors identify structured logging via `logging.debug()` and track evaluation callbacks like `EvalLogCallback`, while verifying modularity through the separation of classes and function responsibilities across independent files.

LLM-based detectors leverage LLMs to examine source code, identify patterns, and assess the adherence of these patterns to architectural rules. Unlike rule-based systems, these detectors are capable of flexible reasoning, allowing them to recognize relevant structures in the code even when they do not precisely conform to predefined syntactic patterns.

Our design emphasizes a high degree of modularity: Each LLM-based detector focuses on identifying a specific fragment of code that contributes to realizing a particular

Table 1. RL Pipeline Implementation Rules

Rule	Description
R01	Maintain a dedicated configuration file specifying all RL pipeline parameters, including environment settings, agent hyperparameters, and training policies.
R02	Ensure modular design by separating environment creation, agent design, training logic, and evaluation into distinct components.
R03	Incorporate structured logging mechanisms to record events and metrics at each stage of the RL pipeline, including environment setup, training progress, reward trends, and evaluation outcomes.
R04	Define consistent data formats (e.g., CSV, JSON) for storing training metrics, policy weights, and performance records to facilitate analysis and reproducibility.
R05	Implement checks to ensure that trained agents are compatible with various versions of the environment before deployment.
R06	Integrate hyperparameter optimization frameworks (e.g., Grid Search, Bayesian Optimization) to automate the tuning process.
R07	Incorporate checkpointing strategies that save the model's state at regular intervals.
R08	Regularly test policy generalization by evaluating the trained agent on variations of the environment.
R09	Set fixed random seeds and track software/library versions to ensure that training experiments are reproducible.
R10	Establish evaluation protocols that compare agent performance against baseline policies and predefined benchmarks.

rule. Rather than attempting to assess an entire rule, each detector contributes a single piece to the overall validation. This granularity has two distinct advantages: it enhances the precision and reliability of the LLM's responses, and it allows for transparent traceability between the rule and its concrete realizations in the source code.

The detection workflow follows structured steps: the LLM scans code for relevant elements, checks them against architectural rules, and links any violations to specific code lines. It works within a larger framework that manages data, settings, and reporting, ensuring reliable validation.

4 Case Studies

In this section, we describe the case studies used to evaluate our approach and test the performance of the rule detection. Table 5 summarizes the case studies.

4.1 Industrial Case Study

To test our rule-based assessment approach, we applied our *ML pipeline insights service* to a real industrial system. This system is a flexible framework for training both single-agent and multi-agent reinforcement learning (RL) models across different environments. It includes modules for managing agents, customizing training environments, and running both centralized and decentralized learning. The framework also supports advanced hyperparameter tuning and optimization to improve AI performance.

Table 2. RL Model Versioning and Lifecycle Management Rules

Rule	Description
R11	Maintain a centralized model registry to store all trained RL models along with comprehensive metadata (version ID, training environment conditions, hyperparameters, performance metrics).
R12	Interactions with the model registry or other system components (training pipelines, deployment orchestrators, inference services) should only occur via defined HTTP APIs.
R13	Implement versioning semantics (e.g., semantic versioning or incremental numeric identifiers) consistently across all models stored in the registry.
R14	Deployments of RL models should use Kubernetes manifests or Helm charts for scalability and controlled rollouts.
R15	Implement automated rollback mechanisms triggered by threshold breaches or deployment errors.
R16	Integrate a standardized model evaluation step within the pipeline, systematically comparing candidate RL models using predefined metrics.
R17	Automate the selection and promotion of the optimal RL model candidate based on evaluation results.
R18	Define and monitor key indicators (e.g., environment state changes, performance degradation) explicitly as automated triggers to initiate RL model retraining pipelines without manual intervention.
R19	Use configurable threshold parameters for retraining triggers via YAML, JSON, or similar configuration files.
R20	Implement structured logging of retraining events, capturing trigger reasons, timestamps, relevant metrics, and outcomes.
R21	Store metadata such as training parameters, environment configurations, simulation settings, and performance metrics.
R22	Upon successful training in simulation, automatically register the new model version in the model registry along with its metadata.
R23	Ensure that deployment stages reference specific model versions from the registry.
R24	Maintain separate stages (e.g., development, staging, production) with distinct model versions to facilitate safe testing and deployment. e.g. canary releases and A/B testing.
R25	Incorporate small real-world testing of newly deployed models.

