A Model-Driven, Metrics-Based Approach to Assessing Support for Quality Aspects in MLOps System Architectures

Stephen John Warnett[a,b,*], Evangelos Ntentos[a], Uwe Zdun[a]

[a]*Research Group Software Architecture, Faculty of Computer Science, University of Vienna, Währinger Straße 29,1090 Vienna, Austria,*
[b]*UniVie Doctoral School Computer Science DoCS, Faculty of Computer Science, University of Vienna, Währinger Straße 29, 1090 Vienna, Austria,*

Abstract

In machine learning (ML) and machine learning operations (MLOps), automation serves as a fundamental pillar, streamlining the deployment of ML models and representing an architectural quality aspect. Support for automation is especially relevant when dealing with ML deployments characterised by the continuous delivery of ML models. Taking automation in MLOps systems as an example, we present novel metrics that offer reliable insights into support for this vital quality attribute, validated by ordinal regression analysis. Our method introduces novel, technology-agnostic metrics aligned with typical Architectural Design Decisions (ADDs) for automation in MLOps. Through systematic processes, we demonstrate the feasibility of our approach in evaluating automation-related ADDs and decision options. Our approach can itself be automated within continuous integration/continuous delivery pipelines. It can also be modified and extended to evaluate any relevant architectural quality aspects, thereby assisting in enhancing compliance with non-functional requirements and streamlining development, quality assurance and release cycles.

Keywords: software architecture quality, metrics, distributed system modelling, distributed software architecture, MLOps, machine learning

1. Introduction

Machine learning (ML) systems pose unique challenges compared to traditional software architectures due to the intricate interplay of software development, data science, data engineering, ML models as a service and ML-related end-service engineering in the form of machine learning operations (MLOps) and its parallels with DevOps. Noteworthy challenges in ML application software structure include complex dependencies among software modules [1], technical debt, anti-patterns [2] and design challenges in ML model engineering such as model selection and reuse [3, 4]. Model provisioning and production runtime monitoring of ML service performance, often in a distributed environment, further exacerbate complexity and underscore the intricacies in organising and decomposing ML systems. Paleyes et al.'s [5] survey of case studies identifies practical challenges in ML system architecture, encompassing tools, services, and architectures within the deployment structure, as well as potential disparities between ML and data science, and software engineering practices. This complexity has led to the emergence of Automated ML [6, 7] and MLOps [8–10] as sub-disciplines within DevOps [11, 12] and continuous delivery (CD) [13].

Studies on open source systems [14], scientific literature [15], and practitioner insights [16] highlight that adopting DevOps and CD results in a complex deployment and delivery architecture. This complexity extends beyond the software architecture of the intended system. It becomes even more pronounced in MLOps systems, where additional aspects such as ML model management and deployment, continuous integration/continuous delivery (CI/CD) pipeline behaviours, ML pipeline behaviours, and ML-specific component architectures [8, 9, 17] must be considered. Compounding the complexity of MLOps systems are the myriad tools and technologies available [8, 9, 18, 19].

Whilst the introduction of MLOps is welcome, the complexity of MLOps systems as described above and the variety of practices [20–22] make it difficult to determine which *Architectural Design Decisions* (ADDs) were applied. The multitude of supporting technologies and programming languages make it even more challenging to achieve this programmatically and generically, especially through source code parsing. It is just as challenging to determine which decision options an architect selected, which specific patterns and practices were applied, and the extent of support for core architectural quality attributes such as automation, model management and simplified deployment of models in the systems. Within MLOps systems, as exemplified in this study, implementing an automated assessment method would significantly augment understanding of the overall properties of the systems and their alignment with preferred practices and desired qualities. Background on the most important MLOps and ADD concepts and terms used in this study can be found in Section 2.

The necessary understanding outlined above is particularly pertinent in the realm of automation in MLOps systems be-

*Corresponding author
Email addresses: stephen.warnett@univie.ac.at (Stephen John Warnett, evangelos.ntentos@univie.ac.at (Evangelos Ntentos), uwe.zdun@univie.ac.at (Uwe Zdun)

cause the level of automation is directly influenced by practices related to integration and delivery, such as data processing and pipeline triggering — especially within an automated CD framework [23]. MLOps systems supporting automation help to streamline and accelerate the deployment and management of ML models in production environments. Teams can iterate quickly, maintain consistency, and ensure reliability in ML workflows. Automation also reduces manual effort, minimises errors, and enhances efficiency, allowing organisations to deploy and maintain ML models at scale whilst improving time-to-market and overall productivity.

In previous work [20, 21], we identified several ADDs along with corresponding decision options and their respective decision drivers (which are synonymous with, or closely related to, quality aspects). In those studies, we achieved theoretical saturation, which is the stage in a grounded theory study where the collection and analysis of further data no longer produce any new theoretical insights. It is important to note that the work described in this paper is not based solely on our prior work. Our findings from those studies were also noted in related work (see Section 3). Indeed, we studied much scientific and practitioner literature from other authors [8, 19, 22, 24–28] since our initial studies were published, and these studies corroborated our original findings based on grey literature. Note that a renewed search of practitioner and scientific literature before commencing this study did not yield any new insights concerning ML ADDs. Therefore, we concluded that there was no need for a further extensive literature review.

Several crucial quality attributes for ML systems that are essential for ensuring their effectiveness and robustness include performance efficiency, which ensures optimal resource use, and maintainability, which focuses on ease of modification for improvements; scalability and portability, which address a system's ability to adapt to different capacities and environments; and interoperability, which ensures seamless information exchange between systems. Process and work automation in ML is also a significant quality attribute — automation enhances consistency, reduces human error, and accelerates the deployment of ML models. Key automated processes include CI/CD pipelines, automated testing, and model monitoring. These processes ensure that ML models are consistently validated, updated, and maintained without manual intervention, leading to improved reliability and efficiency in production environments. Automation thus plays a critical role in maintaining high quality and operational excellence in MLOps systems. Together, these attributes help in developing high-quality ML systems with robust architectural designs in MLOps. Note that we are not assuming any particular automation concept, paradigm or set of predefined practices. In this study, we modelled and manually assessed twenty-two system architectures, each of which followed its particular level of automation and set of practices (see Section 6).

The ADDs selected for this study are a subset of those discovered in the aforementioned work, specifically those linked to automation-related factors since the study is concerned specifically with automation-related aspects of ML architectures and automation-relevant practices. The ADDs under consideration are *"How to Automate Integration and Delivery in a Machine Learning Context?"*, *"How to Trigger a Machine Learning Pipeline or Orchestrator?"* and *"How to Automatically Process the Data Used for Model Building?"* and are described in detail in Section 5 and listed in the first column of Table 2 together with their respective decision options. The ADDs are derived from established practices found in grey literature, such as informal guidelines for practitioners, blog posts, public repositories, and similar sources. Building upon this architectural knowledge, we aim to automatically evaluate the degree of architectural conformance to automation-relevant practices within modelled MLOps system architectures. Thus, this paper aims to explore whether it is possible to determine the level of support for automation as a central quality aspect within MLOps systems.

The focus of this paper is thus to address the following research questions (RQs):

- **RQ1** How can MLOps systems' support for the core quality aspect of automation be assessed in the context of Architectural Design Decisions and their respective decision options?
- **RQ2** What types of metrics can be applied to evaluate these levels of support, and how effective are they?

To tackle these research questions, we introduced a set of metrics designed for diverse ADDs related to automation, encompassing all known decision options. We describe a model-driven, metrics-based methodology that relies on modelling elements solely obtainable from MLOps systems, architectural descriptions, reference architectures, or industrial architecture guides. These quantifiable system models encapsulate the architecture, properties, and behavioural aspects of the systems, thereby facilitating the evaluation of support for ADDs and enabling automated measurement for quality assessment of MLOps system architectures via custom metrics.

Our approach involves establishing a ground truth through the manual evaluation of a set of system models, gauging their adherence to the full spectrum of considered decision options and their combinations. Specifically, the pertinent ADDs and their associated decision options focus on *Integration and Delivery*, *Pipeline Triggers*, and *Data Processing*.

The ground truth creation entails objectively assessing each decision option's presence and level of support in a system. We develop an ordinal rating scheme, establishing a scale to measure the degree of automation support.

Implementing detectors in Python for analysing modelled systems and calculating our custom metrics, we assess the effectiveness of our metrics in predicting the ground truth through ordinal regression analysis. We show that simple yet effective metrics provide reliable insights, particularly regarding automation, which is crucial for practitioners. Our prediction models demonstrate remarkable accuracy in predicting the ground truth assessment. Our approach can be automated within a CI/CD pipeline to help identify complex architectural features like ADDs and quality attributes, streamline development cycles and ensure compliance with non-functional requirements.

The core contributions of this paper are (1) a novel approach for the semi-automated, model-driven and metrics-based statistical assessment of conformance of MLOps systems to automation-related ADDs; (2) detailed modelling of twenty-two MLOps system architectures and the continued development of a reusable MLOps metamodel; (3) the definition and application of novel, technology-agnostic automation-specific metrics for assessing MLOps systems.

In contrast to our prior works, which investigated the ML workflow and identified workflow and deployment-related ADDs, this paper considers automation metrics and their quantification for the assessment of automation support as an MLOps system quality.

Another novel contribution of our study is the systematic process employed in sourcing, modelling, and assessing various MLOps system architectures. Whereas our prior work mostly focused on the discovery of ML-related ADDs or modelled only certain aspects of very few systems, this study comprehensively models a number of MLOps system architectures in great detail and develops an extensive MLOps metamodel.

Our paper offers another contribution to the MLOps architecture field by providing new technology-agnostic metrics tailored to evaluate support for automation-related ADDs. We provided empirical validation of the metrics by performing ordinal regression analysis to develop reliable and reusable prediction models. In contrast, our prior work was more qualitative than quantitative.

This paper is organised as follows: Section 2 provides background on MLOps and ADD concepts and terms. Section 3 compares related work, whilst Section 4 outlines our research methods, ground truth definition, and the system modelling approach. Section 5 details the ADDs and associated decision options considered in the study. The assessment process for the ground truth is explained in Section 6. Moving forward, Section 7 introduces our proposed metrics, and Section 8 presents the results of the metric calculations and ordinal regression analysis. Section 9 discusses our research questions, results, a practical application of our approach and potential threats to validity. Future work is discussed in Section 10, and Section 11 provides concluding remarks.

2. Background

This section provides background information on MLOps concepts and terminology as they are understood in this study, as well as ADD concepts and terms.

2.1. MLOps: Concepts and Terms

MLOps is a holistic approach to the ML workflow involving training, deploying, and managing ML models in production environments [8]. It is rapidly becoming an essential approach for engineering ML systems, as it defines processes and practices for models' availability and stability throughout their lifetime. Nevertheless, MLOps is still at a somewhat early stage, and there is only sometimes agreement or consensus regarding what MLOps as a relatively new set of practices actually means [18, 24, 25, 29]. To aid understanding of the main concepts, we will describe how MLOps is defined in this work and how it relates to DevOps and the deployment and management of ML systems as a whole. Note that this section provides a high-level general description of the concepts and terminology. We do not claim that any particular set of practices is correct or prescribe rules for how MLOps systems should be structured, nor do we prescribe links between concepts (e.g. validation) and particular practices (e.g. use of a CI/CD pipeline).

