RLOps Pipeline Development with Low-Code and Large Language Models for Industry 4.0

Stephen John Warnett*
Research Group Software Architecture, Faculty of Computer Science, University of Vienna
Vienna, Austria
UniVie Doctoral School Computer Science DoCS, Faculty of Computer Science, University of Vienna
Vienna, Austria
stephen.warnett@univie.ac.at

Uwe Zdun
Research Group Software Architecture, Faculty of Computer Science, University of Vienna
Vienna, Austria
uwe.zdun@univie.ac.at

Sebastian Geiger
Siemens Aktiengesellschaft Österreich
Vienna, Austria
sebastian.a.geiger@siemens.com

Abstract

Machine learning operations (MLOps) automate common tasks throughout the entire life cycle of a machine learning model. In Industry 4.0 environments, cyber-physical production systems increasingly utilise reinforcement learning (RL) models to optimise and automate production processes, giving rise to reinforcement learning operations (RLOps). However, manually configuring RLOps pipelines requires specialised development operations (DevOps) knowledge, and this can limit the potential for automation. We propose a template-based approach for rapidly creating and deploying RLOps pipelines in Industry 4.0 environments that minimises the required programming effort and expertise. Based on the Pipes and Filters pattern, our modular solution leverages large language models (LLMs) for automated pipeline creation. It enables fully automated execution, including model training, testing and deployment, with built-in quality control to ensure correct configurations. We validate our approach through evaluation with multiple LLMs in an Industry 4.0 context. Our results demonstrate that our solution, used with a suitable LLM, can reliably generate and execute RLOps pipelines with low error rates, thereby reducing development time and the need for specialised DevOps knowledge.

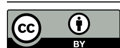
CCS Concepts

• **Computing methodologies** → **Machine learning**; **Reinforcement learning**; **Artificial intelligence**; • **Software and its engineering** → **Software organization**; *Continuous integration*; *Continuous deployment*; • **Applied computing** → *Industrial automation*; *Cyber-physical systems*; *Manufacturing systems.*

Keywords

MLOps, RLOps, Machine Learning, Reinforcement Learning, Industry 4.0, Cyber-Physical Production Systems

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
CAIN '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2475-6/2026/04
https://doi.org/10.1145/3793653.3793779

ACM Reference Format:
Stephen John Warnett, Uwe Zdun, and Sebastian Geiger. 2026. RLOps Pipeline Development with Low-Code and Large Language Models for Industry 4.0. In *2026 IEEE/ACM 5th International Conference on AI Engineering - Software Engineering for AI (CAIN '26), April 12–13, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3793653.3793779

1 Introduction

Machine learning operations (MLOps) [1–3] are practices for the automated execution of tasks related to the machine learning (ML) [4, 5] workflow for supervised and unsupervised learning. Applied to reinforcement learning (RL) [6], MLOps is termed RLOps (Reinforcement Learning Operations) [7]. Industry 4.0 [8, 9] integrates digital and physical technologies, and this has led to the development of cyber-physical production systems (CPPSs) [10] which improve efficiency, flexibility and customisation [11].

The differences between RLOps in Industry 4.0 and the operations of cloud-based ML systems are significant. In contrast to centrally controlled cloud environments, an Industry 4.0 CPPS imposes distinct constraints that affect pipeline design and deployment. RLOps for Industry 4.0 must interact directly with physical production systems, adhere to industrial safety and communication protocols, and integrate with existing automation infrastructure.

MLOps, RLOps and Industry 4.0 CPPSs place great value on automation. Yet one of the activities central to automation – continuous integration/continuous delivery (CI/CD) during pipeline development – remains largely manual. This activity requires specialised development operations (DevOps) [12, 13] knowledge, as well as software engineering and ML/RL know-how [2, 14].

To address these challenges, we propose an approach to automating the generation of GitLab RLOps CI/CD pipelines using LLMs, based on low-code [15, 16] and template-based user-written specifications. We consider two research questions:

RQ1 *How accurately can a low-code, template-based, modular flow approach using LLMs generate Industry 4.0-specific RLOps pipeline configurations?*

RQ2 *How do popular LLMs perform in generating Industry 4.0-specific RLOps pipeline configurations in real industrial contexts?*

This study is part of an ongoing collaboration between the University of Vienna and Siemens Aktiengesellschaft Österreich. We validate our approach against a project derived from Siemens' real Industry 4.0 pipelines and typical use cases for RLOps CI/CD pipeline development, and empirically validate and compare the pipelines generated by seven LLMs against reference configurations and expected execution results.

