The Journal of Systems & Software 199 (2023) 111615

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

The pipeline for the continuous development of artificial intelligence models—Current state of research and practice☆

Monika Steidl [a,*], Michael Felderer [a,b], Rudolf Ramler [c]

[a] University of Innsbruck, 6020 Innsbruck, Austria
[b] Blekinge Institute of Technology, 371 79 Karlkskrona, Sweden
[c] Software Competence Center Hagenberg GmbH (SCCH), 4232 Hagenberg, Austria

ARTICLE INFO

Article history:
Received 30 March 2022
Received in revised form 27 December 2022
Accepted 10 January 2023
Available online 14 January 2023

MSC:
08-09
99-00

Keywords:
Continuous development of AI
Continuous (end-to-end) lifecycle pipeline for AI
MLOps
CI/CD for AI
DevOps for AI
Multivocal literature review

ABSTRACT

Companies struggle to continuously develop and deploy Artificial Intelligence (AI) models to complex production systems due to AI characteristics while assuring quality. To ease the development process, continuous pipelines for AI have become an active research area where consolidated and in-depth analysis regarding the terminology, triggers, tasks, and challenges is required.

This paper includes a Multivocal Literature Review (MLR) where we consolidated 151 relevant formal and informal sources. In addition, nine-semi structured interviews with participants from academia and industry verified and extended the obtained information. Based on these sources, this paper provides and compares terminologies for Development and Operations (DevOps) and Continuous Integration (CI)/Continuous Delivery (CD) for AI, Machine Learning Operations (MLOps), (end-to-end) lifecycle management, and Continuous Delivery for Machine Learning (CD4ML). Furthermore, the paper provides an aggregated list of potential triggers for reiterating the pipeline, such as alert systems or schedules. In addition, this work uses a taxonomy creation strategy to present a consolidated pipeline comprising tasks regarding the continuous development of AI. This pipeline consists of four stages: *Data Handling*, *Model Learning*, *Software Development* and *System Operations*. Moreover, we map challenges regarding pipeline implementation, adaption, and usage for the continuous development of AI to these four stages.

1. Introduction

An increase in available data and computing power, as well as improving algorithms, allow exploring the options of AI in many different application fields. AI and its subcategories, Machine Learning (ML) and Deep Learning (DL), enable new intelligent products and services to achieve a specific goal (Boucher, 2020).

To harness the power of AI, it is necessary to deploy and integrate AI models into production systems and to assure the quality of the resulting continuously evolving and self-adapting systems (Fursin et al., 2020; Stone et al., 2016; Tao et al., 2019; Pieters, 2011). However, quality assurance requires thorough attention to guarantee safe and reliable behaviour and increase the accountability and responsibility of the involved AI systems (Pieters, 2011; Tao et al., 2019; Hand and Khan, 2020; Lenarduzzi et al., 2020). AI characteristics such as the inherent non-determinism lead to a certain degree of uncertainty (Stone et al., 2016; Tao et al., 2019; Pieters, 2011).

One possible solution for ensuring quality during the development of AI are automated end-to-end CI/CD lifecycle pipelines (Mishra and Otaiwi, 2020). These pipelines are well established in traditional software development; however, need more research when adapting them to AI models because these pipelines not only need to handle code but also data and the model itself in addition to a large system-level complexity (Fischer et al., 2020; Granlund et al., 2021). These pipelines focus on automating and monitoring all phases of system development, such as the integration, testing, and deployment, as well as the management of the infrastructure.

These pipelines for the continuous development of AI are currently highly researched, where a synthesis of the current research provides an evidence-based foundation of the established work to avoid misconceptions, discover gaps in the knowledge field and assist research in exploring the phenomenon with further studies. Thus, the main goal of this paper is to systematically identify relevant conceptual ideas, as well as synthesize and structure research in the area of pipelines for the continuous development of AI. Three research questions (RQ) have been

☆ Editor: Alexander Serebrenik.

* Corresponding author.
E-mail addresses: monika.steidl@uibk.ac.at (M. Steidl), michael.felderer@uibk.ac.at (M. Felderer), Rudolf.Ramler@scch.at (R. Ramler).

https://doi.org/10.1016/j.jss.2023.111615

derived from this overall goal, which we answer in this paper via a Multivocal Literature Review (MLR) and follow-up interviews with practitioners from academia and industry.

1. Which terms are commonly used to describe a pipeline for the continuous development of AI? How do these terms differ in their specific meaning?
2. Which tasks have to be handled by pipelines for the continuous development of AI? What are possible triggers for starting the pipeline for the continuous development of AI?
3. What are potential challenges when implementing, adapting, and using a pipeline for the continuous development of AI?

The remainder of the paper is structured as follows: Section 2 presents necessary background knowledge on continuous software engineering and DevOps. Furthermore, this section also provides an overview of related work with regards to the pipelines for the continuous development of AI and describes the novel contribution provided by this paper. Section 3 explains the information extraction and taxonomy creation methodologies. Section 4 introduces the terminology, triggers, taxonomy, and challenges. Section 5 applies TensorFlow Extended (TFX), a lifecycle pipeline for AI, to the proposed taxonomy. Section 6 handles the threats to validity and Section 7 concludes the paper and introduces future work.

2. Background and related work

2.1. Continuous software engineering and DevOps

This section broadly specifies the main background knowledge required for this paper. This section covers the general terms continuous software engineering, CI/CD, and DevOps followed by a more detailed description of these terms in the context of AI. For a more detailed description of the AI related terms, please see Section 4.1.

According to the established roadmap for continuous software engineering by Fitzgerald and Stol (2017), continuous software engineering describes the continuous development lifecycle which includes continuous practices and concepts, such as Continuous Integration (CI), Continuous Delivery (CD), continuous deployment and Development and Operations (DevOps) (Fitzgerald and Stol, 2017). CI is a process that focuses on integrating code changes to the main software repository while automatically ensuring software quality (Spieker and Gotlieb, 2019; Yasar, 2020a; Pentreath, 2019; Ståhl and Bosch, 2014). CD describes the tasks after CI and delivers or releases the new and tested software features to a staging or test environment (Gmeiner et al., 2015; Karamitsos et al., 2020; Yasar, 2020a; Vadavalasa, 2020). Continuous Deployment requires that CD already deployed the software to some environment other than production to ensure that the software can be continuously and automatically deployed to the production environment and to the actual users (Karamitsos et al., 2020; Yasar, 2020a; Fitzgerald and Stol, 2017). CI/CD for AI are techniques to automate the deployment process for AI models (Borg, 2021; Zhang et al., 2020) (see Section 4.1.2).

DevOps is a continuous software development approach that includes several principles and practices, such as CI, CD, and continuous deployment to manage a software system lifecycle. The term consists of Development (Dev) and Operations (Ops). Dev uses agile methods, such as Scrum or Kanban, and allows a self-directed and self-organized software development with several teams (Stirbu et al., 2021; Yasar, 2020a; Kim et al., 2016). Ops includes the tasks necessary to run an application, such as infrastructure management (Yasar, 2020a). DevOps for AI not only takes into consideration traditional software development but focuses on the added complexity of AI development, such as data handling (Rausch and Dustdar, 2019) (see Section 4.1.1).

MLOps expands DevOps and takes into consideration the added complexity of developing ML based applications (Anon, 2021b) (see Section 4.1.3).

The (end-to-end) lifecycle management describes the handling of specific tasks for the continuous development of AI, which starts with data collection and finishes with the deployment and monitoring of the AI model (Aguilar Melgar et al., 2021; Brumbaugh et al., 2019; Zhou et al., 2020) (see Section 4.1.4).

Continuous Delivery for Machine Learning (CD4ML) is the technical implementation of MLOps concept to automate the pipeline (Mäkinen et al., 2021; Shtelma and Shiviah, 2020) (see Section 4.1.5).

2.2. Related work

Over the past years, a large number of publications focused on the topic of pipelines for the continuous development of AI (e.g., Tamburri (2020), Fischer et al. (2020), Renggli et al. (2021), Mäkinen et al. (2021), Mäkinen (2021), Alnafessah et al. (2021), Nguyen-Duc et al. (2020), Kolltveit and Li (2022), Nguyen-Duc et al. (2020), Amershi et al. (2019), Martínez-Fernández et al. (2022), Washizaki et al. (2019), Lewis et al. (2021)) as well as data handling (Munappy et al., 2020; Rodriguez et al., 2020; Ereth, 2018). In Table 1, we list work that is most closely related to our study regarding the continuous development of AI due to their similar methodology (Systematic Literature Review (SLR) and Multivocal Literature Review (MLR)). The table describes the scope and research goal of related studies, and it indicates how the work maps to the three research questions addressed by our paper. The following paragraphs then provide an extensive analysis of the identified related work.

Firstly, several related studies (Karamitsos et al., 2020; John et al., 2021a,b; Lwakatare et al., 2020b; Figalist et al., 2020; Fredriksson et al., 2020; Kolltveit and Li, 2022; Kreuzberger et al., 2022; Lorenzoni et al., 2021) are based on a limited amount of identified primary sources, not covering relevant insights from the wide range of existing literature. Our study is based on a comprehensive analysis including over 150 papers.

Secondly, some literature reviews in related work focus on a specific application context such as edge/cloud/hybrid architectures (John et al., 2021a), ML-based software analytics and business intelligence applications (Figalist et al., 2020), or federated learning (Lo et al., 2021). In contrast, our study covers the full scope of AI models, independently of a specific application context.

Thirdly, a range of different research questions are investigated in related studies, not or only partially related to the definition of terms (RQ1), pipelines for the continuous development of AI and triggers for starting the pipeline (RQ2), and pipeline-related challenges (RQ3).