Although MLOps refers specifically to ML tasks, it has many similarities with DevOps: the optimisation of collaboration and automation of the processes of software and ML deployment. Both adopt cross-functional collaboration, process automation, integration/automation of processes and utilisation of CI/CD pipelines for multiple iterations and builds/deployments. They also convey the basic idea of monitoring their performance on an ongoing basis and providing feedback for performance enhancement. However, as a separate practice, MLOps targets concerns related to data, model life cycle, model monitoring and experimentation, which are not typical concerns of DevOps. Therefore, it enables the blending of conventional DevOps approaches with an ML-suitable workflow and practices [24].

The exemplary MLOps workflow, shown in Figure 1, depicts several key steps. The following are the five stages that are typically involved: pipeline triggering, data ingestion and processing, model building, model deployment and model monitoring.

Triggers are used to automatically initiate certain actions, pipelines or workflows based on predefined events or conditions [27]. As in DevOps, CI/CD pipelines often make use of triggers to initiate processes - in the case of ML, to automate the ML workflow. For example, a trigger could be set up to automatically start the data ingestion process when new data becomes available in a data repository or when model performance degrades beyond a specific threshold.

Data ingestion is the stage where data is introduced into the system. This data may be ingested, for example, by request, automatically streamed, provided in batches or provided manually [28]. The raw data may be stored in a data store and then passed onto a further pipeline or pipeline step for processing.

In the data processing [30] stage, the raw data are cleaned, preprocessed and transformed into a suitable format for use in model training. Various types of solutions exist for performing data processing, for instance a data pipeline, an ETL pipeline or a data processing component. Features for use in model training may also be engineered or extracted at this point. This stage is for preparing data that is suited for model training since poorly prepared data would lead to a reduction in the model's performance. Feature stores are used to store the processed data and their features used for training and evaluating models. Both data-related steps are not needed for machine learning that is not based on data sets, e.g. reinforcement learning.

Model building [8] involves using machine learning algorithms on the prepared data to identify patterns and relationships. Selecting the proper algorithms and hyperparameters may greatly affect the performance of the final model, and the best configuration has to be searched depending on the prob-

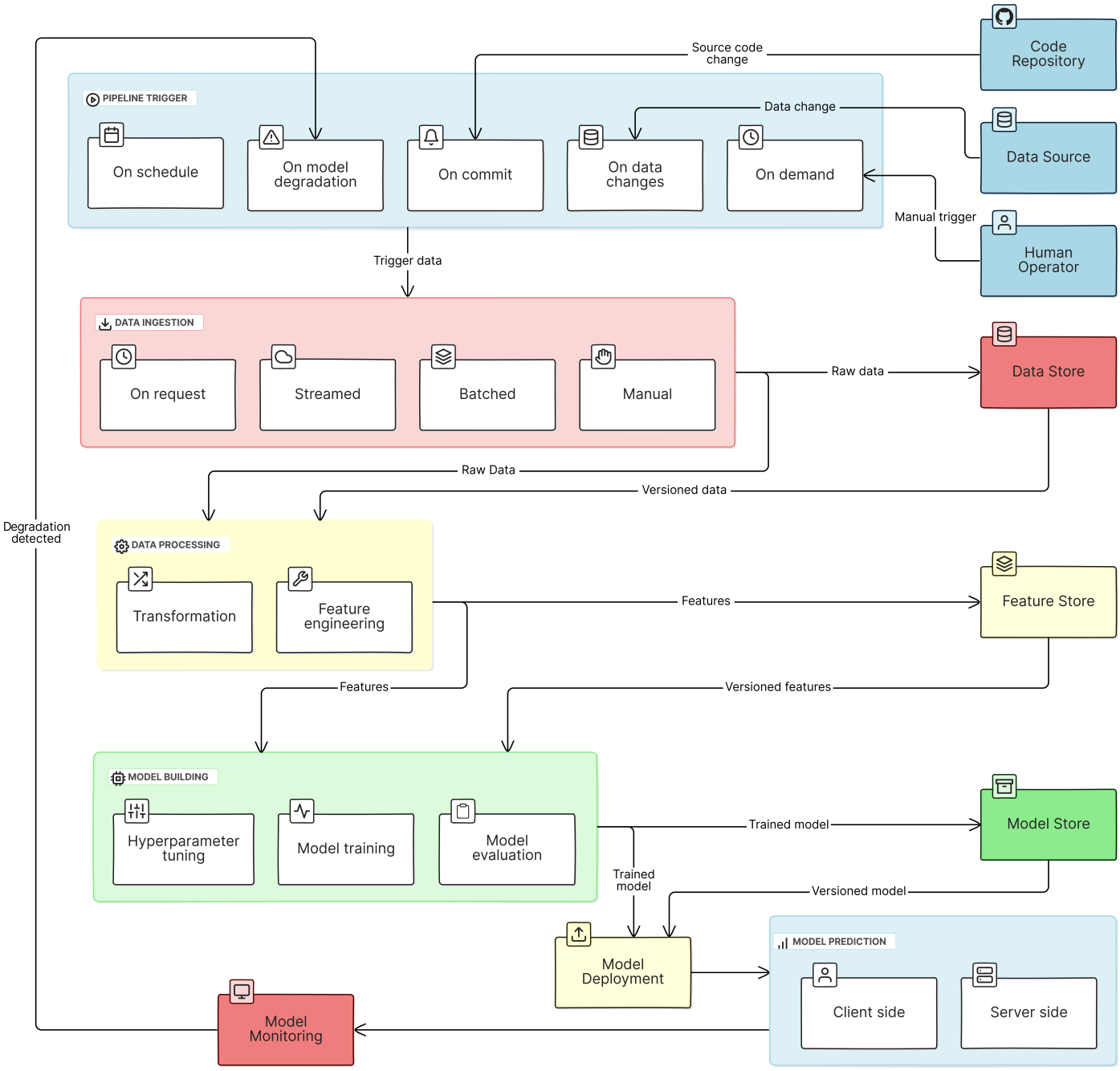

Figure 1: A high-level abstraction of common ML workflow tasks in MLOps.

lem. Models are also validated at this stage. Validation is the process of assessing the performance of the trained model on unseen data. It involves evaluating model metrics and performance indicators like accuracy, precision, recall, and F1 score to ascertain the suitability of a trained model. Model evaluation is essential for ensuring that trained models are performing as expected and for identifying areas where improvements can be made.

The deployment of models [31] makes trained models available for use on target systems. This process involves the packaging of the model and its dependencies into a deployable artefact, such as a Docker container or a serverless function. Through this process, it is ensured that models can be easily and consistently deployed across various environments - development, staging, and production. The delivery of models can be performed, for instance, via build and deployment scripts, in a custom CI/CD pipeline or by an ML orchestrator. For the archiving and versioning of trained ML models and their related artefacts, model repositories can be utilised. These artefacts include model files, as well as related model metrics and metadata. Such versioning of models enables, for example, the possibility of rolling back to previous model versions when necessary.

After a model has been deployed to a production environ-

ment, the continuous assessment of the model's performance through model monitoring begins and involves the collection and analysis of the model's inputs, outputs, and performance metrics to identify any performance degradation. Through model monitoring, it is ensured that the model continues to perform as expected over time. In the tracking of model performance degradation, factors such as prediction latencies, errors, and model or environmental drift can also be considered. As previously mentioned, should model performance degradation be detected, the ML workflow may be automatically triggered[27].

2.2. Architectural Design Decisions: Concepts and Terms

In architectural design, there are some critical decisions – ADDs – to make that form the base on which a software system is constructed. These ADDs establish the architecture and layout of the system and its components and integration alongside highly influencing quality aspects such as speed, extensibility, modularity, security, and modifiability. Of significance in ADDs are *relations*, which refer to connections and interactions between system components [32]. These relations encapsulate aspects like functionality, communication, and data flow.

Software architects have to consider a number of factors when deciding on the architectural design of the system. These factors include functionalities required from the system, the environment that is available and its capacity, necessary tools and technologies, as well as the abilities and preferences of the development team. They also need to make judgements on the relative costs and benefits of various decision options and their impacts on quality characteristics, such as performance, reliability and security [33]. These factors can be considered *decision drivers* (or *forces*) in the context of ADDs, and they encompass the dynamic aspects and constraints shaping the decisions. These aspects can be external or internal, spanning quality aspects, functional and non-functional requirements, technological limitations, organisational policies, and market conditions. Software architects balance the trade-offs among forces to achieve desired outcomes, addressing stakeholder needs effectively through informed decisions.

When making architectural design decisions, software architects often utilise tools and techniques that include but are not limited to elements such as architectural patterns, architectures, and decision frameworks. They also make joint decisions with other stakeholders including developers, project managers, and the end-users on the decisions made. *Patterns* [34] and *practices* refer to established guidelines, methodologies, techniques, and approaches practitioners apply to make informed decisions and design effective software architectures. They are typically derived from industry best practices, architectural principles, and lessons learned from previous projects and described in practitioner literature. They represent decision options that can be selected depending on preferred forces.

3. Related Work

This section delves into the existing research concerning ML practices, patterns, quality aspects and metrics, and we draw comparisons between this body of related work and our study.

3.1. Patterns and Practices and Their Relation to Quality Aspects

ML and MLOps systems have grown in complexity and become more prevalent within the industry. Consequently, more scientific research cataloguing and documenting quality aspects and related patterns and practices is being conducted. Patterns and practices are relevant to ADDs because they represent specific decision options, which are then, in turn, associated with decision drivers (forces), which represent quality aspects that may be of interest to practitioners and help guide the practitioner to the most suitable design decisions.

Sharma and Davaluri [35] detected and analysed design and architectural patterns in two ML applications. They identified various quality attributes of relevance, including reusability, flexibility and testability, which are critical for the sustainable development and maintenance of ML systems.

Washizaki, Uchida et al. [36] analysed ML systems' engineering, architecture, and design (anti-)patterns, suggesting that improving crucial ML quality aspects such as complexity, performance, reliability, and ML model quality can be achieved through best practices.

In Washizaki, Khomh et al.'s [37] multivocal literature review, including an ML practitioner survey, they identified various software engineering design patterns for ML applications. They propose that adopting these patterns could enhance system and software quality attributes, such as usability, reliability, and maintainability, along with ML quality attributes, like ML model robustness, explainability, and fairness.

Kreuzberger et al. [8] conducted mixed-method research, including a literature review, tool assessment, and expert interviews, providing a comprehensive overview of diverse ML aspects like components, architecture, and workflows. They connect principles and practices, exploring the automation potential of manual ML processes and highlighting the importance of automation in MLOps.

Faubel and Schmid [22] conducted a pilot case study assessing MLOps applications in the industry. They observed MLOps practices, architectural patterns, and diverse automation types in surveyed companies. They noted commonalities with aspects relevant to this study, including pipeline triggers, CI/CD automation, pipeline automation, scripts and orchestrators.