The rest of the paper is organised as follows: Section 2 highlights related work, and Section 3 discusses our approach. Our validation is detailed in Section 4, and Section 5 discusses the research questions. Threats to validity are considered in Section 6, and Section 7 concludes and discusses future work.

2 Related Work

MLOps has been widely adopted because it automates ML model deployment; however, it is still fraught with challenges. Zhou et al. [2] recognise the need for MLOps to unify ML system development and address ML model deployment challenges. Sharma et al. [17] study options for implementing ML techniques within CI/CD pipelines, but do not provide a complete solution. Kumara et al. [18] review the industry literature and present a reference architecture for MLOps, highlighting the challenges of selecting tools and designing strategies for MLOps environments.

Li et al. [7] study RL applied to open radio access networks. They note the differences between ML and RL and categorise ML/RL model lifecycle challenges, whilst suggesting best practices for RLOps. Del Real Torres et al. [19] review deep RL approaches for smart manufacturing in Industry 4.0 and 5.0 [20], highlighting the importance of edge devices and CPPSs. Kegyes et al. [21] conduct a systematic literature review of RL applications and methods for Industry 4.0, providing a reference.

Low-code development methods have recently become more common [15, 16]. Hirzel [22] discusses how low-code programming reduces dependence on programming languages such as C, Java, and Python, instead relying on natural and visual languages. Bruhin et al. [23] consider the connection between low-code development platforms (LCDPs) and generative AI, and how generative AI can transform the potential of LCDPs.

LLMs are often used for code generation tasks. Jin et al. [24] systematically assess LLM code generation challenges and suggest guidelines to enhance LLM-based code generation system reliability, robustness and usability, whilst Li and Murr [25] study how GPT-4 models generate synthesised programs and state that GPT-4 model variants can effectively solve HumanEval [26] problems.

Our work complements these studies by unifying related topics, including MLOps, RLOps, Industry 4.0, CI/CD, and the utilisation of LLMs for code generation. To our knowledge, no prior work has addressed this topic for RLOps in Industry 4.0.

3 Approach

3.1 Research Process

Our approach is grounded in practical software engineering methodology. We describe and validate our proposed approach through evaluation in a real-world industrial context, demonstrating its practical utility and effectiveness in Industry 4.0 environments. By focusing on a project with an industry partner, we can identify the challenges and benefits of implementing our tool in a realistic RL pipeline and in industry applications.

Our solution is being developed in collaboration with Siemens Aktiengesellschaft AG as part of a research and development project for the next generation of intelligent CPPSs [11, 27], to integrate it into their workflow. It applies particularly to scenarios where pipeline configurations must be continuously created and edited at scale, i.e. for many factories. To highlight its use, the intended workflow, i.e. how Siemens should use our tool, is discussed in Section 3.5. We developed our approach through a systematic methodology and validated it through evaluation in an industrial context.

3.2 Template-Based Natural Language Low-Code Specifications

Our solution uses a structured, template-based low-code approach to specify RLOps pipelines, utilising LLMs. All pipeline details are specified in a natural-language-based template and serve as a guide to the generation process. A specification based on the template that we pass to the LLM as part of our prompting method (described in Section 3.3) follows:

```
Write a GitLab CI/CD yml file for Python reinforcement
     learning scripts.
Do not use Docker inside the GitLab CI/CD pipeline. Use
     the following context:

Usage: python train_rl.py

Required Files:
train_rl.py
eval_rl.py
test_rl.py

Requirements:
stable-baselines3>=2.1.0
supersuit>=3.9.0

GitLab CI/CD pipeline tasks:
train, eval, test

GitLab CI/CD pipeline generates artifacts:
Yes, it creates artifacts for the models in the train
     stage

GitLab CI/CD pipeline generates Docker images:
No

Tag to use in all stages (except for "deploy", if that
     stage is present):
gpu-runner
```

All study artefacts, including source code and full results, are available in our replication package [28]. Further examples of specification files that adhere to the template are in the `spec.txt` files in the subdirectories of `rl_dataset/evaluation_scenarios/`.