1. **Definition of terms**: SLR or MLR based papers in the identified related work do not elaborate on the definition of terms for continuous development of AI. Mboweni et al. (2022) and Kreuzberger et al. (2022) provide a foundation-based definition on MLOps. Definitions for related terms (e.g., CI/CD for AI) are not considered.
2. **Pipeline tasks for the continuous development of AI**: Related studies target various different approaches and research goals. For instance, Karamitsos et al. (2020) base their CI/CD pipeline for AI on a literature review focusing on DevOps principles and methods such as CRISP-DM, SEMMA and TDSP. Generally, less emphasis is put on tasks necessary to continuously develop AI. John et al.

Table 1
Overview of related work.

Reference	Research method	# Papers	Review period	RQ1: terms	RQ2: tasks	RQ3: challenges	Differences in scope or goal
Karamitsos et al. (2020)	SLR	–	–		•		applied DevOps practices to AI development
John et al. (2021a)	MLR	29	1.2010–8.2020		•	•	context: edge/cloud/hybrid architectures
John et al. (2021b)	MLR	19	1.2015–3.2021		•		maturity model based on MLOps
Lwakatare et al. (2020b)	MLR & interviews	8	–		•		how well CD is applied to ML-enabled systems
Figalist et al. (2020)	SLR & framework	–	–		•	•	context: ML-based software analytics/BI solutions
Lo et al. (2021)	SLR incl. grey lit.	231	1.2016–1.2020				context: federated learning
Nascimento et al. (2020)	SLR	55	1990–2019			•	relationship between SE practice & AI development
Mboweni et al. (2022)	SLR	60	2015–2022	•			term MLOps and main themes in literature
Fredriksson et al. (2020)	SLR	43	before 12.2019				context: (semi-) automatic labelling of data types for ML
Testi et al. (2022)	SLR	–	2015–2022		•	•	classify pipeline types, challenges for AI development in pipeline
Kolltveit and Li (2022)	SLR	24	after 2015		•	•	operationalize AI
Kreuzberger et al. (2022)	SLR & interviews	27	before 5.2021	•	•	•	preprint only: principles and profession for realizing MLOps
Lorenzoni et al. (2021)	SLR incl. grey lit.	33	1.2010–6.2020		•		preprint only; applicability of SE and ML per author, no aggregated taxonomy of tasks
Xie et al. (2021)	systematic mapping study	405	before 7.2020				preprint only; mapping study of AI model lifecycle management (no focus on tasks)
Our study	MLR & interviews	151	2010–2021	•	•	•	AI pipelines: definition of terms, taxonomy of tasks, challenges

• indicates full or partial coverage of targeted RQs.

(2021b) proposes a MLOps maturity model consisting of tasks refering to data, development and release of the ML model. Lwakatare et al. (2020b) executed a MLR to identify how well CD is applied to ML-enabled systems and proposed levels of automation, where the first level indicates the manual process and the fifth level is the fully automated and integrated process where CD is incorporated into the ML workflow process (Lwakatare et al., 2020b). Fredriksson et al. (2020), for example, also executed a SLR but only cover approaches to label different data types to be used for supervised training. Testi et al. (2022) summarize different types of MLOps pipelines, such as ML-based software systems, ML use case applications, ML automated framework where tasks of this automated framework are briefly summarized. Kolltveit and Li (2022) do not consider all required tasks for generating an AI model but focus on operationalising the model via packaging, integration, deployment, serving, inference and monitoring and logging. Kreuzberger et al. (2022) cover principles within technical components (e.g., reproducibility achieved via feature store) and required professions to build the pipeline. Lorenzoni et al. (2021) identified which software engineering processes and practices can be applied to solve issues arising during the development of ML models. However, the paper does not contain an in-depth analysis of the CI/CD phases, and how developers implement them. Moreover, their paper does not consider continuous execution of such phases, as described in this paper. Xie et al. (2021) executed a systematic mapping study to identify demographic data, such as when and where the papers were published, which research methods were applied, and which subtopics were covered in the literature. However, they did not focus on the continuous development tasks but used search terms focusing on characteristics regarding the lifecycle, such as traceability, reproducibility, guidelines, and transparency.

As indicated, DataOps is also addressed in related work by Rodriguez et al. (2020), Munappy et al. (2020), and Ereth (2018) with the focus on the first step of the pipeline for the continuous development of AI, namely data handling. With this paper, we expand their work by investigating all necessary tasks of the entire pipeline including the deployed and monitored of an AI model.

3. **Triggers**: No information regarding triggers of the pipeline for the continuous development is covered by the related SLRs and MLRs.
4. **Pipeline related challenges**: Challenges covered by related work mostly focus on the development of AI in general, but not on the pipeline itself. For instance, Nascimento et al. (2020) illustrate the relationship and dependencies between software engineering practices based on the SWEBOK knowledge areas and respective challenges regarding the development of AI systems. Kreuzberger et al. (2022) focus on organizational, ML system, and only very superficially on operational challenges for adopting MLOps, which is the core of this paper's challenges. Figalist et al.

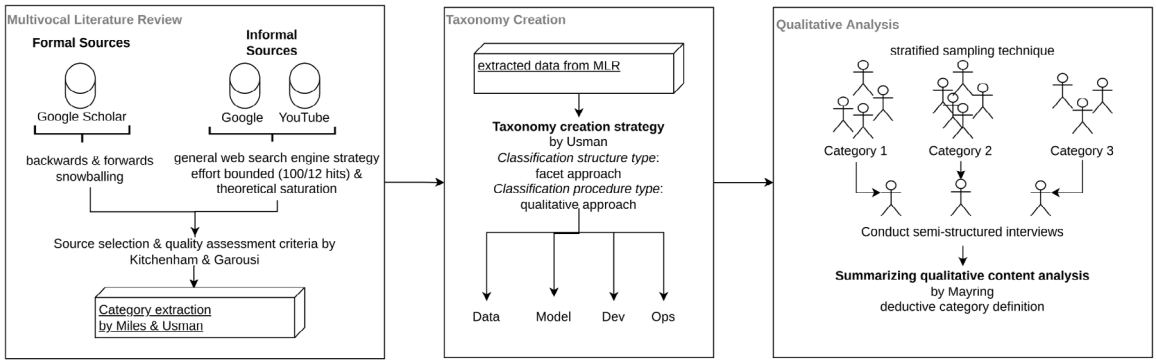

Fig. 1. Study design for MLR based on Garousi et al. (2019), taxonomy development method based on Usman et al. (2017), and deductive category definition based on Mayring (2015).

(2020) identify challenges during prototyping, deployment and update, but they specifically focus on AI models for software analytics and business intelligence. Testi et al. (2022) illustrate AI specific challenges which do not focus on the implementation of the continuous pipeline, such as data labelling. Kolltveit and Li (2022) cover exclusively challenges regarding operationalising the AI model.

Related work, which does not follow the same methodology as our study also focuses on challenges occurring throughout the pipeline for the continuous development of AI systems. These studies, however, identify general challenges of individual tasks during the development process of AI applications. For instance, Paleyes et al. (2020) identify data collection as a challenge, covering issues regarding related storage location and understanding the data set's structure. Baier et al. (2019) differentiate challenges occurring during the pre-deployment (e.g., data structure, data quality, and governance) and deployment (e.g., detecting and handling data drifts) as well as non-technical challenges (e.g., expectation management, trust, transparency). Lewis et al. (2021) only briefly mention general challenges regarding ML system development, summarized to data management, modelling and operationalization, with the aim to evaluate how well available tools are able cope with these challenges.

3. Methodology

In the following, this section covers the three employed research methods (MLR, taxonomy creation strategy and qualitative analysis) to derive the main research contributions. Fig. 1 illustrates a general overview of the research design consisting of a MLR proposed by Garousi et al. (2019) to identify existing literature, taxonomy development method based on Usman et al. (2017) to map the identified aspects in literature to a taxonomy, and the interviews' deductive category definition based on Mayring (2015). These research methods are further explained in the following sections.

3.1. MLR

A MLR was executed to provide a thorough overview and aggregated evidence of important perspectives of pipelines for the continuous development of AI. This allows not only to include published literature but also allows to include grey literature. Grey literature is essential because using a lifecycle pipeline for AI is an emerging research topic in software engineering where formal literature has not been sufficiently published yet (Garousi et al., 2019). The following approach is based on the paper 'Guidelines for including grey literature and conducting multivocal literature reviews in software engineering' provided by Garousi et al. (2019).

Table 2 depicts the general **selection criteria** that apply to both selection processes of formal as well as informal sources.

Table 3 depicts specific selection criteria for **formal sources**. Based on these criteria we derived 37 formal sources as the start data set. Afterwards, we followed Wohlin's proposed backward and forward snowballing procedure (Wohlin, 2014). We derived the final data set that comprises 79 papers between the 15th of April and the 30th of May 2021.

In order to retrieve **formal/scientific sources**, we executed an initial exploratory search with several search terms (see Fig. 2) to collect a start data set (Garousi et al., 2019). We followed the guidelines proposed by Kitchenham and Charters (2007) and Wohlin (2014), Jalali and Wohlin (2012). We used Google Scholar as the search engine to retrieve an unbiased start data set. According to Yasin et al. (2020) results from Google Scholar in combination with grey literature extracted from Google sufficiently extracts necessary sources similar to searches with other databases, such as ScienceDirect, IEEE, ACM digital library and Springer Link.

Table 4 depicts the specific selection criteria for **informal sources**. Based on these criteria we derived informal sources between the 31st of May and the 26th of June 2021.