A systematic literature review conducted by Lima et al. [26] focused on practices, patterns, roles, maturity models, challenges, and tools for automating operational activities in ML model deployment. Emphasising the significance of automation in deploying ML models, they recognised that existing maturity models indirectly gauge activities in ML model development. However, they highlighted the absence of direct assessments related to MLOps, suggesting an opportunity for further research.

The cited related works collectively explore various facets of ML and MLOps systems, often highlighting practices and patterns similar to those considered in this study and their relation to quality aspects. They acknowledge the importance of automation in MLOps systems, aligning with our understanding

of automation as a core quality attribute. Automation streamlines processes, reduces human error, and enhances efficiency — all critical factors in MLOps systems. Moreover, automated pipelines and workflows enable scalability, allowing organisations to handle increasingly large and complex datasets and models.

However, our approach differs in focus and methodology from the related work described previously, as we leverage our knowledge of patterns, practices and automation as a quality attribute to develop a model-driven, metrics-based approach for assessing support for automation in the context of ADDs in MLOps systems. By integrating automation assessment directly into the architectural design and development phases, quality assurance for automation can be employed from the outset, ensuring that MLOps systems are designed with automation in mind. This proactive approach minimises the need for retrofitting automation into existing systems, reducing technical debt and accelerating time to market for ML solutions.

Furthermore, our metrics-based approach provides quantifiable measures of automation support, enabling stakeholders to make informed decisions about the trade-offs between automation and other quality attributes. By quantifying automation readiness at the architectural level, organisations can prioritise investments in automation infrastructure and tooling, strategically allocating resources to maximise automation benefits whilst mitigating associated risks. Ultimately, our study extends beyond acknowledging the importance of automation in MLOps systems to providing an approach for evaluating and enhancing automation support as a first-class quality attribute.

3.2. ML Metrics

There is a distinct lack of studies addressing the quantitative architectural evaluation of ML and MLOps systems. However, authors have explored using runtime metrics to assess crucial aspects of ML and MLOps systems or ML models.

Cardoso Silva et al. [38] emphasise the significance of factors such as task completion time, CPU and GPU usage, memory usage, disk input/output, and network traffic in MLOps. They propose an ML benchmarking approach to monitor solutions in production, aiding decision-making, resource provision, and operation of ML systems.

Y. Zhou et al. [30] created a functional DevOps-capable ML platform. They built and executed ML pipelines with diverse layers and hyperparameters, collecting and evaluating metrics for ML pipeline steps, the ML platform, and ML models. The study identified potential performance bottlenecks, such as GPU utilisation, offering a valuable practical reference for constructing ML pipeline platforms.

J. Zhou et al. [39] conducted a survey offering an overview of methods proposed in the literature to assess ML model explanations. They identify properties of explainability as targets of quantitative metrics and discuss qualitative, human-centric metrics.

A literature review conducted by Carvalho et al. [40] identifies established methods and metrics for interpreting ML models. It highlights valuable work in interpretability assessments and recommends future research areas.

The acknowledged works are relevant to our study as they assess various quality aspects of ML systems and ML models. However, they focus on runtime performance metrics rather than system architecture metrics and do not specifically evaluate ML system support for automation-related ADDs. Whilst runtime metrics are crucial for understanding the operational efficiency and performance of ML systems during execution, they do not provide insights into the underlying patterns and practices that influence automation capabilities. Our approach involves a model-driven, metrics-based strategy to assess automation support in MLOps architectures by incorporating architectural quality metrics into our analysis.

To our knowledge, no prior studies consider architectural quality metrics on modelled MLOps systems, marking the initiation of a new research direction in this field. By leveraging architectural modelling techniques and novel quality metrics, we can systematically evaluate the readiness of MLOps architectures for automation and identify opportunities for improvement.

4. Research Method

In this section, we outline our overall research process, detail our MLOps system search and selection methodology, and give an overview of our ADD ground truth assessments, system modelling method and detectors. Figure 2 provides an overview of our research process. All artefacts are available in our replication package [41].

4.1. Contributions from Prior Work

Our study makes use of results and contributions from previous work [20, 21], where we performed a *Grey Literature Search* and *Grounded Theory Analysis* [42–44] through iterative consideration of practitioner literature. This process enabled us to *Formulate Core ADDs* for ML, the ones of interest for our study being automation-related (see Section 5).

As outlined in Section 1, our previous findings were supported by subsequent scientific literature. Our methodological choices were influenced by the nature of the literature we analysed. To obtain industry-relevant insights, we employed a deductive approach based on Straussian Grounded Theory, which is well-suited for an unbiased exploration of ML-related phenomena grounded in first-hand practitioner experience, especially since we found no prior ML-related Grounded Theory studies. For the white literature (peer-reviewed articles), we adopted an inductive approach to ensure our initial findings aligned with those of other researchers. The combination of these approaches provided us with a confident and nuanced understanding of ML-related practices, particularly in the context of MLOps.

We also devised an *Initial MLOps System Modelling* method [45], which we make use of in this study. Replication packages containing relevant artefacts from our prior studies are available in long-term repositories [46–48].

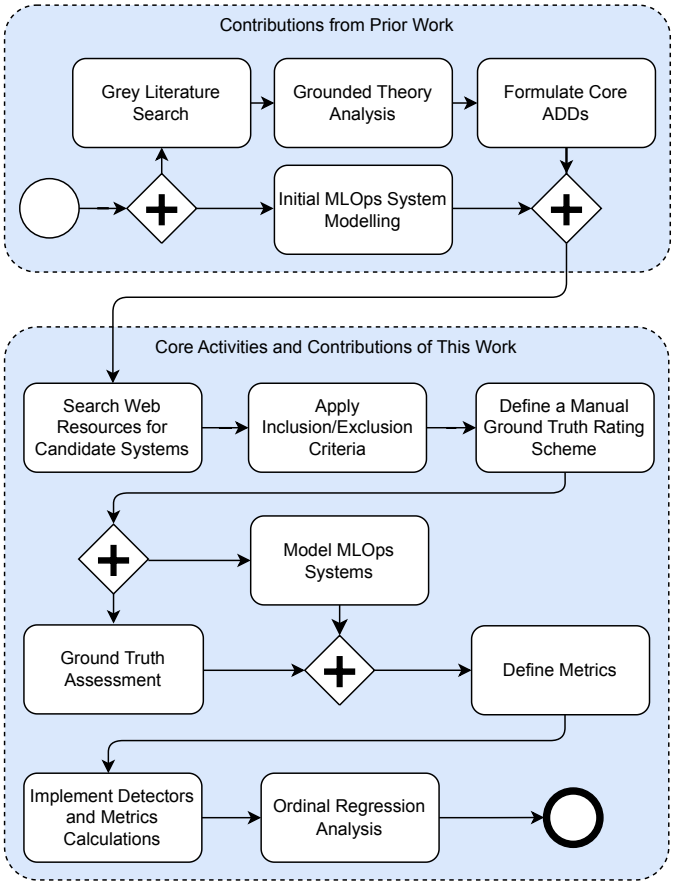

Figure 2: Overview of the research process we followed in this study.

4.2. Core Activities and Contributions of this Work

Search Web Resources for Candidate Systems
Our first step involved using popular topic portals (InfoQ, DZone), search engines (Google, DuckDuckGo, Bing) and source code repositories on GitHub to source relevant MLOps systems and descriptions thereof. Our search terms included "MLOps implementation", "MLOps system architecture" and "MLOps system example".

Apply Inclusion/Exclusion Criteria
The next step involved deciding which MLOps architectures resulting from our search to include in our study based on predefined inclusion-exclusion criteria. If a resource met one of the inclusion criteria, then we checked the exclusion criteria, whereas if the resource met none of the inclusion criteria, then we immediately excluded it. If an exclusion criterion was immediately apparent, then we checked neither the inclusion criteria nor any further exclusion criteria. Out of the thirty-two candidates we identified, only nine met our suitability criteria for inclusion in the study. For reproducibility and traceability, this stage of the research process is included in our replication package [41].

We defined our **inclusion criteria** as follows:

- MLOps systems: we considered systems that provide end-to-end solutions for the training, deployment and management of ML models in production environments.
- Architectural descriptions: we included sources that provide detailed descriptions of the architecture and design of MLOps systems.
- Reference architectures: we included reference architectures that serve as blueprints or guidelines for building MLOps systems.
- Industrial architecture guides: we considered guides and best practices for architecting MLOps solutions in an industrial setting.

We established the following **exclusion criteria**:

- Student and pet projects: we omitted projects made by students for educational purposes or personal experiments. We categorised pet projects as GitHub repositories with fewer than 100 stars and 40 forks, suggesting low acceptance and maturity.
- Experimental or proof of concept examples: we excluded sources that discuss or implement technologies unsuitable for real-world industrial use. These include systems that lack production-ready functionality, scalability, or dependability.
- Tools and libraries: we omitted sources that only discuss or implement specific tools or libraries used in the ML workflow, such as data preprocessing libraries or model training frameworks since these sources do not offer a thorough overview of MLOps architectures as a whole.
- Specific parts of the ML workflow: we excluded sources that do not address the entire ML workflow, such as those focusing only on data preparation or model deployment, since they do not take into account the complete MLOps architecture.
- Advertisements: we omitted sources that primarily promote commercial items or services since they may not give objective architectural information.
- Lack of practical architectural examples: we excluded sources that did not provide realistic examples of MLOps architectures applicable to an industrial setting.
- Marketing: we omitted sources that discuss consultancy organisations' techniques since they may not be reflective of general real-world MLOps designs.

Define a Manual Ground Truth Rating Scheme and Ground Truth Assessment
In the subsequent steps, we established a ground truth rating scheme for MLOps systems to understand their support for automation-related ADDs. We performed a manual ground truth assessment as described in Section 6.

Model MLOps Systems

We systematically modelled MLOps system architectures and their variations, ensuring comprehensive coverage of the design space. We utilised an MLOps system metamodel and followed a systematic modelling approach [45] to construct formal architectural models. These models, implemented in Python using Codeable Models[1] and visualised as UML diagrams using PlantUML[2], comprise nodes and connectors representing components and relationships.

The modelling process resulted in twenty-two models based on real-world MLOps systems from vendors providing computing services for ML application developers, including Alibaba Cloud, Amazon AWS, Google, Microsoft, and Red Hat. We assigned each modelled system an ID: base systems, whose IDs always end with a "1", include "AP1", "MA1" and "AC1", and variants of the base systems, whose IDs always end with a number other than "1" such as "AP2", "MA2" and "AC2". Table 1 lists the modelled systems, the size of each model in terms of how many nodes, connectors and pipelines (see our replication package [41] for the source code and model diagrams), a description of each system, and the URL of the source of each base system.