Table 3. Multi-Agent System Deployment Rules

Rule	Description
R26	Utilize established messaging libraries or frameworks to facilitate inter-agent communication.
R27	Employ frameworks that support dynamic agent management and scalability.
R28	Implement logging mechanisms to monitor inter-agent interactions.
R29	Implement conflict detection and resolution mechanisms within the agent framework.

The platform is used to automate production tasks with AI and is designed to work smoothly in real factory settings. Our insights service is integrated into its MLOps pipeline, offering real-time feedback on training and inference to support best practices. Developed by a large team, the framework is being rolled out across factories worldwide. It helps standardize production processes and gives ML engineers and soft-

Table 4. Rules for Managing Models in Inference

Rule	Description
R30	Utilize frameworks that optimize resource usage during model inference.
R31	Maintain a versioned repository of models to track changes.
R32	Implement monitoring tools to evaluate model performance metrics in real-time.

Table 5. Overview of Python-based systems and their functionalities

ID	Description
ICS	This system focuses on an advanced framework for single-agent and multi-agent RL. It enables thorough testing and optimization of AI policies in diverse environments (See Section 4.1 for details).
CS1	In this system, zombies originate at the top border of the screen and move downward along varying, unpredictable trajectories until they reach the bottom border. https://pettingzoo.farama.org/tutorials/sb3/kaz/
CS2	In this system, Waterworld, the simulation revolves around archaea organisms navigating their environment in a quest for survival. https://pettingzoo.farama.org/tutorials/sb3/waterworld/
CS3	This system is a Chess example. It uses the observation and action spaces similar to the AlphaZero method, with slight modifications. https://pettingzoo.farama.org/tutorials/sb3/connect_four/
CS4	This system is a game that uses physics-based challenge where the objective is to guide a ball to the left wall of the game's boundary. https://pettingzoo.farama.org/tutorials/rllib/pistonball/
CS5	This system is a game where 2 players must connect four of their tokens vertically, horizontally, or diagonally. https://docs.agilerl.com/en/latest/tutorials/pettingzoo/dqn.html
CS6	This system is a classic Atari game, where there are two ships controlled by two players who are each trying to maximize their score. https://pettingzoo.farama.org/tutorials/agilerl/MADDPG/
CS7	This system trains AI agents to play Tic-Tac-Toe using the PettingZoo environment. https://github.com/Farama-Foundation/PettingZoo/blob/master/tutorials/Tianshou/3_cli_and_logging.py
CS8	In this system, there are two agents: the 'speaker' and the 'listener'. The 'speaker' agent possesses the ability to communicate verbally but lacks the capability to move autonomously.https://docs.agilerl.com/en/latest/tutorials/pettingzoo/matd3.html#matd3-tutorial
CS9	This system is also a classic Atari game similar to CS6. https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/ppo_pettingzoo_ma_atari.py
CS10	This system is similar to CS1. It trains agents in the "Knights-Archers-Zombies" environment using Black Death wrapper to handle agent deaths effectively. https://pettingzoo.farama.org/tutorials/sb3/kaz/
CS11	This system, HHMARL 2D, simulates hierarchical multi-agent air combat scenarios, where heterogeneous aircraft agents perform fight or escape maneuvers coordinated by a high-level commander policy. https://github.com/IDSIA/hhmarl_2D
CS12	This system, NFVdeep, applies deep reinforcement learning to dynamically orchestrate service function chains in network function virtualization environments. https://github.com/CN-UPB/NFVdeep
CS13	This system, IMIL (Infosys Model Inference Library), offers a unified, high-performance framework for loading and deploying machine learning models across diverse platforms and formats. https://github.com/Infosys/Infosys-Model-Inference-Library
CS14	This system implements a multi-agent concierge using LlamaIndex Workflows, where customizable agents equipped with tools collaboratively manage tasks and interact with users in a coordinated environment. https://github.com/run-Llama/multi-agent-concierge
CS15	This system applies distributed Proximal Policy Optimization (PPO) to manage multi-agent traffic light control in a SUMO-based urban grid, where each intersection acts as an independent reinforcement learning agent. https://github.com/maxbrenner-ai/Multi-Agent-Distributed-PPO-Traffc-light-control
CS16	This system, JAT (Jack of All Trades), is a multi-purpose transformer agent capable of handling diverse reinforcement learning and vision-language tasks using a unified architecture. https://github.com/huggingface/jat