We adopt *quality assessment criteria* to select the sources based on their relevance and to check whether the informal literature search results are valid and free of bias (Garousi et al., 2019). We derive the sources based on the proposed *general web search engine* strategy where we use conventional web search engines (Garousi et al., 2019). Regarding the stopping criteria, we selected two strategies. Firstly, *effort bounded* looks at a predefined number of search engine hits. Secondly, *theoretical saturation* identifies whether no new concepts emerge from additional search results. *Theoretical saturation* was not achieved in seven out of twenty cases.

Fig. 2 illustrates the used ten *search terms* which were based on the established terms of continuous software engineering (Fitzgerald and Stol, 2017). Additional search strings were extracted during an informal pre-search. Example for the used search strings are `Artificial Intelligence AI AND Continuous Integration CI`, `Machine Learning Operations AND MLOps`, and `Machine Learning Operations OR MLOps`.

The MLR identified 151 relevant sources, out of which 79 papers (approximately 53%) were formal sources and 72 informal sources. The extraction process and the retrieved formal and informal literature were documented in a systematic map which is available online (Steidl et al., 2022).

Table 2
General selection criteria for formal and informal sources.

	Inclusion criteria	Exclusion criteria
Year of publication	2010–2021	before 2010
Language	English	Any other language
Accessibility	Full text needs to be accessible	Parts of source available
Relevance	Relevant information to answer the main research questions (e.g. Does the source focus on pipelines for the continuous development of AI)	Sources that focus on using AI to implement DevOps such as Artificial Intelligence for IT Operations (AIOps) or which focus only on team related processes (e.g. team collaboration)

Table 3
Selection criteria for formal sources.

	Inclusion criteria	Exclusion criteria
Quality criteria	Primary peer-reviewed sources	No review process, secondary studies
Search strategy	Google Scholar	Other databases
Stopping criteria	First 100 search hits	Search hits after

Table 4
Selection criteria for informal sources.

	Inclusion criteria	Exclusion criteria
Quality criteria	Fulfil Garousi et al.'s (Garousi et al., 2019) quality assessment criteria	deviation of Garousi et al.'s (Garousi et al., 2019) quality assessment criteria
Search strategy	General web search engine (Google, YouTube)	Specialized databases and websites (e.g. Stackoverflow), contacting individuals directly
Stopping criteria	Effort bounded: Google (100 search hits), YouTube (12 search hits) Theoretical saturation required, otherwise additional hits searched: Google (50 search hits), YouTube (20 search hits)	Search hits after effort bounded and theoretical saturation are fulfilled

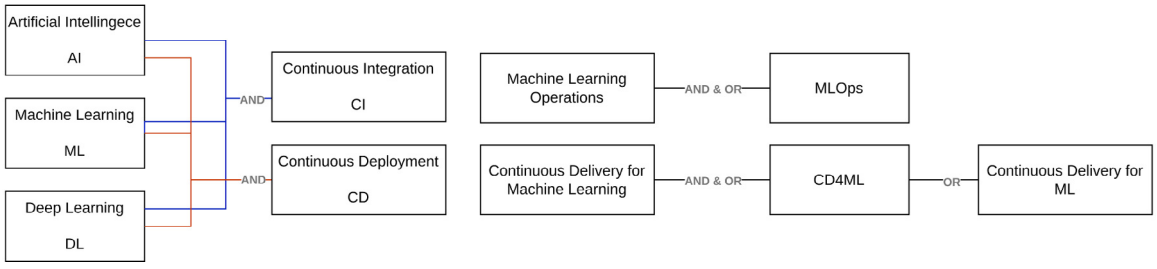

Fig. 2. Search terms used for the Multivocal Literature Review (MLR).

To **extract** the necessary **categories** from the literature, we executed a descriptive qualitative synthesis. Kitchenham and Charters (2007) require to document the extracted information in a tabulated and consistent manner based on the previously defined research questions. For further information on the tabulation of the extracted information, please refer to Steidl et al. (2022). We derive further subcategories of the research questions as suggested by Stol et al. (2016) and create the taxonomy as proposed by Usman et al. (2017). We adopted Stol et al. (2016) coding strategy, namely open and axial coding to break down, examine, compare, conceptualize, and categorize information. We base the categories on a previous pilot study. These categories were closely related to the terms used in continuous software engineering, hence commonly accepted within the field (Usman et al., 2017). Because the predefined set of categories did not cover all tasks handled with the pipeline for the continuous development of AI, we added additional categories via an iterative process.

After selecting the source and extracting necessary categories, we executed a **test–retest process** proposed by Kitchenham and Charters (2007) to evaluate the rater's data extraction consistency. Further information is provided in Section 6.

3.2. Taxonomy creation

We categorized the extracted data via a taxonomy based on the revised taxonomy creation strategy proposed by Usman et al. (2017). Firstly, we defined the units of the classes/categories which are based on DevOps phases because they are commonly accepted within the field. We add extracted information to the respective class/category via a qualitative approach (Usman et al., 2017). For the classification structure type, we use a facet approach because research on pipelines for the continuous development of AI applications is still a new and evolving field. The facet approach allows us to easily adapt the taxonomy if further

Table 5
Defined categories for the stratified sampling.

	Category 1	Category 2	Category 3
Area of Research	Academia	Industry	Industry (Start-ups)
Experience with AI	Yes	Yes	Yes
Experience with lifecycle pipelines	Yes	Yes	Should know concept of continuous lifecycle pipeline
General experience	Published work on lifecycle management for AI	Extensive & regular deployment of AI developers (no data scientists)	AI development and deployment

Table 6
Overview and background of interviewed participants.

Participant	Knowledge about pipelines for the continuous development of AI	Experience	Industry	Region	Category
A	Team lead for AI initiative un- & supervised algorithms	5.5 years	Social Media	America	2 - Industry
T	Research projects: continuous software engineering practices for traditional and AI software	10 years	Academia	Finland	1 - Academia
Z	Sales Engineer for MLOps platform, AI development	1 year	MLOps software	America	2 - Industry
P	Team Lead AI application engineering	2 years	Text analysis	Austria	3 - Industry (Start-ups)
R	Technical Team Lead for ML deliveries (e.g. sentiment analysis, speech assistant)	3.5 years	Automotive	Germany	2 - Industry
B	Research (Area manager for services and solutions): AI innovations & image processing	5.5 years	Research/ Consultancy	Austria	2 - Industry
C	Implementation of pipeline for the continuous development of AI Experience in Kubernetes & ML	4 years	MLOps pipeline	America	3 - Industry (Start-ups)
V	Research (Senior research project manager): pipeline for the continuous development of AI with TFX	3 years	Research/ Consultancy	Austria	2 - Industry
D	Regulatory compliance in MLOps & Certification body for AI	2 years	Academia (Healthcare)	Finland	1 - Aca-demia

research is done on this topic. The identified facets comprise the stages *Data Handling*, *Model Learning*, *Software Development* and *System Operations*.

3.3. Qualitative analysis

We checked via a qualitative approach if the derived information from the literature is correct and comprehensive enough to provide a thorough depiction of existing knowledge. For this, we conducted interviews because they allowed us to explore and understand individual experiences from a sample by outlining the complexity and diversity of the observed environment (Miles et al., 2014). To select the interview participants, we adopt a stratified sampling technique (Robinson, 2014) with the three groups illustrated in Table 5.

Based on the selection criteria, we identified nine participants. Table 6 provides an overview of the involved participants.

The semi-structured interviews included introductory questions, questions about different definitions of pipelines for the continuous development of AI, tasks handled via these pipelines, and an evaluation of the proposed taxonomy as well as challenges when implementing, adapting, and using such a pipeline. We conducted the interviews between the 30th of July and the 10th of September 2021 with an average duration of 49 min. We analysed the interviews according to Mayring (2015) with the *summarizing qualitative content analysis*. We adopted the *deductive category definition* to extract information regarding the pipeline. In addition, we categorized the information on theoretically derived aspects from the MLR (Mayring, 2015). Thus, we derive the coding agenda from the identified stages and tasks from the previously mentioned taxonomy creation. By doing so we unambiguously assign the participants' statements to the identified categories (Mayring, 2015).

4. Results

This section presents the information obtained from the literature and interviews. The following section elaborates the different terminologies, followed by identified triggers to start/restart the pipeline. The subsequent section elaborates the created taxonomy which describes the pipeline for the continuous development of AI and included tasks. The final section explores challenges regarding the implementation, adaption and usage of pipelines for the continuous development of AI.

4.1. Terminologies

In the following subsections, several terms and their main characteristics and differences, including (1) DevOps for AI, (2) CI/CD for AI, (3) MLOps, (4) (End-to-End) Lifecycle Management,

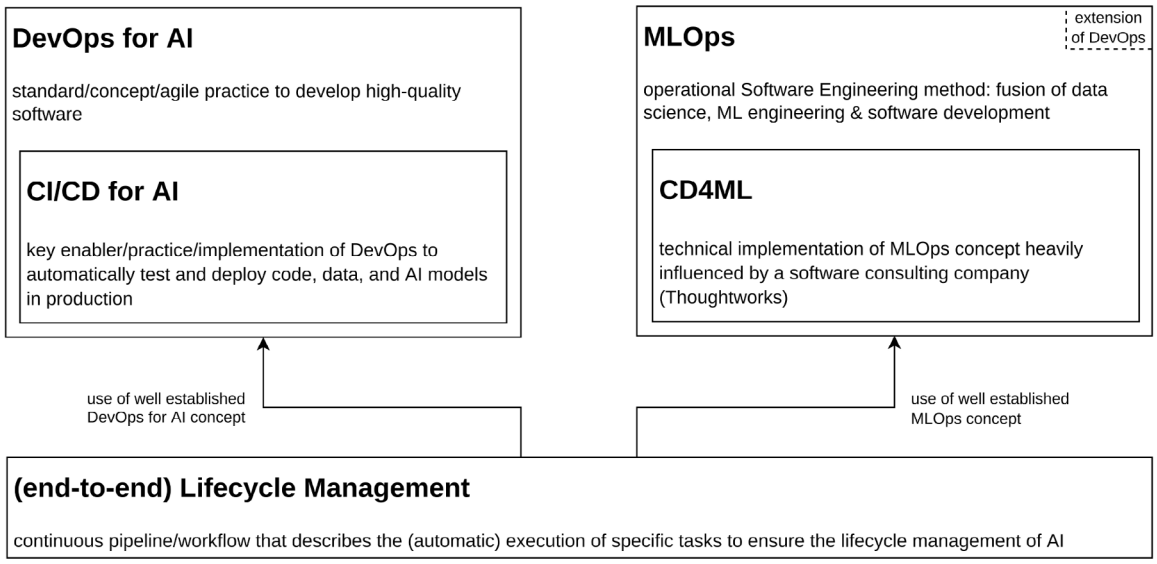

Fig. 3. Summary of terms - DevOps for AI, CI/CD for AI, MLOps, (end-to-end) lifecycle management, and CD4ML.

and (5) CD4ML are discussed. Fig. 3 illustrates the main characteristics of these terms. The interviewees indicated that the differences between the terms are unknown in practice. During the literature study, the common ground of all terms is that they describe the automation for the continuous development and improvement of AI models via pipelines.