3.3 Prompting Method

We use few-shot prompting [29] for each pipeline configuration generation, error detection and validation step, including examples within the prompt to help the LLM produce better results. When retries are required, we include the conversation history and any detected issues to help the LLM improve its output. This technique (prompt chaining [29]) helps ensure that each step of the generation flow is correctly completed.

Figure 1 shows how our tool interacts with an LLM to generate a CI/CD pipeline configuration based on a low-code specification. Our tool first creates a system message (**System Message Setup** in Figure 1) instructing the LLM to create a GitLab CI/CD YAML file, as in the prompt below:

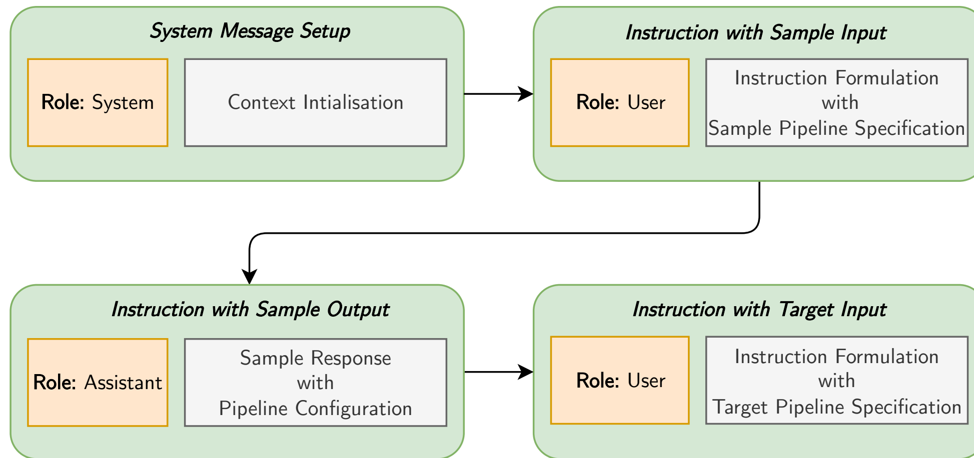


Figure 1: Our approach's conversation template.

```
You are an expert programmer. Your language of choice is
     Python, and you want to
create a GitLab CI/CD YAML pipeline configuration file
     (".gitlab-ci.yml") for your project.
Don't explain the file; just generate it.
```

It then formulates a specific instruction (**Instruction with Sample Input**) and provides the example pipeline specification already shown in Section 3.2. Our tool then provides the LLM with a sample response (**Instruction with Sample Output**), containing a corresponding output GitLab CI/CD pipeline configuration as follows:

```
default:
  image: python:3.9-bullseye
  before_script:
    - pip install pettingzoo[butterfly]>=1.24.0 stable-
         baselines3>=2.0.0 supersuit>=3.9.0

stages:
  - train
  - eval

variables:
  MODEL_PATH: ./models
  TRAIN_STEPS: 10000

train:
  stage: train
  tags:
    - gpu-runner
  script:
    - mkdir -p $MODEL_PATH
    - python ./rl_script.py --train --steps $TRAIN_STEPS
  artifacts:
    paths:
      - $MODEL_PATH

eval:
  stage: eval
  tags:
    - gpu-runner
  script:
    - python ./rl_script.py --eval
  needs:
    - job: train
      artifacts: true
```

The program then assembles the conversation by combining the system message, the query, the example pipeline specification, the example GitLab CI/CD pipeline configuration file response and the actual pipeline specification from which to generate a GitLab CI/CD pipeline configuration file (**Instruction with Target Input**). If errors or warnings occur, the tool attempts to correct the mistakes by retrying a configurable number of times before selecting the response with the fewest errors or warnings.

3.4 Pipeline Generation Architecture

Our tool comprises components with dedicated responsibilities. **Generator components** include the central elements that execute the generation or detection flows using an LLM. **LLM components** abstract the functionalities of specific LLMs and facilitate the generation of RLOps pipeline configurations based on provided specifications. **Conversation components** manage interactions with the LLM by abstracting conversation instances and maintaining context and interaction state. A **data transfer object** [30] is passed through generation steps in the Pipes and Filters implementation. It contains a generation report that is built as the generation and validation progress. The sequence of operations, such as generation, detection and validation, can be executed sequentially or in parallel. **Detection components** can, for instance, check for errors in YAML syntax, the stage order of generated pipelines, whether a generated pipeline configuration contains code blocks, or whether a pipeline stage generates artefacts. A **validation component** might use the GitLab API to check whether the generated pipeline configuration is valid. **Supporting components** are used, for example, to commit the generated pipeline and its project to GitLab.