MLOps system model views commonly include component and pipeline views. Component views depict nodes representing various elements such as execution environments, cloud components, platforms, and connectors indicating their relationships, specifying details like artefact (e.g. ML model) provision, triggers (typically pipeline triggers), trigger data, i.e. the data associated with a trigger event, such as information on the trigger event itself or data relevant for the component or pipeline that was triggered, deployment targets (e.g. the components or environments where models will be deployed and executed to make decisions or predictions), and pipeline interactions and relations. Figure 3 depicts a component view of the modelled MLOps system AP1 (see Table 1).

Pipeline views display internal pipeline details, including an initial node, pipeline nodes for execution order, and a final node. Fork nodes allow parallel execution, converging at a join node. They convey various aspects like triggers (described above), data processing (the stage where raw data is transformed into a format that is suited for ML, like data sanitisation, feature engineering and normalisation), model training (the process by which a model learns from data), testing and evaluation (using a test dataset to assess how well a model generalises to unseen data), validation (tuning the hyperparameters of a model with a validation dataset and preventing overfitting) and deployment (described above), with metadata describing automation (e.g. whether a pipeline step runs automatically) and deployment or runtime environments (such as the specific component associated with the environment). Figure 4 illustrates a pipeline UML view of system AP1.

Define Metrics
We defined objective, technology-agnostic metrics to measure conformance to previously identified automation-relevant decision options for ADDs. Our custom metrics are described in detail in Section 7.

Implement Detectors and Metrics Calculations
Modelling MLOps system architectures enabled us to create source code detectors that analyse components, pipelines, and their relationships, automatically calculating custom metrics and supporting ordinal regression analysis. These detectors enable the detection of metrics-relevant architectural properties, offering a reusable approach for new systems and metrics. While the detectors can be adapted for a checklist approach, this alternative is less effective and requires more manual effort during a system analysis than our metrics.

Ordinal Regression Analysis
Finally, we applied ordinal regression analysis, a common statistical approach, to assess how well the metrics predicted the previously established ground truth for the selected design decisions and modelled architectures. Our analysis and evaluation are described further in Section 8.

5. Architectural Design Decisions

This section introduces three automation-related ADDs and their associated decision options. We selected them for the significance of their influence on automation in MLOps. These ADDs encompass various decision drivers (forces), relations and practices found in the grey literature. The following discussion on the relevance of these decision options on automation supports our rationale for the manual ground truth assessment detailed in Section 6.

5.1. How to Automate Integration and Delivery in a Machine Learning Context? ("Integration and Delivery")

When deciding *How to Automate Integration and Delivery in a Machine Learning Context*, choosing *No Integration or Delivery Automation* is the most straightforward option. However, numerous experts strongly discourage this approach, emphasising its adverse effects on various factors, notably *process and work automation*.

An alternative option for continuously automating the deployment process involves utilising a *CI/CD Pipeline*. Our sources strongly endorse this option, especially for *process and work automation* and for facilitating *automated provisioning*, which pertains to the actual automation of the deployment step. In Figure 3, for instance, relevant components are **AWS Code Build** and **AWS Code Pipeline**, and in Figure 4, the pipeline nodes from **Train** onwards are relevant.

Another approach involves executing automated deployments through *Build and Deployment Scripts*. Whilst this choice contributes to *process and work automation*, it might involve manual intervention instead of achieving complete automation.

The *Machine Learning Orchestrator* represents another viable decision option. Employing an orchestrator enhances *process and work automation*, reducing the reliance on manual ex-

[1] https://github.com/uzdun/CodeableModels
[2] https://plantuml.com

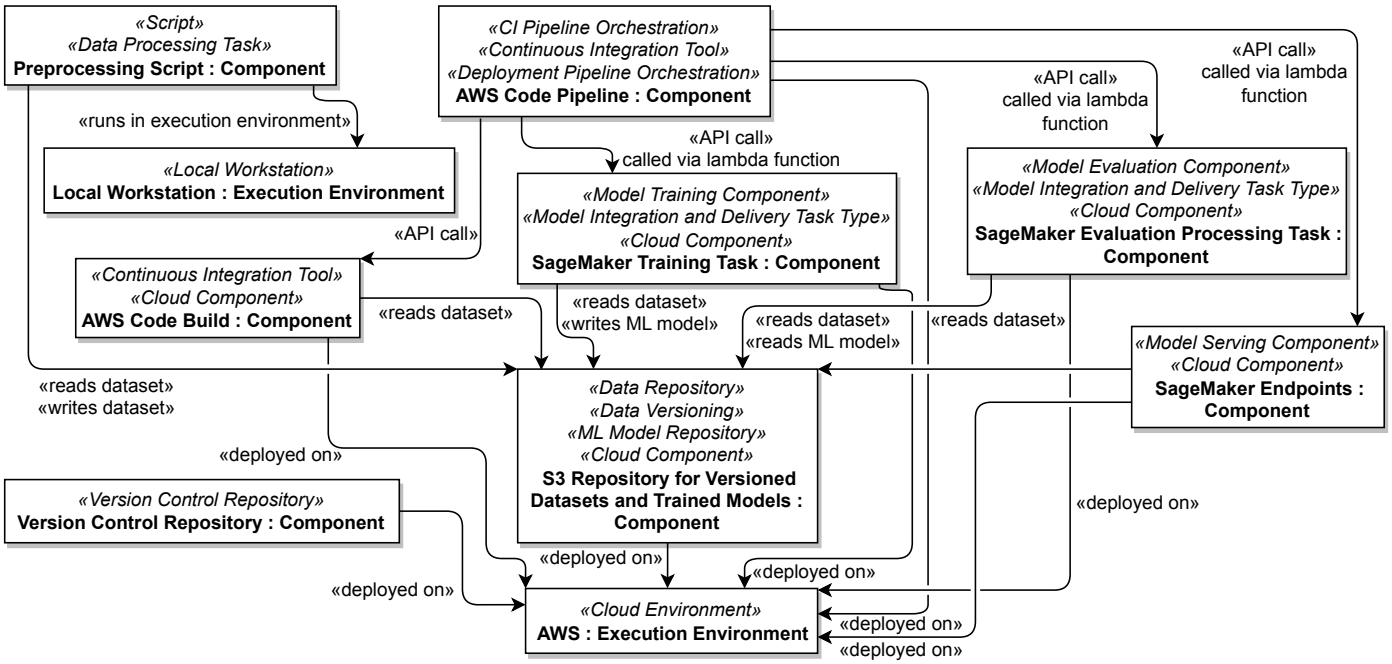

Figure 3: Component UML model diagram for the MLOps system AP1.

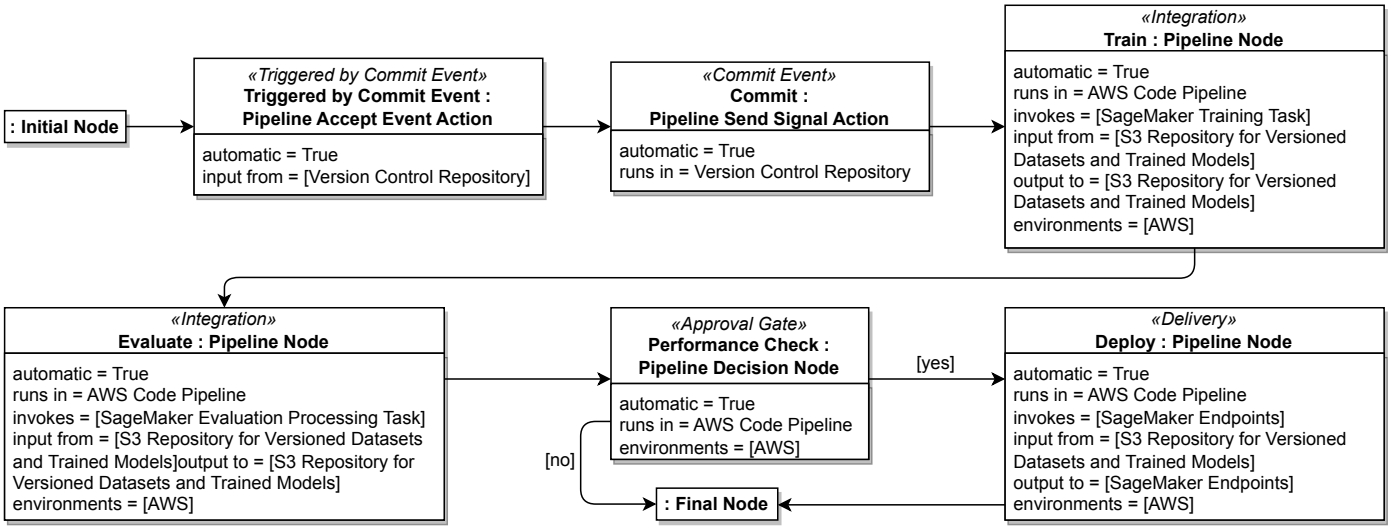

Figure 4: Deployment pipeline UML model diagram for the MLOps system AP1.

ecution and minimising the potential for human error, thereby allowing engineers to focus on other tasks.

5.2. How to Trigger a Machine Learning Pipeline or Orchestrator? ("Pipeline Triggers")

ML pipelines and orchestrators can be initialised via triggers. Triggers exhibit diverse characteristics, and the decision options' forces for this ADD are inherited from any higher-level decision option, e.g. whether a *CI/CD Pipeline*, *Build and Deployment Scripts* or *Machine Learning Orchestrator* was used. The trigger options themselves imply various levels of support for automation.

The trivial case is not to use an automated trigger, that is, to use an *On-Demand Trigger* (i.e. manual execution, such as by a human operator), which suggests no automation. This lack of automation allows the omittance of this trigger in the study. An *On Commit Trigger* activates components following a commit event in a source control management or data version repository system, thus offering some degree of automation. For system AP1, for instance, this is visible in the **Triggered by Commit Event** pipeline accept event action in Figure 4. An *On Schedule Trigger* initiates a component at specific times or with defined regular intervals. An *On Availability of New Training Data Trigger* can lead to the triggering of a *CI/CD Pipeline*, *Build and Deployment Scripts* or *Machine Learning Orchestra-*

Table 1: Included MLOps System Architectures: IDs, Model Sizes, Descriptions and Sources