ware architects clear insights into how training pipelines are built and maintained across projects. This case study shows our tool works well in a complex, real-world environment.

5 Validation

In this section, we validate our approach by demonstrating how the defined rules support best practices in RL systems. Our evaluation uses three popular LLMs, OpenAI GPT-4o, Qwen2.5 72B, and Llama 3.3 70B. The OpenAI models were run on Microsoft Azure[2], while the five freely available models were deployed via OLlama[3] on a Cisco UCS C245 M6 server. We evaluated their effectiveness across diverse RL settings and addressed the following research questions.

This paper aims to answer the following research questions:

- **RQ1 Accuracy:** How accurately does the LLM-based rule checking mechanism ensure compliance with best practices for RL training compared to the custom-based rule checking mechanism?
- **RQ2 Automation:** How to automatically validate these architectural rules?

5.1 Validation Setup

The validation involved three main steps: defining architecture rules covering RL pipelines, model versioning, multi-agent setups, and inference; implementing a tool to automatically check code against these rules; and applying this tool in a case study to assess rule compliance in real RL systems.

To evaluate our rule-based detectors, we use three metrics: precision, recall, and F1-score. *True Positives (TP)* are correctly identified best practice cases, *False Positives (FP)* are incorrect detections, and *False Negatives (FN)* are missed valid cases. We exclude *True Negatives (TN)* since our ground truth only includes compliant code.

$$P = \frac{TP}{TP + FP} \quad R = \frac{TP}{TP + FN} \quad F_1 = \frac{2PR}{P + R}$$

Precision (P) shows how accurate the detections are, recall (R) shows how complete they are, and the F1-score balances both to give an overall performance measure. This focus helps us assess how well the method detects compliant code.

5.2 Results

This section presents the validation results, summarized in Tables 6 7 8 9, assessing the accuracy of the rule-based detectors.

5.3 Results on the Industrial Case Study

The *custom-based detectors* achieved consistently high performance across all categories, with perfect precision, recall, and F1 scores in most cases. These detectors were particularly effective at capturing the structure and logic of RL-specific patterns such as checkpointing, modular training loops, version control mechanisms, and agent-environment configuration.

[2] `https://azure.com`
[3] `https://oLlama.com`

All LLMs also performed strongly, especially for general RL pipeline rules and multi-agent setup, where they achieved perfect scores. These models reliably identified practices such as agent orchestration, modular environment definitions, and deployment-ready configurations. However, their recall dropped significantly in rules related to *model versioning and inference management*. While their precision remained high (1.0), indicating correct predictions when rules were found, the low recall (0.5) shows that they missed many relevant code instances. These suggest that while LLMs can identify well-known or syntactically explicit practices, they are less sensitive to subtler implementation patterns or project-specific naming conventions, especially in version control and inference flows.

The *N/A* in the tables indicate cases where no relevant rules were supported or detected within the system.

5.4 Results on the Open-Source Case Studies

RL Pipeline Implementation. The custom-based detectors demonstrated strong and consistent performance across all case studies, with F1 scores generally exceeding 0.90 and reaching 1.0 in systems such as CS6–CS9 and CS12–CS15. These detectors successfully identified best practices such as modular environment and agent creation, structured logging, and separation of training logic. In comparison, Qwen2.5 and Llama 3.3 also showed high precision but varied in recall. Qwen2.5, for instance, scored well in CS3 and CS7 (F1 = 0.91 and 0.89), but underperformed in CS4 and CS14 (F1 = 0.50 and 0.46). GPT-4o generally had lower recall, such as in CS6 and CS7, indicating it often failed to identify some rule occurrences even when predictions were accurate.