According to three of the interviewed participants, the terms describe the process of adapting the standard software lifecycle to AI modelling in order to minimize the time between iterations and to ease the whole process of continuously developing and deploying AI models.

4.1.1. DevOps for AI

Some authors use the term DevOps for AI to describe *'[...] methods for managing the software lifecycle'* (Rausch and Dustdar, 2019) as stated by Rausch et al. It is seen as a standard for modern software development to ensure higher data, as well as code quality (Rausch and Dustdar, 2019; Breuel, 2020). DevOps for AI is a concept and agile practice to reduce time and resources between deployment iteration cycles (Rausch and Dustdar, 2019; Karlaš et al., 2020; Breuel, 2020).

4.1.2. CI/CD for AI

CI/CD are key enablers or techniques for DevOps to stabilize, optimize and automate the deployment process of AI models (Borg, 2021; Zhang et al., 2020; Baroni, 2018). According to Karlas et al. CI/CD supports the *'[...] deployment to the infrastructure used to serve models in production'* (Karlaš et al., 2020). Thus, CI/CD takes into consideration not only validating and testing code and components, but also handles the (semi-)automatic and iterative validation and testing of data, data schemas, and models (Anon, 2021b; Karlaš et al., 2020).

4.1.3. MLOps

According to Google Cloud, MLOps *'[...] appl[ies] DevOps principles to ML systems'* (Anon, 2021b) and participant Z and R confirm that MLOps is an extension of DevOps. According to Raj et al. it is an *'[...] emerging method to fuse machine learning engineering with software development'* (Raj et al.). This statement is underlined by other sources as well (Raj et al.; Martel et al.; Sangiovanni et al., 2020; Mäkinen et al., 2021; Zhou et al., 2020; Martínez-Fernández et al., 2021; Renggli et al., 2021). Breuel defines it as following: *'MLOps is a set of practices that combines ML, DevOps and Data Engineering, which aims to deploy and maintain ML systems in production reliably and efficiently'* (Breuel, 2020).

The key difference between other terms is that MLOps strongly takes into consideration the company's culture and illustrates how cross-functional teams, such as data analysts, system operators, as well as data and software engineers collaborate via a harmonized process (Sangiovanni et al., 2020; Pölöskei, 2020; Junsung et al., 2019; Martínez-Fernández et al., 2021; Xu, 2020; Stirbu et al., 2021; Tandon and Pati, 2021; Keating, 2020). This statement was confirmed by participant T and R.

Similar to DevOps for AI, MLOps helps with the continuous, quick, seamless and reliable deployment of multiple AI versions which are deployed in a heterogeneous and distributed environment via infrastructure and tools (Raj et al.; Bourgais and Ibnouhsein, 2021; Martínez-Fernández et al., 2021; Fursin et al., 2020). A new practice, called Continuous Training (CT) is introduced that according to Google Cloud *'[...] is concerned with automatically retraining and serving the models'* (Anon, 2021b). Therefore, CT uses collected feedback and production data (Anon, 2021b; Karlaš et al., 2020; Anon, 2021b,a; Mulkens, 2020).

The AI model quality is strongly dependent on the used data sets and the model is only a small part of the entire software system (Yasar, 2020a,b; Mäkinen et al., 2021; Haviv, 2020; Patel and Edwards, 2019; Ammanath et al., 2021; Renggli et al., 2021). This statement was underlined by participant Z.

According to the Google Cloud documentation (Anon, 2021b), MLOps can be divided into three different levels of maturity depending on the degree of automation. Another definition by Microsoft was identified during the interviews where five levels of technical implementation of MLOps are defined[1].

4.1.4. (End-to-end) lifecycle management

This term includes the word **management**, which describes the handling of specific tasks of the continuous development of AI (Vartak et al., 2016; Katsiapis et al., 2020). Essential management tasks start with data collection and finish with AI model deployment and monitoring in production (Aguilar Melgar et al., 2021; Brumbaugh et al., 2019; Zhou et al., 2020; Chard et al., 2019; Miao et al., 2017a), as verified by participant T. Vartak

[1] Microsoft's maturity levels for MLOps: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model, accessed 16.12.2021

et al. further specifies tasks included in the model management such as *'[...] tracking, storing and indexing large numbers of machine learning models so they may subsequently be shared, queried and analyzed'* (Vartak et al., 2016).

Based on these tasks, the **(end-to-end) lifecycle** for AI models describes a well-fitted pipeline that should achieve the best possible quality and stability of the AI components via several iterations until the model cannot be improved any further (Karlaš et al., 2020; Brumbaugh et al., 2019; Miao et al., 2017b,c). During these iterations, several data sets, artefacts, models and application configurations are created, which need to be managed, searched, shared and analysed (Vartak et al., 2016; Zhou et al., 2020). For example, it is crucial for Brumbaugh et al. to *'[...] have correct values for the features that correspond to the timestamp of the labels'* (Brumbaugh et al., 2019).

Several authors use the term continuous pipeline or workflow to describe the automatic execution and reiteration of tasks to ensure the lifecycle management of AI (Barrak et al., 2021; Miao et al., 2017c; Zhou et al., 2020; Lwakatare et al., 2020a; Baylor et al., 2019). According to Barrak et al. a *'[...] pipeline of tools [...] automate[s] the collection, preprocessing, cleaning and labelling of data.'* (Barrak et al., 2021)

The term end-to-end lifecycle management benefits from the idea of automation for the whole model lifecycle. The term uses well-established concepts from software development to cope with many model iterations, such as DevOps in combination with CI/CD (Zhou et al., 2020; Bachinger and Kronberger, 2020; Aronchick et al., 2020) and MLOps (Patel and Edwards, 2019). However, in contrast to MLOps, lifecycle management does not focus on the interpersonal collaboration between different teams (Miao et al., 2017a).

4.1.5. CD4ML

CD4ML is a technical implementation of the MLOps concept to automate the pipeline for the continuous development of AI (Mäkinen et al., 2021; Granlund et al., 2021; Sato et al., 2019b,a; Shtelma and Shiviah, 2020). Therefore, CD principles are used to span the AI lifecycle management and apply them to AI applications (Mäkinen et al., 2021; Gorcenski, 2019; Windheuser and Sato, 2020). Participant T as well as the extracted information from the literature identify that this term is proposed, promoted and heavily influenced by Thoughtworks (Mäkinen et al., 2021; Stirbu et al., 2021; Anon, 0000), which defines CD4ML as *'[...] a software engineering approach in which a cross-functional team produces machine learning applications based on code, data, and models in small and safe increments, that can be reproduced and reliably released at any time in short adaptation cycles'* (Sato et al., 2019b).

4.2. Triggers

The following section discusses four trigger types, including (1) feedback and alert systems, (2) orchestration service and schedule, (3) repository, and (4) other triggers. AI models need to be iteratively adapted and retrained to provide reliable quality in production over a long period of time (Baylor et al., 2017, 2019; Baroni, 2018; Arnold et al., 2020). Therefore, according to Moesta and Tamisin (2020) and two participants (R and Z), context-specific triggers depending on the AI model, business requirements and retraining strategies exist that start or restart the pipeline. For example, triggers may take into consideration the optimal threshold where the benefits obtained by an updated (i.e., retrained) model outweigh the effort involved in the retraining (Baylor et al., 2017; Ettun, 2019; Kronberger et al., 2020; Schelter et al., 2018; Rausch et al., 2020). Participants R and D indicated that it is a trial and error process to minimize resource consumption where several different team members identify the appropriate approach. Thus, triggers combining different approaches may also be feasible (Rausch and Dustdar, 2019; Raj et al.).

4.2.1. Feedback and alert systems

Collected feedback during runtime or alerts may be used to trigger the pipeline (Derakhshan et al., 2019). Three interviewees identified data events as a potential trigger, whereas information extracted from literature also covers data and model changes (Rausch and Dustdar, 2019; Visengeriyeva et al., 2021; Schruhl and Windheuser, 2020).