System	Model Size	System Description
AP1	18 nodes, 25 connectors, 1 pipeline	Guide to building an automated MLOps pipeline using Amazon AWS CodePipeline, CodeBuild, SageMaker, Lambda, S3, the AWS container registry and Docker (source: https://tinyurl.com/mlops-system-ap).
AP2	21 nodes, 25 connectors, 2 pipelines	Variant of AP1 with partial support for automated *Build and Deployment Scripts*, advanced triggers but no basic triggers, and support for data processing via a *Data Pipeline*.
AP3	19 nodes, 22 connectors, 2 pipelines	Variant of AP1 with support for *Build and Deployment Scripts*, a single basic trigger and no data processing automation.
MA1	55 nodes, 58 connectors, 3 pipelines	Reference architecture for a Microsoft Azure project that demonstrates how to automate an end-to-end ML/AI workflow (source: https://tinyurl.com/mlops-system-ma).
MA2	53 nodes, 56 connectors, 3 pipelines	Variant of MA1 with partial support for integration and delivery automation via *CI/CD Pipelines*, no basic trigger support but support for an advanced trigger, and partial data processing automation via an *ETL Pipeline*.
MA3	24 nodes, 24 connectors, 1 pipeline	Variant of MA1 with no integration and delivery automation, support for a single advanced trigger but no basic triggers, and data processing automation support via a *Data Pipeline*.
AC1	33 nodes, 67 connectors, 1 pipeline	Reference architecture and workshop/tutorial to implement an Amazon AWS MLOps pipeline that automates the typical procedures used by ML practitioners (source: https://tinyurl.com/mlops-system-ac).
AC2	31 nodes, 63 connectors, 1 pipeline	Variant of AC1 with support for *Build and Deployment Scripts*, a single advanced trigger but no basic trigger and partial support for data processing automation via a *Data Processing Component*.
AC3	36 nodes, 71 connectors, 1 pipeline	Variant of AC1 with support for a *Machine Learning Orchestrator*, full trigger support and partial support for data processing automation via an *ETL Pipeline*.
GV1	78 nodes, 137 connectors, 3 pipelines	Google Cloud Platform implementation of MLOps with Vertex AI, Smart Analytics, Keras, TFX and Model Builder SDK (source: https://tinyurl.com/mlops-system-gv).
GV2	54 nodes, 96 connectors, 1 pipeline	Variant of GV1 with no integration and delivery automation, no automation via triggers, but support for data processing automation via a *Data Pipeline*.
GV3	70 nodes, 126 connectors, 3 pipelines	Variant of GV1 with partial *Machine Learning Orchestrator* support, support for a single basic trigger and partial support for data processing automation via a *Data Processing Component*.
AS1	35 nodes, 55 connectors, 1 pipeline	Alibaba Cloud repository and article focusing on the ML pipeline with MLOps using Serverless Workflow, Function Compute, and Container Service for Kubernetes provided by GitHub (source: https://tinyurl.com/mlops-system-as).
AS2	36 nodes, 57 connectors, 1 pipeline	Variant of AS1 with partial support for an *ML Orchestrator*, support for almost all triggers, and support for data processing automation via a *Data Processing Component*.
AS3	34 nodes, 56 connectors, 1 pipeline	Variant of AS1 with partial support for all three integration and delivery automation options, full trigger support and support for data processing automation via a *Data Processing Component*.
RO1	34 nodes, 61 connectors, 1 pipeline	MLOps architecture description using Red Hat OpenShift, applying DevOps and GitOps principles (source: https://tinyurl.com/mlops-system-ro).
RO2	29 nodes, 53 connectors, 1 pipeline	Variant of RO1 with partial *CI/CD Pipeline* support, support for a single advanced trigger but no basic triggers and no data processing automation support.
RO3	19 nodes, 32 connectors, 0 pipelines	Variant of RO1 with absolutely no automation support.
GM1	18 nodes, 16 connectors, 1 pipeline	Google Cloud Architecture Center description of an MLOps architecture at the basic level of maturity, where building and deploying ML models is entirely manual (source: https://tinyurl.com/mlops-system-gm).
GA1	55 nodes, 69 connectors, 2 pipelines	Describes an architecture with continuous model training via an automated ML pipeline, automated retraining on new data, automated data and model validation, pipeline triggers and metadata management to achieve continuous delivery of a model prediction service (source: https://tinyurl.com/mlops-system-ga).
GC1	64 nodes, 77 connectors, 3 pipelines	Architecture for a robust, automated CI/CD system for building, testing, and deploying new pipeline components to a target environment (source: https://tinyurl.com/mlops-system-gc).
GC2	64 nodes, 77 connectors, 3 pipelines	Variant of GC1 with similar support for integration and delivery automation and automation via triggers, but slightly worse support for data processing automation via *Data Pipelines*.

tor when new training data becomes available. An *On Model Performance Degradation Trigger* can be used when ML model performance deteriorates, and an *On Changes to the Data Distribution Trigger* can be applied when statistical changes in any dataset used for training ML models surpass a predefined threshold.

5.3. How to Automatically Process the Data Used for Model Building? ("Data Processing")

A fundamental automation-related ADD pertains to data processing automation for ML model building. The trivial option is to opt for *No Data Processing Automation*. Three alternative decision options to the trivial case offer vital advantages to

varying degrees, such as facilitating *automated data collection* and the potential for a high level of *process and work automation*, and are described below.

Some practitioners advocate for an ETL pipeline, where "ETL" refers to Extract, Transform, and Load. It entails extracting data from diverse sources, transforming it for analysis or storage, and loading it into a database, data warehouse, or data lake. The main aim is to ready the data for downstream analytics or applications, often integrating and consolidating data from various sources to ensure quality and consistency. ETL pipelines are typically batch-oriented and scheduled periodically for data upkeep. However, this approach is documented to be only occasionally employed in practice [20].

A prevalent choice among the automation architectures commonly suggested in the previously cited work is using a *Data Pipeline*. This approach involves conducting data processing within a dedicated, adaptable pipeline. Unlike ETL pipelines, data pipelines encompass a wider range of operations beyond just extraction, transformation, and loading. They can include tasks such as data ingestion, data preprocessing, feature engineering, ML model training, evaluation, and deployment. This decision option stands out as the most versatile among the automated choices.

Whilst pipeline architectures are frequently proposed, there are also suggestions for *Data Processing Components* that do not adhere to a pipeline structure. Unlike an ETL pipeline, a data processing component may not necessarily involve the extraction and loading of data and, in contrast to data pipelines, which encompass the entire flow of data, *Data Processing Components* instead focus on performing specific data processing tasks such as data transformation, feature engineering, data augmentation, scaling, and other preprocessing steps required for ML model training and analysis.

6. Ground Truth Rating Scheme and Assessment

This section offers insight into establishing the ground truth data for each ADD outlined in Section 5.

We established the ground truth by a thorough systematic manual assessment of the selected systems based on the findings from grey literature in prior studies, as well as multiple rounds of independent review and validation by all three members of the author team. We discussed any instances of disagreement in the ground truth assessment and refined it until we reached a consensus. Thanks to our use of a high-level abstraction of MLOps systems according to an existing, extensible MLOps metamodel during the modelling process (see Section 4.2), it was straightforward for us to identify the elements relevant to the ground truth assessment collectively. We corrected any inconsistencies both in the metamodel and in our ground truth assessment.

We categorised the relevant decision options as either *supported*, *partially supported*, or *not supported* (denoted by **S**, **P**, or **N**, respectively in Table 2), or, for decision options that are simply present or absent, with Boolean values.

By combining the outcome of all decision options for each ADD and system and informed by the descriptions from the literature of the force impacts of the various decision options in Section 5, we then derived an ordinal assessment of how well the decision as a whole was supported in each modelled system, according to the following ordinal scale:

- **++**: Automation is very well supported.
- **+**: Automation is well supported, but aspects of the system could be improved to better support automation.
- **o**: Aspects of the system need to be improved concerning automation, but substantial support for automation is present.
- **-**: Aspects of the system need to be improved concerning automation, but some support for automation is present.
- **--**: No support for automation is present.

Whilst our ordinal scale of **--** to **++** adheres to the given order, it does not assume equal intervals, unlike a Likert scale. The specific assessment criteria for each ADD are described in Sections 6.1, 6.2 and 6.3, and we recorded the resulting ordinal assessments in the "Assessments" rows of Table 2.

Consider system AP1 in Table 2, for instance. For the *Integration and Delivery* ADD, AP1 does not support the decision option *Build and Deployment Scripts* (category **N**), it supports the decision option *CI/CD Pipeline* (category **S**), and it does not support the decision option *Machine Learning Orchestrator* (category **N**). With reference to the scoring scheme defined in Section 6.1, this system receives a score of **++** for this ADD since the decision option *CI/CD Pipeline* is supported and *all integration and delivery tasks are automated and take place within a CI/CD Pipeline*. This score also corresponds to the generic scoring scheme defined above, where **++** denotes that *automation is very well supported.*

Following the establishment and assessment of the ground truth data, we assessed how well the novel metrics defined in Section 7 predicted the ground truth data by performing an ordinal regression analysis. We describe the ordinal regression analysis in Section 8.

6.1. Integration and Delivery

In Section 5, we discuss the level of support for automation for each *Integration and Delivery* option, and on that basis, we devised the following scoring framework for our ground truth assessment of this decision.

- **++**: **All** integration and delivery tasks are automated and take place within a *CI/CD Pipeline*.
- **+**: The **majority** of (but not all) integration and delivery tasks are automated and take place within a *CI/CD Pipeline* OR **all** integration and delivery tasks are automated and take place via a *Machine Learning Orchestrator*.
- **o**: The **majority** of (but not all) integration and delivery tasks are automated and take place via a *Machine Learning Orchestrator* OR the **majority or all** integration and

delivery tasks are automated and take place within *Build and Deployment Scripts*.

- **-**: **Some** (but not the majority or all) integration and delivery tasks are automated and take place within a *CI/CD Pipeline*, via a *Machine Learning Orchestrator* or within *Build and Deployment Scripts*.
- **--**: **No** integration and delivery task automation is supported.

6.2. Pipeline Triggers

Drawing upon the reasoning provided in Section 5 in support of the *Pipeline Triggers* decision, we established the ensuing scoring scheme for our ground truth assessment:

- **++**: **At least one** basic trigger (*On Commit Trigger* or *On Schedule Trigger*) is supported, and **all** applicable advanced automatic triggers are supported (*On Availability of New Training Data Trigger*, *On Model Performance Degradation Trigger*, *On Changes to the Data Distribution Trigger*).
- **+**: **At least one** basic trigger (*On Commit Trigger* or *On Schedule Trigger*) is supported, and **at least one** advanced automatic trigger is supported (*On Availability of New Training Data Trigger*, *On Model Performance Degradation Trigger*, *On Changes to the Data Distribution Trigger*).
- **o**: **At least one** basic trigger (*On Commit Trigger* or *On Schedule Trigger*) is supported, but **no** advanced triggers are supported.
- **-**: **At least one** advanced trigger is supported (*On Availability of New Training Data Trigger*, *On Model Performance Degradation Trigger*, *On Changes to the Data Distribution Trigger*), but **no** basic triggers are supported (*On Commit Trigger* or *On Schedule Trigger*).
- **--**: **No** automatic triggers are supported.

6.3. Data Processing

Lastly, based on the forces we described in Section 5 for the *Data Processing* decision, we can establish the subsequent scoring scheme for our ground truth assessment:

- **++**: **All** model-building-relevant data processing tasks are automated and occur within a *Data Pipeline*.
- **+**: **The majority** of model-building-relevant data processing tasks are automated and occur within a *Data Pipeline* OR **all** model-building-relevant data processing tasks are automated and occur within an *ETL Pipeline* or *Data Processing Component*.
- **o**: **The majority** of model-building-relevant data processing tasks are automated and occur within an *ETL Pipeline* or *Data Processing Component*.
- **-**: **Some** model-building-relevant data processing tasks are automated and occur within a *Data Pipeline*, an *ETL Pipeline* or a *Data Processing Component*.
- **--**: **No** data processing automation is supported.