Model Versioning and Lifecycle Management. Custom detectors consistently achieved perfect scores across all systems (F1 = 1.0), showing strong capability in detecting practices such as checkpointing with metadata, rollback mechanisms, and version tracking. In contrast, all LLM-based detectors exhibited performance degradation in this category. Recall values were particularly low, resulting in low to moderate F1 scores. For example, in CS4 and CS9, Qwen2.5 and Llama had F1 scores as low as 0.12–0.13. GPT-4o showed relatively better recall in CS2 and CS5 (F1 = 0.75 and 0.60), but still fell short of the custom-based detector performance.

Multi-Agent Aspects and Deployment. For this practice, custom-based detectors again performed optimally in all relevant systems, achieving perfect precision and recall. These detectors effectively recognized common multi-agent configurations, communication mechanisms, and environment setups. The LLM-based detectors showed variable performance. While Qwen2.5 had strong results in CS14 (F1 = 0.86), it performed poorly or failed entirely in systems like CS3, CS12–CS16. GPT-4o and Llama 3.3 achieved strong results in CS9 and CS14 (F1 = 0.80–0.86) but showed limited recall elsewhere. The inconsistencies suggest that multi-agent patterns that are not explicit or follow custom abstractions challenge the LLMs.

Managing Models in Inference. Inference rules performing models and their integration into evaluation scripts—were reliably captured by custom detectors in all applicable

Table 6. Results on Rules for RL Pipeline Implementation

Case Study	Custom-Based			Qwen 2.5			GPT-4o			LLaMA 3.3		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
ICS	1.00	0.89	0.94	0.90	1.00	0.95	0.89	0.89	0.89	0.90	1.00	0.95
CS1	1.00	0.80	0.89	1.00	0.60	0.75	0.56	1.00	0.71	0.83	1.00	0.91
CS2	1.00	0.80	0.89	1.00	0.60	0.75	0.50	0.60	0.55	0.67	0.80	0.73
CS3	1.00	0.80	0.89	0.83	1.00	0.91	0.50	0.80	0.62	0.50	0.80	0.62
CS4	1.00	0.75	0.86	0.38	0.75	0.50	0.50	0.50	0.50	0.33	0.75	0.46
CS5	1.00	0.89	0.94	0.86	0.67	0.75	0.88	0.78	0.82	0.88	0.78	0.82
CS6	1.00	1.00	1.00	0.56	0.83	0.67	1.00	0.17	0.29	0.60	1.00	0.75
CS7	1.00	1.00	1.00	0.80	1.00	0.89	0.50	0.12	0.20	0.80	1.00	0.89
CS8	1.00	1.00	1.00	0.60	1.00	0.75	0.60	0.50	0.55	0.56	0.83	0.67
CS9	1.00	1.00	1.00	0.44	1.00	0.62	0.40	0.50	0.44	0.44	1.00	0.62
CS10	1.00	0.80	0.89	0.67	0.40	0.50	0.50	0.80	0.62	0.67	0.80	0.73
CS11	1.00	0.88	0.93	0.80	1.00	0.89	0.78	0.88	0.82	0.80	1.00	0.89
CS12	1.00	1.00	1.00	0.70	1.00	0.82	0.62	0.71	0.67	0.70	1.00	0.82
CS13	1.00	1.00	1.00	0.50	1.00	0.67	0.44	0.80	0.57	0.50	1.00	0.67
CS14	1.00	1.00	1.00	0.30	1.00	0.46	0.12	0.33	0.18	0.30	1.00	0.46
CS15	1.00	1.00	1.00	0.60	1.00	0.75	0.62	0.83	0.71	0.60	1.00	0.75
CS16	1.00	0.83	0.91	0.60	1.00	0.75	0.60	1.00	0.75	0.60	1.00	0.75

systems. In contrast, LLMs struggled across most systems, with frequent N/A entries indicating no rule detections. When applicable, Qwen2.5 and Llama achieved moderate results (e.g., CS5–CS9 with F1 between 0.50–0.80). GPT-4o performed slightly better in CS9, CS11–CS13, but generally failed to detect rules in systems where inference handling was not explicitly coded or used unconventional naming.