A monitoring system monitors and collects **data** from production to trigger the pipeline in case of irregular data events such as data updates and data drifts (Breck et al., 2019; Baylor et al., 2019). Data drifts occur when the distribution within the data set changes (Gorcenski, 2019; Vuppalapati et al., 2020; Garcia et al., 2018; Erb, 2019). This occurs when data varies due to seasonal changes, or any other insertion, deletion or update of data values (Baroni, 2018; Martel et al.; Liu et al., 2019; Baylor et al., 2017; Zhou et al., 2020). The interview participants highlighted the deletion of data. For example, participant T mentioned that due to privacy restrictions and the data regulation requirements in Europe, users have the right that their associated data is forgotten. Thus, according to participant T '[...] it is only fair that the deleted data is no longer used in the ML model'. Data updates may also happen if the shape of the data, such as table or constraint definitions, may change due to schema updates based on software updates, requirement changes or migrations (Gorcenski, 2019).

According to participants T, R and D, data updates should improve the model. To avoid triggering the pipeline continuously, triggers may occur periodically or when a specific threshold is attained (Amershi et al., 2019; Polyzotis et al., 2017; Breck et al., 2019). Similar to the results obtained from the literature, it is ambiguous for participant D what the appropriate amount of new data to change the outcome of a model is. According to participant R, the changes need to be extensive enough to significantly impact the model. Not mentioned in the extracted literature's information, however, indicated by participant R, is that event streaming platforms, such as Apache Kafka or other event hubs, may be used to semi-automate the triggering process.

Model updates may be triggered due to the deterioration of the model's performance, and scores in production below a specific threshold, also called model or concept drift (Martel et al.; Zhou et al., 2020; Janardhanan, 2020; Lopez Garcia et al., 2020; Renggli et al., 2019b; Xin et al., 2021). It occurs when the problem the model was designed to solve changes, and this problem needs to be reformulated (Visengeriyeva et al., 2021; Srinivasan, 2021; Rosenbaum, 2020). Technical performance scores indicating a trigger are throughput, latency, and the utilization of a Graphics Processing Unit (GPU) (Zhang et al., 2020; Baroni, 2018). Two participants (Z and C) use these performance metrics as triggers.

4.2.2. Orchestration service and schedule

Additional triggers are an orchestration service or a scheduled time (Rausch and Dustdar, 2019; Brumbaugh et al., 2019; Raj, 2020). For example, one participant triggered their pipeline once a week. On the one hand, one may argue that fixed schedules hinder the pipeline to be reactive enough or needless pipeline executions are triggered (Baylor et al., 2019). On the other hand, schedules help to optimize the retraining frequency, allocation of computing resources and execution order of pipeline jobs, which is especially important if edge and cloud resources are involved (Boag et al., 2017; Anon, 2021b; Lopez Garcia et al., 2020; Rausch et al., 2019).

4.2.3. Repository

Repository updates are used as traditional triggers to guarantee that the latest changes are tested and available to the users. For example, pull requests to the repository as a commit or merge requests identify changes to the data, model or code (Aronchick et al., 2020; Shtelma and Shiviah, 2020; Anon, 0000; Erb, 2019; Patel, 2020; Sierra, 2018). When using this approach, participant T indicated that the data set and the appropriate code should be in the same system and under the same source control.

4.2.4. Other triggers

Although manual triggers are sparsely elaborated in the collected literature (Lwakatare et al., 2020b; Raj, 2020; Liu and Brooks, 2020; Aronchick et al., 2020), three out of nine participants use triggers that involve human interaction to identify if sufficient data is available. One of the reasons is that the new data needs to be labelled manually. Another reason is that humans can better estimate if their model requires retraining and if the improved quality still satisfies the user's needs.

Another possible trigger is a change of the infrastructure, hardware, or architectural constraints to maintain the performance and functionalities of the AI model (Lopez Garcia et al., 2020; Jackson et al., 2018; Katsiapis et al., 2020; Wachsmuth et al., 2012)

4.3. Pipeline

This section covers the four pipeline stages including their tasks when triggering the pipeline. The four stages are (1) *Data Handling* - executing data handling, followed by the stage (2) *Model Learning* - implementing the model development, the stages (3) *Software Development* - building the AI application, and(4) *System Operations* - focusing on a smoothly running system and information collection in production. Fig. 4 illustrates these stages as a taxonomy that we derived from the literature and the interviews. However, Garcia et al. Garcia et al. (2018) as well as two participants (P and T) emphasized that the pipeline and task execution strongly depends on the individual context, e.g. organizational policies for running the pipeline, and whether implementing the tasks outweigh the costs. In four out of nine participants' organizations (A, B, C and V), the pipelines are not fully automated.

As depicted in Fig. 4, the pipeline is not linear but relies on **feedback** loops throughout the process. This is a key characteristic in agile development and allows to include continuous feedback to improve the AI model and the collection of relevant data from production (Amershi et al., 2019). For example, collecting data in production improves the training data set which ultimately increases model quality and efficacy (Khan, 2018).

Ensuring quality is an integral part of the whole engineering procedure carried out in several steps such as data, model and system-specific tests (Martel et al.; Mäkinen, 2021). The intention of this paper is not to give an all-encompassing picture of quality assurance techniques used for AI. It only depicts approaches which are applicable for continuous pipelines and which we identified during the MLR. In addition, the scope and types of tests strongly vary from implementation to implementation.

4.3.1. Data handling

The stage *Data Handling* covers the end-to-end lifecycle of data curation. Not only allows the pipeline to handle tasks more efficiently, but also the quality of a AI model strongly depends on the data availability, quality, and preprocessing techniques (Amershi et al., 2019; Raj et al., 2020; Banerjee et al., 2020; Visengeriyeva et al., 2021). The data pipeline manipulates the initial data via intertwined tasks, such as data preprocessing, testing, versioning, and documentation, until the data can be used for model training (Raj et al., 2020). A study conducted by Hummer et al. (2019) indicates that the data handling uses 7% of the total execution time, but this time can be reduced due to parallelized computing procedures (Pölöskei, 2020; Barrak et al., 2021; Miao et al., 2017b; Hummer et al., 2019; Brumbaugh et al., 2019). This is possible because workflows may be specified as a Directed Acyclic Graph (DAG) (Pölöskei, 2020; Barrak et al., 2021; Miao et al., 2017b; Hummer et al., 2019; Stirbu et al., 2021).

`Data Preprocessing`

Initially, data is prepared for model design and training (Visengeriyeva et al., 2021). Therefore, data collection including data injection, preparation, labelling and feature extraction needs to be executed to transform the raw data. This step is often defined in form of rules part of a script that defines how the raw data should be manipulated, transformed and compared (Martel et al.; Lwakatare et al., 2020b; Visengeriyeva et al., 2021). Some steps may be skipped, if the data set was already preprocessed in previous iterations (Rosenbaum, 2020). It is essential that the data handling and transformations undertaken during data preprocessing in the pipeline are consistent with the data handling in production to avoid a training–serving skew (Baylor et al., 2017; Zweben, 2021; Castanyer et al., 2021; Pentreath, 2019; Liu and Brooks, 2020). Three authors proposed to use TFX, an end-to-end lifecycle management platform provided by Google, to avoid the training–serving skew by exporting the tasks for data transformations that are again used in production and the training and serving pipeline does not need syncing (Baylor et al., 2017; Polyzotis et al., 2017; Olston et al., 2017; Zweben, 2021). However, sometimes deviations in the data preprocessing pipelines is desirable because data set and its size differs, persistent data stores provide the data for training whereas data in production is non-static where data needs to be processed fast (Liu and Brooks, 2020; Wilkiewicz et al., 2019).

Data can be **collected** from multiple distributed on-premise data centres, external public or private cloud storage (Sangiovanni et al., 2020; Lwakatare et al., 2020a; Banerjee et al., 2020; Brumbaugh et al., 2019; Raj et al., 2020). The data sets may be already available (e.g., open source or internally available) or needs to be collected from multiple devices where the data may be stored in different formats, such as tabular data, logs, key–value stores or input files (Amershi et al., 2019; Junsung et al., 2019; Brumbaugh et al., 2019; Aguilar Melgar et al., 2021; Karlaš et al., 2020; Jackson et al., 2018; Polyzotis et al., 2018). In cases where not enough data can be extracted, three interviewees (P, B and V) stated that they synthetically generate data to balance the data set. If too much data is available, the data set is reduced where participant A mentioned the risk of introducing bias.

The **preparation** strongly depends on the type of data (Jackson et al., 2018; Raj et al.; Sangiovanni et al., 2020; Brumbaugh et al., 2019). For example, pipelines may discard incomplete or irrelevant data or outliers and noisy records (Raj et al.; Amershi et al., 2019; Junsung et al., 2019; Rivero et al., 2020; Nashaat et al., 2019; Saucedo, 2020), anonymize data (Nashaat et al., 2019), (windowed/bucketed) aggregate data (Brumbaugh et al., 2019), or decompress and resize images, or filter and tokenize text (Raj et al.; Sangiovanni et al., 2020; Breck et al., 2019). In addition, numeric data may be normalized via feature scaling (Raj, 2020; Raj et al., 2020). The MLR only extracted the z-transformation and Box–Cox transformations (Banerjee et al., 2020) for data preparation but did not extract further information on the technical details, algorithms or implementations of the data preparation tasks.

Data labelling is necessary for supervised learning as indicated by participant P and C. Therefore, each record receives a meaningful ground truth label indicating the expected output.