The execution flow is defined using a simple Python DSL that implements the Pipes and Filters pattern and specifies the generation, detection and validation steps. This structured generation flow, with focused steps, contextual feedback, validation and retries, helps address accuracy challenges such as AI hallucinations and incorrect classifications that are common to LLMs [31–33].

```
LCF(components=[
    GenerationRetry(
      GenerateGitlabCIYmlFileFromSpecFile,
        ParallelSuccessfulIfAllSuccessful(
          [
            GitlabCINoLinterIssuesDetection,
            GitlabCIArtifactsCreatedDetection,
            GitlabCICheckStageOrderFromSpecFile
          ]
        )
      ),
      GitlabCommitGeneratedProject])
```

In the above example, we use retry logic (`GenerationRetry`) to generate a GitLab CI/CD YAML file from a specification file (`GenerateGitlabCIYmlFileFromSpecFile`). We then run error detection measures in parallel (`ParallelSuccessfulIfAllSuccessful`): linting the generated GitLab pipeline configuration file (`GitlabCINoLinterIssuesDetection`), verifying that the pipeline configuration creates artefacts (`GitlabCIArtifactsCreatedDetection`) and checking that the pipeline has the correct order of stages (`GitlabCICheckStageOrderFromSpecFile`). We then commit and push the generated pipeline configuration with its project to GitLab (`GitlabCommitGeneratedProject`) for execution.

3.5 Integration and Workflow

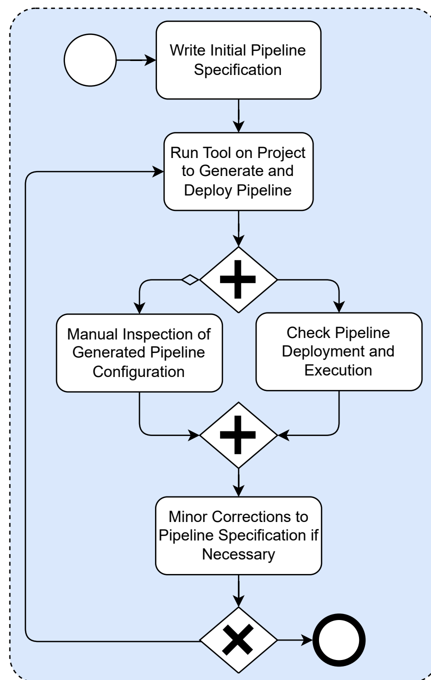


Figure 2: Detailed workflow steps for a user of the tool.

Figure 2 shows how a practitioner can use our tool. In the first step (**Write Initial Pipeline Specification**), the practitioner writes an initial version of the pipeline specification for their RL project, based on the template example in Section 3.2. The user then runs our tool on their specification file and project (**Run Tool on Project to Generate and Deploy Pipeline**). Our tool uses an LLM to generate the GitLab CI/CD pipeline configuration file from the specification, which is then automatically committed to a GitLab repository. As the pipeline is running, the practitioner can keep an eye on the pipeline's execution status, i.e. that it deployed and executed successfully (**Check Pipeline Deployment and Execution**) and, if necessary, note any errors that occur. Simultaneously, the practitioner can inspect the generated pipeline configuration (**Manual Inspection of Generated Pipeline Configuration**) to verify that the pipeline configuration file is acceptable, i.e. that there are no obvious mistakes. Suppose an error is apparent in the generated pipeline configuration or its execution. In that case, the practitioner can make corrections to the pipeline specification (**Minor Corrections to Pipeline Specification if Necessary**) and rerun the tool to generate a new pipeline configuration if required. The resulting workflow, therefore, includes an iterative feedback loop that allows the user to continuously refine the pipeline specification.

4 Approach Validation

4.1 Context

We use an RL project for optimising factory manufacturing (provided in our replication package) based on Siemens' industrial pipelines. To protect intellectual property, the industrial project was adapted to remove proprietary elements, retaining the key RLOps features, such as multiple pipeline stages, artefact management and deployment phases.