6 Discussion

This section examines the research questions and discusses other lessons learned.

RQ1. Accuracy The evaluation across the industrial case study and the sixteen open-source case studies confirms that the overall detection accuracy of the approach is high when using custom rule-based detectors. These detectors consistently achieved near-perfect scores across all rule categories (Precision: 1.0, Recall: 0.96, and F1-score: 0.98 on average), demonstrating their effectiveness in capturing full rule implementations, including practices like metadata tracking, inference-ready model handling, or agent orchestration.

LLM-based detectors showed promising results but exhibited variance across rule types and systems. Qwen2.5 achieved an average Precision of 0.71, Recall of 0.74, and F1-score of 0.69, with relatively strong performance in common RL pipeline configurations and multi-agent aspects but lower sensitivity for versioning and inference patterns. GPT-4o and Llama 3.3 performed similarly, with GPT-4o reaching an average F1 of 0.67 and Llama 3.3 achieving 0.68, both affected by recall drops in complex or less explicit code structures. These results indicate that LLMs can detect many true positives but still miss implicit or tightly coupled implementations.

RQ2. Automation Both evaluated detection approaches support fully automated execution once configured. However, the degree of required *manual setup and adaptation*

Table 7. Results on RL Model Versioning and Lifecycle Management Rules

Case Study	Custom-Based			Qwen 2.5			GPT-4o			LLaMA 3.3		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
ICS	1.00	1.00	1.00	0.33	1.00	0.50	0.33	1.00	0.50	0.33	1.00	0.50
CS1	1.00	1.00	1.00	0.25	0.33	0.29	0.33	1.00	0.50	0.22	0.67	0.33
CS2	1.00	1.00	1.00	0.50	0.33	0.40	0.60	1.00	0.75	0.20	0.33	0.25
CS3	1.00	1.00	1.00	0.33	0.67	0.44	0.27	1.00	0.43	0.25	1.00	0.40
CS4	1.00	1.00	1.00	0.07	1.00	0.13	0.33	1.00	0.50	0.07	1.00	0.13
CS5	1.00	1.00	1.00	0.36	0.83	0.50	0.43	1.00	0.60	0.38	0.83	0.53
CS6	1.00	1.00	1.00	0.23	1.00	0.38	1.00	0.33	0.50	0.20	1.00	0.33
CS7	1.00	1.00	1.00	0.14	0.67	0.24	0.67	0.67	0.67	0.20	1.00	0.33
CS8	1.00	1.00	1.00	0.20	1.00	0.33	0.50	0.67	0.57	0.23	1.00	0.38
CS9	1.00	1.00	1.00	0.07	1.00	0.12	0.08	1.00	0.15	0.07	1.00	0.13
CS10	1.00	1.00	1.00	0.50	0.33	0.40	0.60	1.00	0.75	0.25	0.67	0.36
CS11	1.00	1.00	1.00	0.33	1.00	0.50	0.36	1.00	0.53	0.33	1.00	0.50
CS12	1.00	1.00	1.00	0.20	1.00	0.33	0.38	1.00	0.55	0.20	1.00	0.33
CS13	1.00	1.00	1.00	0.07	1.00	0.12	0.07	1.00	0.12	0.07	1.00	0.12
CS14	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS15	1.00	1.00	1.00	0.27	1.00	0.42	0.17	0.50	0.25	0.27	1.00	0.42
CS16	1.00	1.00	1.00	0.13	1.00	0.24	0.13	1.00	0.24	0.13	1.00	0.24

effort varies significantly. The custom detectors provide the highest accuracy across all rule categories but require rule-specific implementation tailored to the project context. This includes coding logic to trace rule-specific workflows, such as verifying inference-time model loading sequences.

In contrast, LLM-based detectors require no manual coding per rule but show varying performance across environments and rule types. They generalize well for common RL pipeline patterns, especially for modular design or agent orchestration, but struggle in accurately detecting rules that involve implicit logic or custom abstractions, such as lifecycle hooks or inference flow logic. These models offer a low-effort automation entry point but require post-hoc validation or fallback detection when full rule coverage is critical.