7. Metrics

This section presents the metrics we have proposed for each decision option discussed in Section 5. Following our iterative manual assessment and modelling of the MLOps system architectures from Table 1 to establish the ground truths ratings described in Section 6 and summarised in Table 2, we defined at least one metric per ADD decision option to measure their support for automation. In Section 8, we describe how we performed ordinal regression analysis to assess how well the custom metrics defined below in this section predicted the ground truth data.

The metrics have been deliberately designed to be straightforward, allowing them to represent each decision effectively. They may be either a Boolean value indicating the absence or presence of a given decision option within the respective system or a continuous value in [0, 1], with 0 signifying the worst-case scenario where there is no support for the decision option, and 1 symbolising an ideal scenario in which the decision option receives full support.

7.1. Metrics for the Integration and Delivery Decision

The ratio of components that support the *Build and Deployment Scripts* option is calculated by the **BDS** metric.

$$BDS = \frac{|\textit{integration and delivery tasks in build and deployment scripts}|}{|\textit{integration and delivery tasks in the system}|}$$

The proportion of components enabling the *CI/CD Pipeline* option is revealed by the **CID** metric.

$$CID = \frac{|\textit{integration and delivery tasks automated in a CI/CD pipeline}|}{|\textit{integration and delivery tasks in the system}|}$$

The fraction of components that back the *Machine Learning Orchestrator* option is given by the **MLO** metric.

$$MLO = \frac{|\textit{integration and delivery tasks automated in an ML orchestrator}|}{|\textit{integration and delivery tasks in the system}|}$$

7.2. Metrics for the Pipeline Triggers Decision

We devised the following *Pipeline Triggers* metrics:

- The existence of the *On Commit Trigger* option is signified by the **COT** metric, which yields a Boolean value.
- Presence of the *On Schedule Trigger* option is conveyed by the Boolean **SCT** metric.

Table 2: Ground Truth Data for ADDs, Decision Options and the Included MLOps System Architectures

ADDs/Decision Options	AP1	AP2	AP3	MA1	MA2	MA3	AC1	AC2	AC3	GV1	GV2	GV3	AS1	AS2	AS3	RO1	RO2	RO3	GM1	GA1	GC1	GC2
Integration and Delivery																						
Build and Deployment Scripts	N	P	S	N	N	N	N	S	N	N	N	N	N	N	P	N	N	N	P	N	N	N
CI/CD Pipeline	S	N	N	P	P	N	S	N	N	S	N	N	N	N	P	P	P	N	N	S	S	S
Machine Learning Orchestrator	N	N	N	P	N	N	N	N	S	N	N	P	S	P	P	N	N	N	N	P	P	P
Assessments	++	-	o	+	-	--	++	o	+	++	--	+	+	-	o	+	-	--	--	++	++	++
Pipeline Triggers																						
On Commit	T	F	F	T	F	F	T	F	T	T	F	F	T	T	T	T	F	F	F	F	T	T
On Schedule	F	F	F	T	F	F	F	F	T	T	F	T	F	T	T	F	F	F	F	T	T	T
Availability of New Training Data	F	T	F	T	F	F	T	T	T	F	F	F	F	T	T	F	F	F	F	T	T	T
Model Performance Degradation	F	T	F	T	F	F	F	F	T	F	F	F	F	F	T	T	T	F	F	T	T	T
Changes to the Data Distribution	F	T	F	F	T	T	F	F	T	F	F	F	F	T	T	F	F	F	F	T	T	T
Assessments	o	-	--	+	-	-	+	-	++	o	--	o	o	+	++	+	-	--	--	++	++	++
Data Processing																						
Data Pipeline	N	S	N	N	N	S	N	N	N	N	S	N	N	N	N	N	N	N	N	P	P	P
ETL Pipeline	N	N	N	N	P	N	S	N	P	N	N	N	N	N	N	N	N	N	N	N	N	N
Data Processing Component	N	N	N	N	N	N	N	P	N	P	N	P	S	S	S	N	N	N	N	N	N	N
Assessments	--	++	--	--	-	++	+	o	-	o	++	o	+	+	+	--	--	--	--	+	+	-

- Whether the *On Availability of New Training Data Trigger* option is represented in the system is signified by the **TDT** metric, also a Boolean.

- An indicator for the presence of the *On Model Performance Degradation Trigger* option is the Boolean **MPT** metric.

- The availability of the *On Changes to the Data Distribution Trigger* option is informed by the Boolean **DDT** metric.

The following equation thus gives the value for any of the above metrics:

$$COT, SCT, TDT, MPT, DDT = \begin{cases} True: & \text{if } \textit{the model contains} \\ & \textit{at least one link of} \\ & \textit{the relevant trigger type;} \\ False: & \textit{otherwise.} \end{cases}$$

7.3. Metrics for the Data Processing Decision

The proportion of components in favour of the *Data Pipeline* option is indicated by the **DAP** metric.

$$DAP = \frac{|\textit{data processing tasks automated in a data pipeline}|}{|\textit{data processing tasks in the system}|}$$

The share of components supporting the *ETL Pipeline* option is revealed by the **ETP** metric.

$$ETP = \frac{|\textit{data processing tasks automated in an ETL pipeline}|}{|\textit{data processing tasks in the system}|}$$

The ratio of components that are aligned with the *Data Processing Component* option is shown by the **DPC** metric.

$$DPC = \frac{|\textit{data processing tasks in a data processing component}|}{|\textit{data processing tasks in the system}|}$$

When applying the methodology described in this paper, a human architect familiar with the system architecture would be involved in classifying system features, interpreting metrics, and setting thresholds. Generally, a lower value is worse than a higher value, depending on how desirable supporting a decision option is for an architect. In this context, please note that the system's modularity does not directly influence the value of the metrics; instead, it is the nature of the containing component that matters.

8. Ordinal Regression Analysis

Ordinal regression models the relationship between an ordinal dependent variable and independent predictors. It helps understand the odds of an observation belonging to a specific category, assess the strength and direction of relationships, evaluate predictor significance, and build accurate regression models for predictions with new data.

The results of the ordinal regression analysis[3] concerning each ADD are showcased in Table 4. The outcome variables, which depend on the ground truth assessments (ordinal variables) for each decision as elucidated in Section 6 and shown in Table 2, are integral to the analysis. Furthermore, the metrics expounded upon in Section 7 and summarised in Table 3 serve as the independent predictor variables and are either Boolean or continuous values measured within the interval [0, 1].

[3] When conducting the ordinal regression analysis, we utilised the *lrm* function from the *rms* package in R [49].

Table 3: Metrics Calculation Results for the Included MLOps Architectures

Metrics	AP1	AP2	AP3	MA1	MA2	MA3	AC1	AC2	AC3	GV1	GV2	GV3	AS1	AS2	AS3	RO1	RO2	RO3	GM1	GA1	GC1	GC2
Integration and Delivery																						
BDS	0	0.333	1	0	0	0	0	1	0	0	0	0	0	0	0.667	0	0	0	0	0	0	0
CID	1	0	0	0.615	0.462	0	1	0	0	1	0	0	0	0	0	0.875	0.375	0	0	1	1	1
MLO	0	0	0	0.385	0	0	0	0	1	0	0	0.857	1	0.333	0	0	0	0	0	0.429	0.375	0.375
Pipeline Triggers																						
COT	T	F	F	T	F	F	T	F	T	T	F	F	T	T	T	T	F	F	F	F	T	T
SCT	F	F	F	T	F	F	F	F	T	T	F	T	F	T	T	F	F	F	F	T	T	T
TDT	F	T	F	T	F	F	T	T	T	F	F	F	F	T	T	F	F	F	F	T	T	T
MPT	F	T	F	T	F	F	F	F	T	F	F	F	F	F	T	T	T	F	F	T	T	T
DDT	F	T	F	F	T	T	F	F	T	F	F	F	F	T	T	F	F	F	F	T	T	T
Data Processing																						
DAP	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0.5	0.5	0.333
ETP	0	0	0	0	0.333	0	1	0	0.333	0	0	0	0	0	0	0	0	0	0	0	0	0
DPC	0	0	0	0	0	0	0	0.5	0	0.6	0	0.8	1	1	1	0	0	0	0	0	0	0

The ordinal response variable is divided into increasing intercept categories, where each of $\geq$ [-], $\geq$ [o], $\geq$ [+], $\geq$ [++] as introduced in Section 6 and expanded upon in Section 6 represents the transition point of the corresponding level (i.e. for the dependent variable). The intercept coefficients in our regression models quantify the impact of each independent predictor variable on the outcome, indicating how they influence the odds of transitioning between response categories. Positive coefficients suggest an increased likelihood of moving to a higher category with each unit increase in the independent variable, whilst negative coefficients imply the opposite, indicating a reduced likelihood. For example, a +5 coefficient corresponds to a five-fold increase in the dependent variable for every unit increase in the independent variable, and a -30 coefficient signifies a thirty-fold decrease.

The regression model C-index, also known as the concordance index or the area under the Receiver Operating Characteristic (ROC) curve, measures the discriminatory power [50] of a regression model. The original C-index assesses how well a regression model distinguishes between different categories or levels of the ordinal response variable. The bias-corrected C-index accounts for potential bias in a regression model. C-index values are continuous in the interval $[0, 1]$, with higher values indicating better discriminatory power. Harrell [49] recommends the utilisation of bootstrapping, a resampling technique, to obtain nearly unbiased estimates of a regression model's future performance[4]. A C-index of 1 represents perfect discrimination, whilst 0.5 indicates random chance. Regression models are considered good when the C-index exceeds 0.7 and strong when it surpasses 0.8. A C-index value of 1 signifies perfect predictive power. As illustrated in Table 3, all C-index values exceed 0.8, affirming our regression models' effectiveness in predicting individual outcomes.

The Brier score in ordinal regression assesses the quality of probabilistic predictions by measuring the mean squared difference between predicted probabilities and actual outcomes [51]. Unlike simple accuracy metrics, it offers a nuanced evaluation of calibration and accuracy. Lower scores signify better predictive accuracy, with 0 representing a perfect match. All three regression models demonstrate low Brier scores, indicating their capability for accurate predictions.

In ordinal regression, p-values for regression model coefficients indicate predictor significance, often below 0.05 for statistical significance [52]. In Table 4, all p-values are below 0.05, confirming the models' statistical significance and highlighting the substantial influence of included metrics on dependent variables.

Ordinal regression analysis is used for outcome variables with ordered categories that do not have equal distances between them. Using an overall metric based on average weight can violate the fundamental assumption of ordinal regression, as it implies equal intervals between categories, which may not be the case. Moreover, ordinal regression allows for modelling the probability of being in a particular category or lower based on a set of predictors. For this reason, we do not use averages of the various decision option metrics to achieve an overall score for a modelled system since this could oversimplify the complex relationships between predictors and outcome variables. Determining appropriate weights for the various components of composite metrics is challenging, and relying on an average requires an accurate reflection of each component's significance. This issue is similar to those encountered with other metrics, such as the Maintainability Index [53]. Analysing each ADD separately and applying ordinal regression analysis provides a more nuanced understanding of the data than taking averages (see Table 4) and results in reusable ordinal regression models, so it is highly suited to this study.