The project consists of components that train RL models, and a production simulator that optimises factory production in a decentralised yet collaborative manner between cyber-physical production units by optimising key performance indicators. A custom environment defined in `factory_robot_env.py` models the environment within which a factory robot will operate. This environment features continuous observation and discrete action spaces, enabling realistic simulation of robotic decision-making.

The agent training and evaluation process is coordinated through scripts in the file `sb3_agent.py`. These scripts automate training, evaluate the agents' performance across multiple episodes during training and validate the agents' predictions. The training hyperparameter configuration, evaluation settings and file paths are stored in `config.py`. Template-based low-code CI/CD specifications used as input to LLMs are provided as `spec.txt` files.

Our project emulates the required pipeline stages. The `test` stage lints the Python source files based on `.pylintrc`, then runs the unit tests from `test_factory_robot_env.py`, `test_sb3_agent.py` and `test_sb3_agent_runtime.py`. The `train` stage then trains a model and provides it as an artefact to subsequent stages. The `model_test`, `eval` and `release` stages run in parallel and use the model artefact provided by `train`. The `model_test` stage validates the trained model's predictions, whilst `eval` evaluates the model based on its mean reward and standard deviation. The `release` stage builds a Docker container image containing the model and an executable application that runs it. The `deploy` stage deploys the model.

4.2 Validation Scenarios

We defined four sequential evaluation scenarios corresponding to typical RLOps CI/CD workflow tasks: 1) ***Create a New Pipeline***: a new pipeline configuration is specified by the practitioner and created by the LLM, 2) ***Edit the Pipeline***: artefacts are set to expire in one week, the model path environment variable is changed and the `gpu-runner` tag is specified in all stages except `deploy`, 3) ***Add to the Pipeline***: a `release` stage and its corresponding source files are added, as are unit tests, and the user specifies that the `release` stage should not include a tag to identify a runner, and 4) ***Remove from the Pipeline***: unused environment variables, files, the `release` stage and any `release`-relevant specification text and files are removed.

4.3 Large Language Models

Our validation uses seven popular LLMs: OpenAI GPT-3.5 Turbo (Version 2023-06-13), OpenAI GPT-4o (Version 2024-08-06), Qwen2.5 72B, Qwen2.5 Coder 32B, Llama 3.3 70B, Code Llama 70B and DeepSeek R1 70B. We selected the LLMs from the standard benchmark EvalPlus [34], which is based on peer-reviewed studies and provides a leaderboard of LLMs for code generation tasks.

4.4 Validation Variables

For validation, we designed variables that were specific enough to be usable but general enough to apply to any MLOps or RLOps pipeline with minimal to no modification. We defined them in a structured manner based on our knowledge of RLOps pipeline requirements, combining domain-specific knowledge of RL processes and LLM applications and decomposing relevant aspects of RLOps pipelines into quantifiable, discrete items reflecting relevant pipeline properties and behaviour.

We thus assessed pipeline generation quality across 31 Boolean variables in three categories: 1) ***generation tasks*** (e.g. correct Python images, required stages in order and correct artefact management), 2) ***natural language interpretation*** (e.g. job dependencies), and 3) ***run-time execution*** (e.g. successful testing, training and deployment). Siemens verified that these variables fairly represent aspects of Industry 4.0 RLOps pipeline requirements. Our replication package contains definitions of the validation variables under `_evaluation/evaluation_variable_definitions.md`.

4.5 Validation Results

We executed each validation scenario three times for each LLM and calculated average error rates by comparing generated pipelines to reference configurations and evaluating execution outcomes against expected outcomes. Our approach successfully generated RLOps pipeline configurations with low error rates, demonstrating viability for industrial deployment. GPT-4o achieved zero errors across all criteria. Qwen2.5 Coder and GPT-3.5 Turbo also performed well with average error rates of 0.05 and 0.07, respectively. Multiple LLMs achieved acceptable performance (error rates ≤ 0.1), indicating that pipeline generation is not dependent on a single LLM instance.

5 Discussion

RQ1 *How accurately can a low-code, template-based, modular flow approach using LLMs generate Industry 4.0-specific RLOps pipeline configurations?* Our approach was highly effective at generating RLOps pipelines during our Industry 4.0 RLOps validation, with low error rates and high accuracy across five LLMs, including a perfect zero-error rate for GPT-4o.