Further Lessons Learned While LLMs provide flexibility and adaptability across projects and libraries without explicit reimplementation, their limitations in rule completeness and traceability remain evident. For example, in systems like CS4, CS13, and CS16, LLMs failed to detect rules that custom detectors handled precisely. This highlights the benefit of hybrid use: LLMs can quickly scan unfamiliar systems, whereas custom detectors should be employed when consistent detection or complete traceability is required. We recommend applying LLM-based detectors in early review phases for a broad but coarse overview, then refining findings with custom detectors where critical rules or complex code structures are involved. This staged use ensures scalability for ML observability pipelines, such as in CI/CD scenarios, while maintaining trust in detection accuracy.

7 Threats to Validity

In this section, we discuss the potential threats to the validity and the steps taken to mitigate these threats.

Table 8. Results on Multi-Agent Aspects and Deployment Rules

Case Study	Custom-Based			Qwen 2.5			GPT-4o			LLaMA 3.3		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
ICS	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
CS1	1.00	1.00	1.00	1.00	0.33	0.50	1.00	1.00	1.00	1.00	0.33	0.50
CS2	1.00	1.00	1.00	1.00	0.50	0.67	0.67	1.00	0.80	1.00	0.50	0.67
CS3	1.00	1.00	1.00	0.00	0.00	0.00	0.50	1.00	0.67	0.67	1.00	0.80
CS4	1.00	1.00	1.00	0.50	1.00	0.67	1.00	0.50	0.67	0.33	0.50	0.40
CS5	1.00	1.00	1.00	0.50	0.50	0.50	0.50	0.50	0.50	0.50	1.00	0.67
CS6	1.00	1.00	1.00	0.50	1.00	0.67	1.00	0.50	0.67	0.50	1.00	0.67
CS7	1.00	1.00	1.00	0.33	0.50	0.40	1.00	1.00	1.00	0.50	1.00	0.67
CS8	1.00	1.00	1.00	0.50	1.00	0.67	0.67	1.00	0.80	0.50	1.00	0.67
CS9	1.00	1.00	1.00	0.50	1.00	0.67	0.67	1.00	0.80	0.67	1.00	0.80
CS10	1.00	1.00	1.00	N/A	0.00	N/A	0.75	1.00	0.86	1.00	0.33	0.50
CS11	1.00	1.00	1.00	0.50	1.00	0.67	0.50	1.00	0.67	0.50	1.00	0.67
CS12	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS13	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS14	1.00	1.00	1.00	0.75	1.00	0.86	0.75	1.00	0.86	0.75	1.00	0.86
CS15	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS16	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A

External validity refers to the generalizability of our findings. While our study includes one industrial and sixteen open-source case studies, broader testing across more domains is needed to strengthen generalizability. The current diversity helps mitigate threats, but future work will expand the dataset.

Internal validity ensures the accuracy of results and their attribution to the interventions. We maintained internal validity by rigorously defining and applying rules with our assessment framework. Detection algorithms were carefully implemented and manually verified to identify true positives and false negatives. Multiple researchers cross-checked the results to minimize biases, ensuring that did not introduce issues.

Construct validity checks if the study measures what it intends to. Our construct validity relies on accurately specifying and detecting RL training best practices. We mitigated threats by basing rule definitions on a thorough literature review and established practices. Acknowledging interpretations and expertise, we incorporated feedback from experts to refine the rules and ensure they reflect best practices accurately.

8 Related Work

In this section, we discuss related studies and compare them to our study.

Several studies explore various aspects of RL methodologies and applications. Lee et al.[9] analyze the evolution of RL algorithms, emphasizing the transition from single-agent to multi-agent systems, focusing on distributed optimization. Canese et al.[1] outline multi-agent algorithms, comparing them across key attributes essential for multi-agent RL applications such as non-stationarity, scalability, and observability.

A survey by Samsami and Alimadad [15] provides an overview of distributed RL techniques, discussing key challenges such as efficient data usage and the balance between exploration and exploitation. Similarly, Hoffman et al. [6] introduced Acme, a modular framework designed to simplify the implementation of scalable RL algorithms.