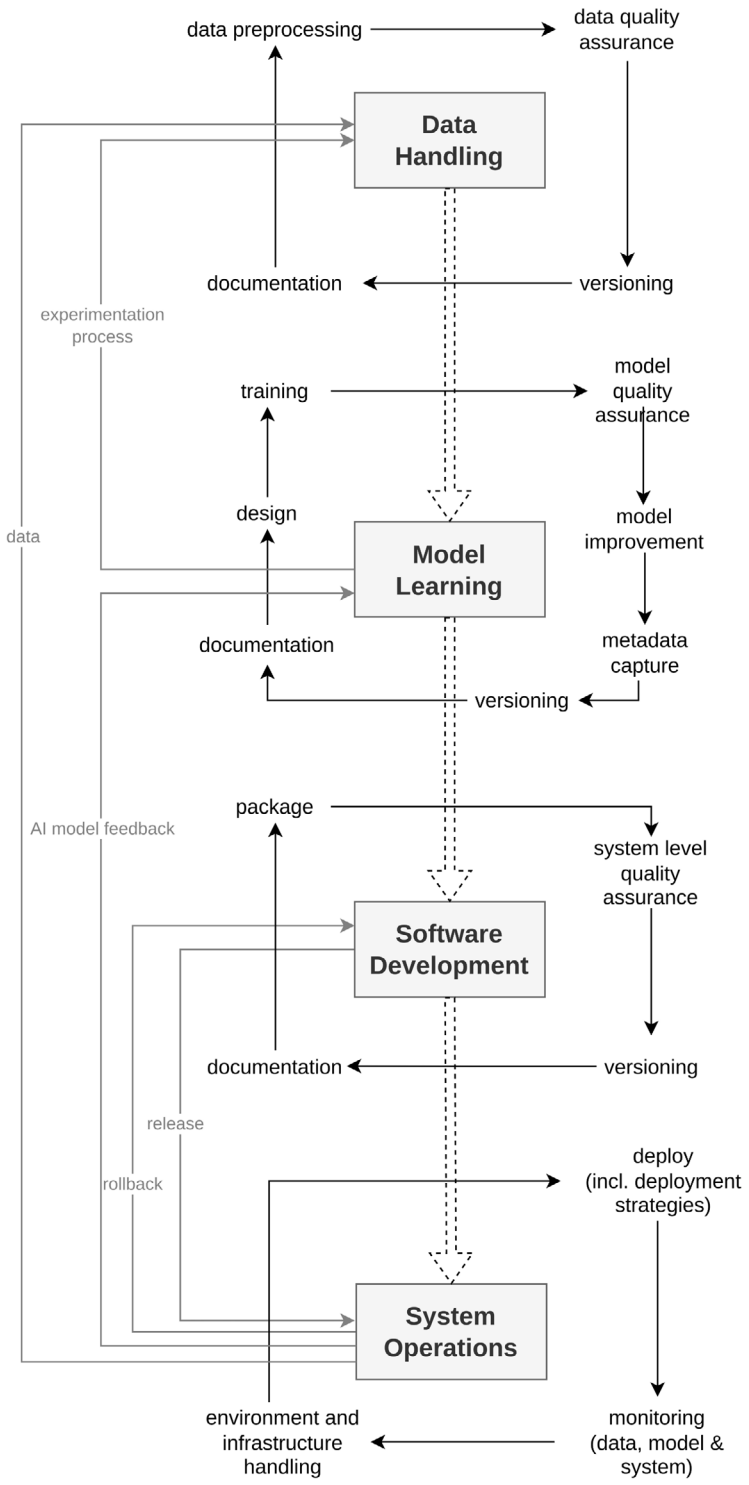

Fig. 4. Continuous lifecycle pipeline for AI applications adapted from Lwakatare et al. (2020a) and Tamburri (2020).

Other learning techniques, such as reinforcement learning, use demonstrations as labelled data (Amershi et al., 2019; Junsung et al., 2019). To automate this process, the lifecycle management platform ease.ml for example uses a model runner which identifies corresponding labels for input features (Karlaš et al., 2020). Unsupervised algorithms do not require labels, thus this task can be skipped (Raj et al., 2020; Gharibi et al., 2021).

For **feature extraction** or **feature engineering**, necessary patterns need to be manually or automatically discovered and extracted, which is strongly dependent on the AI model's context and algorithm used (Raj et al.; Sangiovanni et al., 2020; Amershi et al., 2019; Brumbaugh et al., 2019; Banerjee et al., 2020; John et al., 2020; Rivero et al., 2020). Automatic feature selection techniques include, e.g., particle swarm optimization (Xue et al., 2013), recursive feature elimination, principal component analysis (Jolliffe, 2005), and auto encoders (Ng, 2011). For supervised learning, extracting relevant features guides the model training and identifies which features are worth exploring as input for the AI model (Karlaš et al., 2020; Banerjee et al., 2020; John et al., 2020; Polyzotis et al., 2018). One important part the pipeline

should provide consistent feature extractions of the offline and online inference environment, which may be achieved by a point-in-time correctness. This is necessary because for the training data set a vector consisting of features and associated labels is used. If features from the future are used in the vector which are not part of the labels, data leakages may occur (Brumbaugh et al., 2019; Polyzotis et al., 2017).

`Data Quality Assurance`

The pipeline runs continuously, thus, reusable components for testing data are necessary to avoid introducing bugs in the data and propagating bugs down to the model training (Baylor et al., 2017; Wilkiewicz et al., 2019; Breck et al., 2019; Azimi and Pahl, 2021). This makes it easier to identify faulty data in the beginning, before it negatively influences the model and computation resources are wasting for producing inadequate results (Wilkiewicz et al., 2019; Breck et al., 2019; Polyzotis et al., 2018; Bachinger and Kronberger, 2020; Stirbu et al., 2021). Participant R and B emphasized that the better the data quality is, the better the model results are.

Data and **feature validation** validates batches of data. These batches can either be evaluated via a single-batch or inter-batch validation. Single-batch validation assumes that data batches collected in succession do not differ drastically and comply with a specific shape (Breck et al., 2019; Polyzotis et al., 2018). Thus, a stable description, such as a data schema, identifies expected features, feature types and values, and the correlation of different features (Baylor et al., 2017; Polyzotis et al., 2017; Breck et al., 2019; Sato, 2020). Participant C also uses manually or automatically created data schemas. Information provided in this schema are the expected type, presence, valency (Baylor et al., 2017; Breck et al., 2019; Caveness et al., 2020), and distribution of categorical values (Polyzotis et al., 2018; Baroni, 2018; Wilkiewicz et al., 2019). Inter-batch validation identifies differences between different batches, such as training and serving data or successive training data sets (Breck et al., 2019). Inter-batch validation identifies changes in the statistical characteristics or the encoding of feature values, such as using Boolean values instead of anticipated *1* and *0* (Caveness et al., 2020; Breck et al., 2019; John et al., 2020; Polyzotis et al., 2018). Statistical characteristics are discovered via calculating the distribution distance between training and serving data. Therefore, a distance threshold is calculated to identify if the deviation results in an error. The distance is calculated via two distributions including its probabilities (Caveness et al., 2020). Other examples for calculating the statistical characteristics are Kullback–Leibler divergence, cosine similarity, or statistical goodness-of-fit tests. These approaches are implemented in TFX (Breck et al., 2019). Participant A and R check the right shape of data as common validation practice.

Data can also be validated via a **data quality** measurement which identifies how suitable the data is to meet the users' requirements (Azimi and Pahl, 2021). Data quality can be measured via the six dimensions of data quality, namely completeness, uniqueness, consistency, validity, accuracy and timeliness (Sangiovanni et al., 2020; Amershi et al., 2019; Azimi and Pahl, 2021; Renggli et al., 2021). Completeness identifies how well the used fraction of the data set represents the corresponding main data set or real-world data. Uniqueness states that no duplicates occur in the data set. Consistency identifies the extend to which semantic rules of a data set are violated. Validity identifies if all data values comply with a specific format. Accuracy defines how well the data is suited and certified to execute a specific task. Timeliness describes if the data set is up-to-date for a task (Renggli et al., 2021). Another way of identifying if the available data is sufficient to meet the user requirements is an automatic feasibility analysis proposed by ease.ml. It calculates a bayes error based on already extracted features which are used for training. The Bayes error estimator identifies the minimum error rate achievable by any classifier. Then, this error is compared to a desired target performance, such as the accuracy. If ease.ml identifies the data set as insufficient, the user can clean up the data manually based on a list of dirty examples, or collect more data (Aguilar Melgar et al., 2021; Renggli et al., 2019a, 2020, 2021).

Unit tests can consist of tests that verify that the data ingestion works correctly (Lwakatare et al., 2020a). Unit tests can either state the input, execute a transformation and state the expected output (Sato, 2020) or use data schemas. These tests identify if there are differences between the schema and the assumptions in the code (Breck et al., 2019).

`Data Versioning`

AI models require massive amounts of data for model training which need to be stored and versioned to guarantee traceability and compliance with regulations, such as the General Data Protection Regulation (Zhou et al., 2020; Tamburri, 2020). Participant C, V and D highlighted the need for a full data lineage if important models are based on this data. Hence, not only data and its dependencies but also the data processing steps need to be versioned. In practice, however, storing the entire lineage of data is fairly impossible due to space restrictions. Thus, only the latest version is stored in participant P, T and Z's case and the previous versions are suspended.

The results of the MLR did not explicitly focus on the data storage location, such as cloud or in-house storage. Participant V and D indicated that they stored their data in-house due to the user's preference and regulations regarding for private data, such as patient records.

Data is stored via uniquely identified data snapshots or via reference to the original raw data set (Derakhshan et al., 2019; Karlaš et al., 2020; Amershi et al., 2019; Ciucu et al., 2019; Miao et al., 2017c). If all snapshots cannot be stored due to space constraints, the delta in the data set can be versioned (Mäkinen, 2021). Because traditional version control systems, such as Git, cannot handle the amount of data (Barrak et al., 2021; Ciucu et al., 2019; Janardhanan, 2020), new systems and tools, such as Data Version Control (DVC) have been introduced. DVC can store large files due to an external storage which can be combined with Git via a lightweight metadata file including a hash to indicate the data set version and data set location. The metadata file is then tracked via Git (Barrak et al., 2021; Sato et al., 2019a; O'Brien, 2021). Several space constraints occur with edge devices where data transfer costs should be avoided. Thus, appropriate locality-aware data stores store the data lineage (Rausch et al., 2019).