Our multi-metric approach ensures a robust assessment of regression model performance. All three models emerge as strong, statistically significant predictors of their respective ordinal response variables, collectively providing a comprehensive understanding of automation-related ADD support. Each model contributes valuable insights, collectively offering a well-rounded perspective.

[4]We employed the *validate* function from the *rms* package to achieve bootstrapping and the subsequent calculation of the bias-corrected C-index.

Table 4: Ordinal Regression Analysis Results for the Metrics

Statistical Measures	Values
Integration and Delivery	
Intercept: ≥ [-]	-0.4118
Intercept: ≥ [o]	-2.2673
Intercept: ≥ [+]	-3.5752
Intercept: ≥ [++]	-5.7906
Metric Coefficient (BDS)	2.9438
Metric Coefficient (CID)	5.8622
Metric Coefficient (MLO)	4.4388
Regression Model p-value	8.2844e-09
Regression Model Brier Score	0.0637
Regression Model C-index (original)	0.9895
Regression Model C-index (bias-corrected)	0.9192
Pipeline Triggers	
Intercept: ≥ [-]	-0.1809
Intercept: ≥ [o]	-2.2715
Intercept: ≥ [+]	-4.0512
Intercept: ≥ [++]	-6.4206
Metric Coefficient (COT)	2.4227
Metric Coefficient (SCT)	1.7147
Metric Coefficient (TDT)	1.1081
Metric Coefficient (MPT)	1.4322
Metric Coefficient (DDT)	1.0823
Regression Model p-value	1.6090e-07
Regression Model Brier Score	0.0842
Regression Model C-index (original)	0.9585
Regression Model C-index (bias-corrected)	0.9363
Data Processing	
Intercept: ≥ [-]	-0.8649
Intercept: ≥ [o]	-2.1320
Intercept: ≥ [+]	-3.3447
Intercept: ≥ [++]	-6.4474
Metric Coefficient (DAP)	7.4840
Metric Coefficient (ETP)	4.2296
Metric Coefficient (DPC)	4.1724
Regression Model p-value	5.5376e-09
Regression Model Brier Score	0.05585
Regression Model C-index (original)	0.9946
Regression Model C-index (bias-corrected)	0.9595

9. Discussion

In this section, we address the research questions, discuss a practical application of our approach and examine potential threats to validity.

9.1. Discussion of the Research Questions

Our study addressed two key research questions (RQs) regarding the assessment of automation support in MLOps systems.

RQ1 *How can MLOps systems' support for the core quality aspect of automation be assessed in the context of Architectural Design Decisions and their respective decision options?*

In addressing RQ1, we conducted a rigorous evaluation of MLOps systems' support for automation-related ADDs and their respective decision options. Through an iterative modelling and evaluation process, we established a comprehensive ground truth, ensuring thorough coverage of the design space. Our technology-agnostic metrics, assigned to each decision option, and subsequent ordinal regression analysis demonstrated the effectiveness of our approach in objectively assessing the level of automation support in MLOps systems. This methodology ensures a thorough understanding of the nuances of automation-related ADDs and decision options within the context of MLOps. This iterative process can be valuable in other domains where intricate architectural decisions impact system quality aspects.

RQ2 *What types of metrics can be applied to evaluate these levels of support, and how effective are they?*

In response to RQ2, we introduced generic, technology-agnostic metrics closely tied to typical ADDs concerning automation in MLOps. We programmatically applied these metrics via our detectors, and the regression analysis statistically reinforced their effectiveness in evaluating automation support. Developing technology-agnostic metrics is advantageous as it allows for a broad application across diverse MLOps systems. The effectiveness of the metrics, as demonstrated by regression analysis, suggests a promising avenue for objectively assessing automation support.

RQ2 logically follows from RQ1 as it builds upon the foundational work established in addressing RQ1. In RQ1, we rigorously evaluated the support for automation in MLOps systems by manually assessing ADDs and their associated decision options. For RQ2, we introduced generic, technology-agnostic metrics specifically tailored to these ADDs, expanding upon the evaluation framework developed in RQ1 to provide a broader, more generalisable assessment of automation support across diverse MLOps systems, thereby demonstrating a clear relationship between the two research questions and their measurement methodologies.

Our study employs an iterative evaluation process and technology-agnostic metrics to describe a useful modelling and evaluation process for MLOps researchers and practitioners. Our approach facilitates the modelling of MLOps systems, objective assessment and evaluation of support for automation-related ADDs and the introduced metrics contribute to advancing the understanding of how to detect and assess quality aspects of MLOps systems, with the scope for wider applicability.

The objective of this study is to determine the extent to which various ADD options are supported. Suppose a metric's value is zero (or false, in the case of Boolean metrics). In that case, it indicates that the type of automation associated with this decision option is not feasible within the system. Conversely, a non-zero

value (or true) signifies the presence of the decision option, but it does not guarantee perfect or even correct implementation.

Our methodology and metrics allow us to assess the degree of coverage of automation-related decision options in terms of how well they are supported, enabling us to measure our intended goals and consider automation as a measurable system quality. An architect familiar with a given project would be aware of the relevant decision options based on the force impacts described in Section 5, which provide insights into the preferable decision options considering the trade-offs between them.

9.2. Practical Application

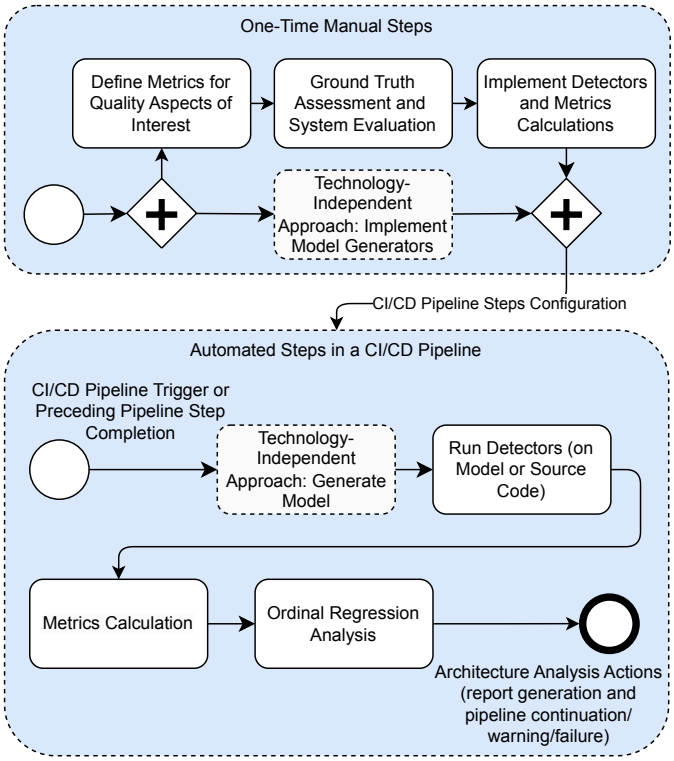

Figure 5: A proposed practical application of our solution.

We have demonstrated our approach's effectiveness in detecting and assessing complex quality aspects of MLOps systems, such as ADDs and pipeline and component properties, which are challenging to identify manually in source code — incorporating our approach as automated steps within a CI/CD pipeline, as depicted in Figure 5, would offer substantial practical benefits.

The manual effort in initial ground truth assessment and system evaluation need only be expended once. We demonstrated our approach for this paper by manually modelling a range of systems, requiring some initial manual effort due to the technology-agnostic approach but enabling subsequent automation of analysis thanks to the high-level abstraction provided by the models. In a real-world application of a specific system, practitioners can avoid the manual modelling effort by automating the most repetitive and labour-intensive steps for their specific system and its technologies within a CI/CD pipeline, either leveraging source code detectors for a specific set of technologies or automating the generation of system architecture model abstractions for analysis with model detectors if a more technology-independent approach is required.

The subsequent regression model analysis can also be automated within a CI/CD pipeline, a report of the outcome generated, and further actions within the pipeline taken such as continuing with the pipeline, failing or issuing a warning. The proposed application of this solution could be utilised as a form of regression testing within the quality assurance process to aid practitioners in automatically identifying when quality aspects of their system in an MLOps context fall below a predefined threshold. More generally, this automated process can yield early indications of how the system is evolving and whether refactoring may be necessary. Thus our reusable ADDs, detector-based analysis and regression models facilitate automation, allowing practitioners to invest effort a single time and reuse results repeatedly and automatically. Through automation of the analysis, organisations can gain valuable insights into the qualities of their MLOps systems whilst accelerating the development cycle and ensuring compliance with non-functional requirements, in particular with regard to quality aspects.

Further extensions could involve more detailed specific design guidance on measures that could be taken or automated optimisation of the system architecture with respect to quality attributes where necessary and appropriate and depending on the practitioner's needs. Indeed, the scope of application is not limited to automation-related ADDs or automation-related qualities of the system architecture: numerous quality aspects of MLOps systems, specifically those ADDs, qualities and forces that practitioners find most critical on a case-by-case basis, for instance, those that enhance model management and reusability, and generally reduce technical debt, could be supported.

9.3. Threats to Validity

Ensuring generalisability across diverse MLOps contexts may compromise the external validity of our findings. However, our confidence in the representativeness of our modelled systems remains strong, as they cover a comprehensive set of automation-related ADDs and systems documented in practitioner and scientific literature [20, 21, 45].

When modelling MLOps system architectures, there is a risk of unintentionally omitting architectural elements, affecting external validity. However, our experience in architectural modelling ensured diligent coverage of encountered phenomena, ultimately affirmed by our results.

We studied, modelled and evaluated third-party systems to enhance internal validity and minimise bias in system composition. Despite potential omissions in our search procedures, we mitigated this through a systematic approach, strict inclusion-exclusion criteria, and thorough searches. Furthermore, multiple experienced authors rigorously reviewed the sources.

To maintain internal validity amid potential subjectivity in ground truth assessment, we simplified the evaluation of decision option support. We used straightforward heuristics and a

simple three-step ordinal scale. Our goal was to reduce false positives, enable technology-independent evaluation, and ensure a non-controversial interpretation.

Although there is a potential risk of interpretive bias in the system modelling process, our experienced author team aimed to develop system models faithfully representing observed phenomena. However, this risk does not significantly impact our study, as our primary objective was ensuring system model accuracy rather than uniformity with models created by other researchers.

Another potential risk to internal validity involves the development of system variants by the authors. However, it is crucial to note that these variants align with documented ADDs and decision options found in the literature [20, 21], and we took care to modify only the necessary aspects for each variant whilst maintaining consistency in all other respects.