Table 9. Results on Managing Models in Inference Rules

Case Study	Custom-Based			Qwen 2.5			GPT-4o			LLaMA 3.3		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
ICS	1.00	1.00	1.00	0.33	1.00	0.50	0.33	1.00	0.50	0.33	1.00	0.50
CS1	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS2	N/A	N/A	N/A	0.00	N/A	N/A	N/A	N/A	N/A	0.00	N/A	N/A
CS3	N/A	N/A	N/A	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS4	1.00	1.00	1.00	0.33	1.00	0.50	N/A	0.00	N/A	0.33	1.00	0.50
CS5	1.00	1.00	1.00	1.00	0.50	0.67	1.00	0.50	0.67	0.67	1.00	0.80
CS6	1.00	1.00	1.00	0.33	1.00	0.50	N/A	0.00	N/A	0.33	1.00	0.50
CS7	1.00	1.00	1.00	0.67	1.00	0.80	N/A	0.00	N/A	0.67	1.00	0.80
CS8	1.00	1.00	1.00	0.33	1.00	0.50	N/A	0.00	N/A	0.33	1.00	0.50
CS9	1.00	1.00	1.00	0.50	0.50	0.50	0.67	1.00	0.80	1.00	1.00	1.00
CS10	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS11	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
CS12	1.00	1.00	1.00	0.67	1.00	0.80	1.00	0.50	0.67	0.67	1.00	0.80
CS13	1.00	1.00	1.00	0.33	1.00	0.50	0.33	1.00	0.50	0.33	1.00	0.50
CS14	N/A	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A
CS15	1.00	1.00	1.00	0.67	1.00	0.80	1.00	0.50	0.67	0.67	1.00	0.80
CS16	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

In multi-agent RL settings, Zhu et al. [22] proposed MSRL, a training system that employs fragmented dataflow graphs to execute RL algorithms in a flexible and scalable manner. Liang et al. [11] developed RLlib, a library focused on making RL training more scalable by distributing computational tasks efficiently.

The complexities of MARL are addressed by Hernandez-Leal et al. [5] in their comprehensive survey, which tackles challenges such as non-stationarity and scalability in training multiple agents in dynamic environments. Zhang, Yang, and Basar [20] provide an overview of theories and algorithms in MARL, focusing on cooperative and competitive settings and emphasizing the need for robust training practices. While these studies provide theoretical and algorithmic insights into MARL, our work applies these concepts by defining specific rules for multi-agent training and validating their implementation through automated tools.

In automated code analysis and rule-based systems, Schneider et al. [17] discuss rule-based security analysis for microservices. Their work underscores the importance of automated rule checking to maintain best practices in software systems. Our work is similar in its goal of automating the enforcement of best practices, but it focuses specifically on RL training practices rather than security analysis.

9 Conclusions and Future Work

This paper introduced a rule-based assessment approach to automatically validate best practices in RL training pipelines. The approach was implemented as a modular pipeline insights service and evaluated using an industrial system and sixteen diverse open-source case studies. The key RL aspects assessed include RL pipeline practices, model versioning, multi-agent deployment and managing models in inference. The evaluation demonstrated that the *custom rule-based detectors* consistently achieved high accuracy across all systems and rule categories, with near-perfect precision and recall. In contrast, LLM-based detectors enabled fully automated, low-effort validation but showed

variable recall. These findings show that while LLMs provide a scalable and adaptable first level of analysis, custom detectors are indispensable to achieve complete and reliable coverage of rule implementations. This hybrid detection setup allows fast insights in early development stages and deep validation when rules must be strictly enforced.

Future work will expand the rules to include distributed training and environment-specific patterns. We also plan to integrate rule checks into CI/CD pipelines for real-time validation, enhancing transparency of RL systems.

10 Acknowledgments

This work was supported by: FFG (Austrian Research Promotion Agency) project MODIS (no. FO999895431); Austrian Science Fund (FWF) project CQ4CD, Grant-DOI: 10.55776/I6510.