To put the effectiveness of our approach into perspective, we informally compared it with the traditional manual approach to developing pipeline configurations. Based on feedback from Siemens, manual configuration of similar RLOps pipelines usually takes many hours for both experienced DevOps engineers and domain experts. Our template-based method accurately generates pipeline configurations and validates them in around two to three minutes once the simple low-code template is filled out.

RQ2 *How do popular LLMs perform in generating Industry 4.0-specific RLOps pipeline configurations in real industrial contexts?* Our analysis demonstrates that OpenAI's GPT-4o model performed best across all task types, achieving zero errors for all validation variables. Qwen2.5 Coder and GPT-3.5 Turbo also performed exceptionally well. Qwen 2.5 and Llama 3.3 are also usable in our validation, with low variability in their error rates and a high degree of consistency, reliability and accuracy in generating correct outputs. DeepSeek and CodeLlama did not perform very well.

6 Threats to Validity

Construct Validity The effectiveness of natural language-based specification components depends heavily on users' correct specifications. The validity of results could be affected when human inputs vary, leading to inconsistencies in the generated outputs. We ran multiple tests with different LLMs and scenarios to smooth out unexpected outcomes.

External Validity The single RL industry case we examine may not be transferable to other industrial projects. Our limited access to ML and RL projects in the industry could lead us to miss relevant pipeline engineering and RLOps practices. However, we did not set out to evaluate a complete set of practices; instead, we focused on use cases common to developing RLOps pipelines.

Internal Validity Our results depend on the completeness of the specification template, the correctness of completed specification template instances and the LLMs' interpretations of the provided specifications. Our template demonstrates sufficient generality and specificity to produce correct results across multiple LLMs, as evidenced by the validation results.

Conclusion Validity The results of our experiments rely on a single validation and a limited number of use cases. Broadening the study's scope by including more industry projects and use cases could strengthen the validation of our approach and conclusions. Since we have limited access to industry projects, we addressed what we could by creating realistic scenarios.

7 Conclusion and Future Work

This paper presents an approach to automatic pipeline configuration in Industry 4.0. Our work includes contributions that combine correct pipeline configuration generation from our template-based implementation and validation of our tool using LLMs to transform specifications into RLOps CI/CD pipeline instances.

Future work involves integrating our approach with Siemens, adding model lifecycle management practices to generated pipelines and assessing the effectiveness of the new features. We also plan to test our approach in a user study, comparing it with manual creation of CI/CD pipeline configurations.

Acknowledgements

This research was funded in whole or in part by the FFG (Austrian Research Promotion Agency) projects MODIS (no. FO999895431), BEAM (no. FO999933314) and the Austrian Science Fund (FWF) project CQ4CD (Grant-DOI: 10.55776/I6510). For open access purposes, the authors have applied a CC BY public copyright licence to any author-accepted manuscript version arising from this submission.

References

[1] M. Treveil, N. Omont, C. Stenac, K. Lefevre, D. Phan, J. Zentici, A. Lavoillotte, M. Miyazaki, and L. Heidmann, *Introducing MLOps*. O'Reilly Media, 2020.

[2] Y. Zhou, Y. Yu, and B. Ding, "Towards MLOps: A Case Study of ML Pipeline Platform," in *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, Oct 2020, pp. 494–500.

[3] M. M. John, H. H. Olsson, and J. Bosch, "Towards MLOps: A framework and maturity model," in *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2021, pp. 1–8.

[4] S. J. Warnett and U. Zdun, "Architectural design decisions for the machine learning workflow," *Computer*, vol. 55, no. 3, pp. 40–51, 2022.

[5] ——, "Architectural Design Decisions for Machine Learning Deployment," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, 2022, pp. 90–100.

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

[7] P. Li, J. Thomas, X. Wang, A. Khalil, A. Ahmad, R. Inacio, S. Kapoor, A. Parekh, A. Doufexi, A. Shojaeifard *et al.*, "RLOps: Development life-cycle of reinforcement learning aided open RAN," *IEEE Access*, vol. 10, pp. 113 808–113 826, 2022.

[8] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014. [Online]. Available: https://doi.org/10.1007/s12599-014-0334-4

[9] H. Singh and B. Singh, "Industry 4.0 technologies integration with lean production tools: a review," *The TQM Journal*, 02 2024.

[10] L. Monostori, "Cyber-physical Production Systems: Roots, Expectations and R&D Challenges," *Procedia CIRP*, vol. 17, pp. 9–13, 2014, Variety Management in Manufacturing.