In addition, **dependencies**, **data processing steps** and **extracted features** should be versioned. This allows to compute different versions of the data set if required. Dependencies can store the relationship between a data set which was used for training or testing and the ML model version (Lwakatare et al., 2020b; Zhou et al., 2020; Rosenbaum, 2020; Bachinger and Kronberger, 2020). Miao et al. (2017c) and participant T suggest versioning the data processing steps including applied code and metadata (Lwakatare et al., 2020b; Amershi et al., 2019; Stirbu et al., 2021). Extracted features are stored in a centralized repository, such as feature stores, where the definition, access and storage of the features is standardized (Anon, 2021b; Hermann and Del Balso, 2017). For instance, the lifecycle management platforms used by Uber (Michelangelo) (Hermann and Del Balso, 2017), Facebook (FBLearner) (Erb, 2019) or Bighead (Brumbaugh et al., 2019) provide feature stores. This avoids to repeatedly extract feature sets where similar features may have different definitions (Zhou et al., 2020; Brumbaugh et al., 2019; Zweben, 2021).

`Data Documentation`

Documentation includes guidelines with concrete actions such as feature cleaning or naming conventions applicable to data files or folders (Polyzotis et al., 2018; Kronberger et al., 2020; Baroni, 2018). Documentation should be extensive enough to support auditability of AI models. Auditability defines a reviewing process where responsibilities and potential risks associated with the usage of an AI model are identified and root causes can be analysed in case of a failure. Auditability for AI still is an emerging topic that is missing generally established mechanism and regulations (Bourgais and Ibnouhsein, 2021). Although documentation is essential, in practice only one-third of the participants rely on manual data documentation, whereas the others do not document their data related information due to missing guidelines and software support.

4.3.2. Model learning

After data handling, the pipeline executes tasks associated with the AI model learning, such as model design, model training, quality assurance and improvement. Further tasks comprise metadata capturing, versioning of the model and its dependencies, and documentation. According to Zhou et al. (2020), this stage and its respective tasks are most essential throughout the pipeline for the continuous development of AI.

`Design`

Firstly, the pipeline should support taking decisions regarding the model design, such as the appropriate hypothesis (Maskey et al., 2019; Popp, 2019) or regarding the selection of reusable model components which align with the problem domain (Gharibi et al., 2021; Maskey et al., 2019; Wilkiewicz et al., 2019). Pipelines should support context-specific decisions on the best suitable algorithm, feature selection, hyper-parameter setting, data set split, and potential reduction of the training data set (Mäkinen et al., 2021; Amershi et al., 2019; Junsung et al., 2019; Wilkiewicz et al., 2019; Spieker and Gotlieb, 2019). The MLR results did not reveal the concept of Automated Machine Learning (AutoML) within pipelines which takes over design decisions, such as finding an appropriate algorithm, model selection, data selection, and parameter tuning. Participant C illustrated that AutoML may ease the decision making process.

`Model Training`

The pipeline implementation of the model training is dependent on the architecture, distribution and amount of available computation resources (Lopez Garcia et al., 2020; Pölöskei, 2020; Zhou et al., 2020) and context-specific algorithms, such as unsupervised learning, deep learning, and reinforcement learning (Raj et al.; Karlaš et al., 2020; Benbya et al., 2020; John et al., 2020; Rivero et al., 2020; Raj, 2020). Two participants identified that TFX and MLFlow provide the necessary implementations or support for using AI libraries.

`AI Model Quality Assurance`

The pipeline for the continuous development of AI aims to provide faultless, reliable and secure software and to guarantee a trusted decision space (Raj et al.; Lwakatare et al., 2020a; Lopez Garcia et al., 2020). Thus, fail-safe measures need to be introduced via relevant, sufficient and repeatable tests to cover all potential cases due to the continuous pipeline (Fehlmann and Kranich, 2020; Makarov et al., 2021; Fehlmann and Kranich, 2020; Baylor et al., 2019), which is especially important if models' decisions impact human's well-being or can cause harm (Raj et al.; Makarov et al., 2021; Arora et al., 2019; Gorcenski, 2019), as in participant D's case.

Due to these repetitive tests, test data sets are used more often which may lead to **overfitting**. Overfitting occurs when the model's version gets adapted throughout the pipeline cycles to pass the test (Aguilar Melgar et al., 2021; Renggli et al., 2019a; Karlaš et al., 2020; Renggli et al., 2021, 2019b). To test the model for overfitting, three metrics may be used, such as measuring the difference between the validation and test data set (Hubis et al., 2019), the Akaike information criteria or the Bayesian information criteria (Yun et al., 2020).

However, in practice automated tests are not always possible. For instance, participant D indicated that experts had to validate the model's decisions individually. Participant P also stated that they executed quality assurance tests manually due missing configurations in the pipeline but they planed on automating the tests in the future.

In AI the goal is to optimize a specific metric throughout the pipeline's lifecycles instead of simply satisfying functional requirements (Zaharia et al., 2018). For instance, the pipeline compares statistical evaluation metrics with metrics from previous model versions (Nashaat et al., 2019; Zaharia et al., 2018; Santhanam et al., 2019; Bachinger and Kronberger, 2020; Sato, 2020). Participant A and P endorsed to use traditional quality metrics for AI, such as F1 scores, precision and accuracy. Another approach mentioned by participant A is a Normalized Discounted Cumulative Gain for measuring the effectiveness for their search engine model. Gerostathopoulos et al. (2019) proposed a learnability metric specifically adapted for using it during the CI/CD of AI models. Learnability is measured via five SCORE learnability facets, such as solution quality, convergence, overhead, robustness and effect. This SCORE metric was specifically adapted for using it during the CI/CD of AI models (Gerostathopoulos et al., 2019; Olston et al., 2017).

The financial technology sector requires fair models (Huang et al., 2021; Ammanath et al., 2021), thus Huang et al. (2021) established an ethics-by-design metric for the model quality assurance in the continuous development pipeline. This metric uses approaches for feature explainability[2], such as Local Interpretable Model Agnostic Explanations (LIME), Partial Dependency Plots (PDP), and SHapley Additive exPlanations (SHAP), to identify whether and how much each feature contributes to the final prediction (Bourgais and Ibnouhsein, 2021; Yun et al., 2020; Sato et al., 2019a). In practice, interviewee R, A and B stated that they did not include checks for bias in the pipeline's model quality assurance due to their non-critical domains.

`Model Improvement`

The extracted sources from the MLR do not identify any new pipeline-specific model improvement techniques that specifically focus on previous lifecycles. Pipeline implementations use established context and model specific improvement techniques. For instance, pipelines support model compression or pruning (Hummer et al., 2019; Karlaš et al., 2018; Boovaraghavan et al., 2021), model hardening (Gharibi et al., 2021; de la Rúa Martínez, 2020; Gupta and Galinkin, 2020b), and hyper-parameter optimization (Spell et al., 2017; John et al., 2020; Boovaraghavan et al., 2021; Janardhanan, 2020; Duvall, 2018).

`Metadata Capture`

Metadata comprises provenance information which is necessary to govern the modelling lifecycle as well as to receive additional information on the data set and AI model (Miao et al., 2017a; Gharibi et al., 2021; Bachinger and Kronberger, 2020). For example, model specific metadata comprises relevant information on the execution and deployment of AI models, network architecture (Chard et al., 2019), training (Lwakatare et al., 2020a,b; Chard et al., 2019; Vartak et al., 2016), instance metadata and

[2] For further information please refer to the book 'Interpretable Machine Learning', Section 5: https://christophm.github.io/interpretable-ml-book/lime.html, accessed 05.07.2021

runtime metadata (Liu and Brooks, 2020; Hummer et al., 2019; Zhang et al., 2020; Chard et al., 2019). Further metadata includes information about the model's location, registration data, and information on the start and end data (Lwakatare et al., 2020a,b; Martel et al.; Raj et al.; Lopez Garcia et al., 2020; Brumbaugh et al., 2019). Feature extractors provided by several lifecycle management tools, such as TFX, automatically extract metadata (Gharibi et al., 2021; Wilkiewicz et al., 2019).

`Model Versioning`

Model versioning not only captures version model artefacts but also model dependencies to backtrack or reproduce different model versions that quickly evolve over time (Vartak et al., 2016; Miao et al., 2017a,b,c; Li et al., 2021; Baroni, 2018; Peili et al., 2018; Anon, 2021b; Garcia et al., 2018). Although literature highlights the importance of model versioning, several aspects why this is not possible were mentioned by three participants (A, P and B). They indicated that they did not store all models but only stored key model versions due to resource restraints or lack of interest in previous versions. In addition, Granlund et al. (2021) stated that they cannot store model versions due to strict guidelines regarding the patient's data and missing isolated, in-house hardware resources. The inability to store model versions was not solved by any of the extracted pipeline tasks.

Model dependencies capture the relationship to related elements, such as the associated data set, source code and configuration files. This allows to recreate or load a specific model without having to rerun multiple iterations to find the appropriate model (Janardhanan, 2020; Lwakatare et al., 2020a; Miao et al., 2017b,c; Mäkinen, 2021). Additionally, model versioning stores the associated log files (Meynard and Machado, 2021) and evaluation results of a model. This allows to check whether the model versions improve continuously throughout the continuous lifecycle (Lwakatare et al., 2020a; Makarov et al., 2021; Karlaš et al., 2020; Vartak et al., 2016).