References

1. Canese, L., Cardarilli, G.C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., Spanò, S.: Multi-agent reinforcement learning: A review of challenges and applications. Applied Sciences **11**(11), 4948 (2021)
2. Dagster: Ml pipelines: 5 components and 5 critical best practices. Dagster Blog (2023)
3. Deng, Y., Dai, Q., Zhang, Z.: An Overview of Computational Sparse Models and Their Applications in Artificial Intelligence, pp. 345–369. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
4. Eimer, T., Lindauer, M., Raileanu, R.: Hyperparameters in reinforcement learning and how to tune them. International Conference on Machine Learning pp. 9104–9149 (2023)
5. Hernandez-Leal, P., Kartal, B., Taylor, M.E.: A survey and critique of multiagent deep reinforcement learning. Autonomous Agents and Multi-Agent Systems **33**(6), 750–797 (2019)
6. Hoffman, M.W., Oh, J., Andrychowicz, M., et al.: Acme: A research framework for distributed reinforcement learning. arXiv preprint arXiv:2006.00979 (2020)
7. Lakshmanan, V., Robinson, S., Munn, M.: Machine Learning Design Patterns. O'Reilly Media, Inc. (October 2020)
8. Latendresse, J., Abedu, S., Abdellatif, A., Shihab, E.: An exploratory study on machine learning model management. ACM Transactions on Software Engineering and Methodology **34**(1), 1–31 (2024)
9. Lee, D., He, N., Kamalaruban, P., Cevher, V.: Optimization for reinforcement learning: From a single agent to cooperative agents. IEEE Signal Processing Magazine **37**(3), 123–135 (2020)
10. Levine, S.: Reinforcement learning and control as probabilistic inference: Tutorial and review (2018), https://arxiv.org/abs/1805.00909
11. Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., Stoica, I.: Rllib: Abstractions for distributed reinforcement learning. In: International conference on machine learning. pp. 3053–3062. PMLR (2018)
12. Liu, Z., Zhang, J., Shi, E., Liu, Z., Niyato, D., Ai, B., Shen, X.S.: Graph neural network meets multi-agent reinforcement learning: Fundamentals, applications, and future directions. IEEE Wireless Communications (2024)

13. Ntentos, E., Warnett, S.J., Zdun, U.: Rule-based assessment of reinforcement learning practices using large language models. In: 4th International Conference on AI Engineering ? Software Engineering for AI (CAIN) (April 2025)
14. Pateria, S., Subagdja, B., Tan, A.H., Quek, C.Q.: Hierarchical reinforcement learning: A comprehensive survey. Artificial Intelligence Review **55**, 3577–3621 (2022)
15. Samsami, M., Alimadad, A.: A survey of distributed reinforcement learning: Techniques and applications. arXiv preprint arXiv:2011.11012 (2020)
16. Samsami, M.R., Alimadad, H.: Distributed deep reinforcement learning: An overview. arXiv preprint arXiv:2011.11012 (2020)
17. Schneider, S., Quéval, P.J., Milánkovich, Á., Díaz Ferreyra, N.E., Zdun, U., Scandariato, R.: Automatic rule checking for microservices: Supporting security analysis with explainability. Available at SSRN 4658575 (2023)
18. Sharma, R., Davuluri, K.: Design patterns for machine learning applications. In: 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC). pp. 818–821 (2019)
19. Washizaki, H., Khomh, F., Guéhéneuc, Y.G., Takeuchi, H., Natori, N., Doi, T., Okuda, S.: Software-engineering design patterns for machine learning applications. Computer **55**(3), 30–39 (2022)
20. Zhang, K., Yang, Z., Başar, T.: Multi-agent reinforcement learning: A selective overview of theories and algorithms. Handbook of reinforcement learning and control pp. 321–384 (2021)
21. Zhang, K., Yang, Z., Başar, T.: Multi-agent reinforcement learning: A selective overview of theories and algorithms. Applied Sciences **11**(11), 4948 (2021), https://www.mdpi.com/2076-3417/11/11/4948
22. Zhu, J., Lu, Y., et al.: Msrl: A scalable and modularized multi-agent rl training system. arXiv preprint arXiv:2210.00882 (2022)