10. Future Work

Future work will involve applying our results to specific industrial systems, showcasing the applicability and practical benefits of our approach in real-world scenarios. We plan to extend our system modelling techniques to cover other types of ML systems, particularly those with less coverage in the scientific literature. Specifically, we will apply our approach to reinforcement learning operations (RLOps) systems, using industry case studies to evaluate support for automation and other core quality aspects through an automated metrics-based analysis using source code detectors and large language models as opposed to the manual modelling step as proposed in Section 9.2. The solution will involve identifying quality deficits and suggesting necessary extensions by assessing the current coverage of the design space (expressed as ADDs) for MLOps or RLOps systems. We also plan to incorporate our work into an automated process as suggested in Section 9.2 for continuously assessing MLOps or RLOps architectures in an industrial setting, where metrics are repeatedly evaluated to detect changes in system quality.

11. Conclusion

In this study, we focused on developing a method to assess the support for automation-related ADDs and decision options in MLOps systems, demonstrating the feasibility of this approach. We followed a systematic process involving sourcing, modelling, and assessing various MLOps systems and their variants.

We introduced novel metrics covering all potential decision options and implemented detectors to analyse the modelled systems and compute the metrics. We have demonstrated that simple yet effective metrics provide reliable insights into crucial system aspects, such as automation, which are paramount to practitioners. We employed statistical methods, specifically ordinal regression analysis, to establish prediction models, which demonstrated high accuracy in predicting the ground truth assessment. The validation of our novel metrics and metrics-based approach underscores our contribution.

CRediT Authorship Contribution Statement

The CRediT Authorship Contribution Statement will be included in a future revision or the final version of the paper.

Declaration of Competing Interest

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

Data Availability

All relevant source code and artefacts are available in our replication package [41], which is published as a long-term archive.

Acknowledgements

This work was supported by the FFG (Austrian Research Promotion Agency) project MODIS (no. FO999895431).

References

[1] P. Kriens and T. Verbelen, "Software engineering practices for machine learning," *CoRR*, vol. abs/1906.10366, 2019. [Online]. Available: http://arxiv.org/abs/1906.10366

[2] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 2503–2511.

[3] S. Amershi, A. Begel, C. Bird, R. A. Deline, H. C. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 291–300, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:88499204

[4] M. S. Rahman, E. Rivera, F. Khomh, Y. Guéhéneuc, and B. Lehnert, "Machine learning software engineering in practice: An industrial case study," *CoRR*, vol. abs/1906.07154, 2019. [Online]. Available: http://arxiv.org/abs/1906.07154

[5] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM Computing Surveys*, vol. 55, pp. 1 – 29, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:227053929

[6] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu, "Taking human out of learning applications: A survey on automated machine learning," 2019.

[7] M.-A. Zöller and M. F. Huber, "Benchmark and survey of automated machine learning frameworks," 2021.

[8] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," *IEEE Access*, vol. 11, pp. 31 866–31 879, 2023.

[9] N. T. Hewage and D. A. Meedeniya, "Machine learning operations: A survey on MLOps tool support," *ArXiv*, vol. abs/2202.10169, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:247011728

[10] D. Sweenor, S. Hillion, D. Rope, D. Kannabiran, T. Hill, M. O'Connell, and a. O. M. C. Safari, *ML Ops: Operationalizing Data Science*. O'Reilly Media, Incorporated, 2020. [Online]. Available: https://books.google.at/books?id=StTvzQEACAAJ

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

[12] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

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

[14] U. Zdun, E. Ntentos, K. Plakidas, A. E. Malki, D. Schall, and F. Li, "On the design and architecture of deployment pipelines in cloud- and service-based computing - a model-based qualitative study," *2019 IEEE International Conference on Services Computing (SCC)*, pp. 141–145, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:201810286

[15] M. Shahin, M. Zahedi, M. Ali Babar, and L. Zhu, "An empirical study of architecting for continuous delivery and deployment," *Empirical Software Engineering*, vol. 24, 08 2018.

[16] G. Schermann, J. Cito, P. Leitner, U. Zdun, and H. Gall, "An empirical study on principles and practices of continuous delivery and deployment," PeerJ Preprints, Tech. Rep., 2016.

[17] D. A. Tamburri, "Sustainable MLOps: Trends and challenges," *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 17–23, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:232063132

[18] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas, "MLOps - definitions, tools and challenges," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0453–0460.

[19] S. Moreschi, G. Recupito, V. Lenarduzzi, F. Palomba, D. Hastbacka, and D. Taibi, "Toward end-to-end MLOps tools map: A preliminary study based on a multivocal literature review," 2023.

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

[21] ——, "Architectural design decisions for machine learning deployment," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*. IEEE, 2022, pp. 90–100.

[22] L. Faubel and K. Schmid, "An analysis of MLOps practices," Software Systems Engineering, Institut für Informatik, Universität Hildesheim, - Universitätsplatz 1, 31134 Hildesheim, Hildesheimer Informatik-Berichte 1/2023, SSE 1/23/E, 2023.

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

[24] T. Mboweni, T. Masombuka, and C. Dongmo, "A systematic review of machine learning DevOps," in *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2022, pp. 1–6.

[25] M. Testi, M. Ballabio, E. Frontoni, G. Iannello, S. Moccia, P. Soda, and G. Vessio, "MLOps: A taxonomy and a methodology," *IEEE Access*, vol. 10, pp. 63 606–63 618, 2022.

[26] A. Lima, L. Monteiro, and A. Furtado, "MLOps: Practices, maturity models, roles, tools, and challenges - a systematic literature review," in *International Conference on Enterprise Information Systems*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:248718186

[27] Y. Luo, M. Raatikainen, and J. K. Nurminen, "Autonomously adaptive machine learning systems: Experimentation-driven open-source pipeline," in *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Sep. 2023, pp. 44–52.

[28] Y. Haviv and N. Gift, *Implementing MLOps in the enterprise*. Sebastopol, CA: O'Reilly Media, Dec. 2023.

[29] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam, "Demystifying MLOps and presenting a recipe for the selection of open-source tools," *Applied Sciences*, vol. 11, no. 19, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/19/8861

[30] 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)*, 2020, pp. 494–500.

[31] S. Garg, P. Pundir, G. Rathee, P. Gupta, S. Garg, and S. Ahlawat, "On continuous integration / continuous delivery for automated deployment of machine learning models using MLOps," in *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 2021, pp. 25–28.

[32] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pp. 109–120, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:13492610

[33] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Software Engineering, International Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, may 2001, p. 0297. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE.2001.919103

[34] O. Zimmermann, U. Zdun, T. Gschwind, and F. leymann, "Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method," in *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, 2008, pp. 157–166.

[35] R. Sharma and K. Davuluri, "Design patterns for machine learning applications," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, 2019, pp. 818–821.

[36] H. Washizaki, H. Uchida, F. Khomh, and Y.-G. Guéhéneuc, "Machine learning architecture and design patterns," *IEEE Software*, vol. 8, 2020.

[37] H. Washizaki, F. Khomh, Y.-G. Guéhéneuc, H. Takeuchi, N. Natori, T. Doi, and S. Okuda, "Software-engineering design patterns for machine learning applications," *Computer*, vol. 55, no. 3, pp. 30–39, 2022.

[38] L. Cardoso Silva, F. Rezende Zagatti, B. Silva Sette, L. Nildaimon dos Santos Silva, D. Lucrédio, D. Furtado Silva, and H. de Medeiros Caseli, "Benchmarking machine learning solutions in production," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020, pp. 626–633.

[39] J. Zhou, A. H. Gandomi, F. Chen, and A. Holzinger, "Evaluating the quality of machine learning explanations: A survey on methods and metrics," *Electronics*, vol. 10, no. 5, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/5/593

[40] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine learning interpretability: A survey on methods and metrics," *Electronics*, vol. 8, no. 8, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/8/832

[41] S. J. Warnett and U. Zdun, "A Model-Driven, Metrics-Based Approach to Assessing Support for Quality Aspects in MLOps System Architectures: Replication Package," Oct. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13941649

[42] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. New York, NY: Aldine de Gruyter, 1967.

[43] A. L. Strauss and J. M. Corbin, *Basics of qualitative research: techniques and procedures for developing grounded theory*. Sage Publications, Thousand Oaks, Calif, 1998.

[44] J. Corbin and A. L. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociology*, vol. 13, pp. 3–20, 1990.

[45] S. J. Warnett and U. Zdun, "On the understandability of MLOps system architectures," *IEEE Transactions on Software Engineering*, vol. 50, no. 5, pp. 1015–1039, 2024.

[46] ——, "Architectural Design Decisions for the Machine Learning Workflow: Dataset and Code," Nov. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5730291

[47] ——, "Architectural Design Decisions for Machine Learning Deployment: Dataset and Code," Jan. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5823459

[48] ——, "On the Understandability of MLOps System Architectures: Dataset and Code," Feb. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.7752176

[49] J. Frank E. Harrell, *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*, 2nd ed. Springer, 2015.

[50] A. Airola, T. Pahikkala, W. Waegeman, B. De Baets, and T. Salakoski, "An experimental comparison of cross-validation techniques for estimating the area under the ROC curve," *Computational Statistics & Data Analysis*, vol. 55, no. 4, pp. 1828–1844, 2011.

[51] G. W. Brier, "Verification of forecasts expressed in terms of probability," *Monthly Weather Review*, vol. 78, no. 1, pp. 1 – 3, 1950. [Online]. Available: https://journals.ametsoc.org/view/journals/mwre/78/1/1520-0493_1950_078_0001_vofeit_2_0_co_2.xml

[52] M. Cowles and C. Davis, "On the origins of the .05 level of statistical significance," *American Psychologist*, vol. 37, pp. 553–558, 05 1982.

[53] D. I. K. Sjøberg, B. Anda, and A. Mockus, "Questioning software maintenance metrics: A comparative case study," in *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2012, pp. 107–110.

Stephen John Warnett is a researcher with the Research Group Software Architecture at the Faculty of Computer Science, University of Vienna, Austria. His research interests include the intersection of software engineering, software architecture and artificial intelligence. Stephen received a bachelor of science degree with First Class Honours in computer science from the University of Edinburgh, Scotland, and a master of science degree with distinction in engineering from the University of Applied Sciences Technikum Wien, Austria.

Evangelos Ntentos is a PostDoc researcher with a Doctoral Degree in Software Architecture from the University of Vienna. He has been involved in various research projects focusing on DevOps and MLOps practices and architecture evaluation. His teaching experience spans several years as a teaching assistant in the Research Group Software Architecture at the University of Vienna. Evangelos has published in peer-reviewed conferences and journals on topics such as architectural design decisions, architecture evaluation in microservices, RLOps and Infrastructure as Code.

Uwe Zdun is a full professor of software architecture at the Faculty of Computer Science, University of Vienna. His research focuses on software design and architecture, distributed systems engineering (service-based, cloud, IoT, and microservices systems), SW engineering for ML, ML for SW engineering, DevOps and continuous delivery, and empirical software engineering. Uwe has published over 300 peer-reviewed articles and co-authored 4 professional books. He is Associate Editor of the Journal of Systems and Software (JSS) published by Elsevier, Associate Editor of the Computing journal published by Springer, and Associate Editor for design and architecture for the IEEE Software magazine.