[11] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals*, vol. 65, no. 2, pp. 621–641, 2016.

[12] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, 1st ed. Addison-Wesley Professional, 2015.

[13] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[14] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1723–1726.

[15] S. A. A. Naqvi, M. P. Zimmer, P. Drews, K. Lemmer, and R. Basole, "The Low-Code Phenomenon: Mapping the Intellectual Structure of Research," in *Hawaii International Conference on System Sciences*, 2024.

[16] H. El Kamouchi, M. Kissi, and O. El Beggar, "Low-code/No-code Development: A systematic literature review," in *2023 14th International Conference on Intelligent Systems: Theories and Applications (SITA)*, 2023, pp. 1–8.

[17] P. Sharma and M. S. Kulkarni, "A study on unlocking the potential of different AI in Continuous Integration and Continuous Delivery (CI/CD)," in *2024 4th International Conference on Innovative Practices in Technology and Management (ICIPTM)*. IEEE, 2024, pp. 1–6.

[18] I. Kumara, R. Arts, D. Di Nucci, W. J. Van Den Heuvel, and D. A. Tamburri, "Requirements and reference architecture for MLOps: insights from industry," *Authorea Preprints*, 2023.

[19] A. del Real Torres, D. S. Andreiana, Á. Ojeda Roldán, A. Hernández Bustos, and L. E. Acevedo Galicia, "A Review of Deep Reinforcement Learning Approaches for Smart Manufacturing in Industry 4.0 and 5.0 Framework," *Applied Sciences*, vol. 12, no. 23, 2022.

[20] E. Commission, D.-G. for Research, Innovation, M. Breque, L. De Nul, and A. Petridis, *Industry 5.0 – Towards a sustainable, human-centric and resilient European industry*. Publications Office of the European Union, 2021.

[21] T. Kegyes, Z. Süle, and J. Abonyi, "The Applicability of Reinforcement Learning Methods in the Development of Industry 4.0 Applications," *Complex.*, vol. 2021, pp. 7 179 374:1–7 179 374:31, 2021.

[22] M. Hirzel, "Low-Code Programming Models," *Commun. ACM*, vol. 66, no. 10, p. 76–85, Sep. 2023.

[23] O. Bruhin, E. Dickhaut, E. Elshan, and M. M. Li, "The Rise of Generative AI in Low Code Development Platforms - An Analysis and Future Directions," in *Hawaii International Conference on System Sciences*, 2024.

[24] H. Jin, H. Chen, Q. Lu, and L. Zhu, "Towards Advancing Code Generation with Large Language Models: A Research Roadmap," *arXiv preprint arXiv:2501.11354*, 2025.

[25] D. Li and L. Murr, "HumanEval on Latest GPT Models–2024," *arXiv preprint arXiv:2402.14852*, 2024.

[26] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating Large Language Models Trained on Code," 2021.

[27] S. J. Oks, M. Jalowski, M. Lechner, S. Mirschberger, M. Merklein, B. Vogel-Heuser, and K. M. Möslein, "Cyber-Physical Systems in the Context of Industry 4.0: A Review, Categorization and Outlook," *Information Systems Frontiers*, 2022.

[28] S. J. Warnett and U. Zdun, "RLOps Pipeline Development with Low-Code and Large Language Models for Industry 4.0: Replication Package," Jan. 2026. [Online]. Available: https://doi.org/10.5281/zenodo.18320785

[29] DAIR.AI, "Prompt Engineering Guide," https://www.promptingguide.ai/techniques, 2024.

[30] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley, 2012.

[31] C. Parnin, G. Soares, R. Pandita, S. Gulwani, J. Rich, and A. Z. Henley, "Building Your Own Product Copilot: Challenges, Opportunities, and Needs," *arXiv preprint arXiv:2312.14231*, 2023.

[32] K. Jesse, T. Ahmed, P. T. Devanbu, and E. Morgan, "Large Language Models and Simple, Stupid Bugs," in *Proc. 20th International Conference on Mining Software Repositories (MSR)*. IEEE/ACM, May 2023, pp. 563–575. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MSR59073.2023.00082

[33] Z. Xu, S. Jain, and M. Kankanhalli, "Hallucination is inevitable: An innate limitation of large language models," *arXiv preprint arXiv:2401.11817*, 2024.

[34] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.