Because AI model versioning is more complex and requires more storage capacities due to the continuous development, standard version control systems, such as Git cannot be used as model repositories (Breuel, 2020). Potential alternatives are container registries where images are versioned (Singhal and Kumar, 2020) or model repositories that store model versions including code, metadata, test results and dependencies (Chard et al., 2019; Stirbu et al., 2021; Moesta and Tamisin, 2020). During the interviews, participants proposed storage facilities of MLFlow, H2O, DataRobot and Git Large File Storage.

`Model Documentation`

Documentation proves that the model adheres to the regulations and restrictions and works as expected. This is required to receive certifications, which is especially important in the healthcare sector according to participant D. However, seven out of nine participants identified model documentation as a good practice however they hardly ever create and maintain an up-to-date and consistent documentation. Based on the MLR results, software support for the model documentation during the pipeline is not established yet.

The MLR however, already established the information necessary in the model documentation. For instance the documentation should outline the purpose (e.g., requirements and hypothesis) and the different methodologies to achieve the set purpose (e.g., technical decisions, such as chosen model and algorithm design) to effectively mitigate AI related risks (Stirbu et al., 2021; Haakman et al., 2020). Commonly, decisions made in the model creation need some rework. Thus, it is essential to document which model and test design decisions have already been tried out and which results were achieved (Garcia et al., 2018; Rivero et al., 2020). Additionally, documented assumptions explain the reached decisions (Haakman et al., 2020).

4.3.3. Software development

After the model is developed, it must be prepared for deployment. Therefore, the pipeline orchestrates the stage *Software Development* and its related tasks, such as packaging, system-level quality assurance as well as system versioning.

`Package` The build process packages the code and model logic into build artefacts which are deployed in production (Vuppalapati et al., 2020; Zhang et al., 2020; Liu et al., 2019; Liu and Brooks, 2020). The best model is selected and registered in the model registry (Lwakatare et al., 2020a).

Several authors propose transforming the registered model into a system-independent, deployable and hardware-optimized format (Makarov et al., 2021; Zhang et al., 2020; Castellanos et al., 2021). Potential formats established in the pipeline packaging task are ONNX, SavedModel, TensorRT, Predictive Model Markup Language (PMML), and Portable Format for Analytics (PFA). ONNX[3] is an open, standardized format built that represents AI models by using a common set of operators and file format to allow interoperability and serialization of models (Raj, 2020; Visengeriyeva et al., 2021). When the pipeline handles Tensorflow models, it is transformed to SavedModel and TensorRT format (Makarov et al., 2021; Zhang et al., 2020). Another possible format is PMML,[4] which describes statistical and data mining formats and transforms them into an Extensible Markup Language (XML) configuration file (Castellanos et al., 2021; Zaharia et al., 2018; Visengeriyeva et al., 2021). The PFA,[5] is a model interchange format that provides a safe execution environment for AI algorithms (Visengeriyeva et al., 2021).

An essential task in this stage is to containerizes the model and its dependencies to avoid compatibility issues (Chard et al., 2019; Corbeil and Daudens, 2020; Lopez Garcia et al., 2020; Liu et al., 2019; Castellanos et al., 2021; de la Rúa Martínez, 2020; Baroni, 2018; Bachinger and Kronberger, 2020). Extracted sources most often mention Docker (Corbeil and Daudens, 2020; Lopez Garcia et al., 2020; Liu et al., 2019; Castellanos et al., 2021; Pölöskei, 2020; Ciucu et al., 2019). In addition, three interviewees (participant R, C and V) confirm using Docker extensively. Research on other approaches is sparse, however essential for increasing efficiency in cloud computing (de la Rúa Martínez, 2020). Thus, future research suggests using Virtual Machine (VM)s, especially lightweight VMs, type 1 hypervisors, or uni kernels also called library operating systems (de la Rúa Martínez, 2020).

`Software-level Quality Assurance`

According to the paper 'Hidden Technical Debt in Machine Learning Systems' (Sculley et al., 2015), AI models represent only a small part of the whole system landscape where many data sources and software applications interact (Baylor et al., 2017; Huang et al., 2021; Granlund et al., 2021; John et al., 2020). Software-level quality assurance should identify the correct behaviour of the whole system landscape before the system is deployed in production (Sato et al., 2019b; Sato, 2020; Guo et al., 2020). Participants D and R agreed on the importance of this type of quality assurance whereas participant C expressed some concerns because the integration and respective testing are not model related and should be handled via a different CI/CD pipeline.

The continuity of the pipeline requires automated software-level quality assurance tests to frequently execute the tests and compare the results. For instance, integration tests check if different services work together correctly (Pentreath, 2019), or check

[3] For information about ONNX, please refer to https://onnx.ai/, accessed 16.07.2021

[4] For information about PMML, please refer to: http://dmg.org/pmml/v4-4-1/GeneralStructure.html, accessed 19.07.2021

[5] For information about PFA please refer to: Pivarski et al. (2016)

whether the obtained model prediction is correctly transferred to the whole system by comparing this test to results of the model quality assurance (Lwakatare et al., 2020a; Meynard and Machado, 2021; Kent and Doran, 2019). The pipeline can automatically execute compatibility checks between interfaces and the Application Programming Interface (API) endpoints (Anon, 0000; Sato et al., 2019a), which is essential according to participant R when AI models are integrated as microservices. Automated stress and robustness tests identify whether the whole software can perform under expected conditions (Rausch and Dustdar, 2019; Lavin et al., 2021) by evaluating operational metrics such as throughput, latency, and resource usages (Sato, 2020). For example, Uber's end-to-end lifecycle management platform Michelangelo uses an internal benchmarking system to profile certain software parts. This allows measuring how quickly inferences are run for a specific model based on real-life data (Guo et al., 2020).

However, extracted sources reveal shortcomings in the automation of available test strategies to efficiently use them during the pipeline. For instance, because user acceptance tests require human involvement (Pentreath, 2019; Fehlmann and Kranich, 2020; Gorcenski, 2020), Fehlmann and Kranich (2020) tries to solve it via Quality Function Deployment (QFD). This strategy collects customers' expectations and needs and generates a test coverage matrix where a support vector machine generates test cases (Fehlmann and Kranich, 2020).

`Versioning`

Packaged models and the respective quality assurance results of the software level are stored (Granlund et al., 2021; Baroni, 2018; Guo et al., 2020). In addition, it is also necessary to version the pipeline and its associated tasks (Hummer et al., 2019; Brumbaugh et al., 2019; Stirbu et al., 2021). For instance, TFX versions the pipeline as an artefact or source code regarding the implemented pipeline tasks (Anon, 2021a; Sato et al., 2019b) Additionally, ModelDB stores the pipeline as a sequence of actions in a relational database (Vartak et al., 2016). GitHub repositories may store an Azure DevOps pipeline (Rosenbaum, 2020). Moreover, the used pipeline version references the associated model versions or vice versa (Brumbaugh et al., 2019).

`Documentation`

Documenting information about the development stage was not handled by any paper from the literature review but was mentioned by participant T. He proposed to document the quality assurance process and associated outcome and any additional information necessary for a software release. The participant raised the research gap on software tools automatically handling and updating the documentation.

4.3.4. System operations

The proposed framework's stage *System Operations* handles the deployment of the AI model into the system landscape and handles the continuous monitoring of the data, model, and system.

`Deploy`

The deployment is an essential task to make the model available to others, enable collaboration and avoid knowledge silos (Lopez Garcia et al., 2020; Tamburri, 2020). But before deploying the model, the first three out of four criteria need to be fulfilled.

1. All preceding pipeline tasks are executed successfully, such as the quality assurance test suite (Zhou et al., 2020; Lwakatare et al., 2020a; Aguilar Melgar et al., 2021; John et al., 2020).
2. The pipeline identifies the best model that is not necessarily the newly trained model (Raj et al.; Pölöskei, 2020; Olston et al., 2017; Karlaš et al., 2018). However, the research gap arises how to compare model versions fairly and without any bias. Not only does the pipeline need to be aware of whether the validation uses the same data but also if the evaluation remains the same (Garcia et al., 2018; Karlaš et al., 2018; Schelter et al., 2018). One approach to guarantee comparability is to use an unseen dataset (Aguilar Melgar et al., 2021).
3. The model fulfils user-defined deployment criteria, such as a specified increase in accuracy (Aguilar Melgar et al., 2021; Rausch et al., 2019) or, according to participant P, a specific benchmark.
4. Some pipelines may require human involvement, such as manual validation of tests, before deploying the AI model to production (Rosenbaum, 2020; Aronchick et al., 2020; Schruhl and Windheuser, 2020). This is also essential in participant P's case.

The model is deployed on different environments, such as cloud-managed serving platforms, server disks or remote storage, that vary in their infrastructure and strongly depend on the anticipated traffic and financial resources (Schruhl and Windheuser, 2020; Vadavalasa, 2020; Srinivasan, 2021; Mäkinen, 2021; Spell et al., 2017; Díaz-de Arcaya et al., 2020). Cloud-managed serving platforms have the advantage that they can scale to accommodate the need for high-performance computing, low-latency, and memory-intensive requirements (Arora et al., 2019; Castellanos et al., 2021). The interview participants also prefer cloud solutions.

`Deployment Strategy`

Continuous experimentation is a deployment strategy that allows gathering (user) feedback during runtime (Corbeil and Daudens, 2020; Banerjee et al., 2020; Baylor et al., 2017). Examples are A/B tests (Corbeil and Daudens, 2020; Singhal and Kumar, 2020), canary releases (Baylor et al., 2017; Olston et al., 2017) and shadow deployments (Saucedo, 2020; Windheuser and Sato, 2020; Sato et al., 2019a; Mäkinen, 2021; de la Rúa Martínez, 2020; Ettun, 2019). Participant A uses these deployment strategies to identify potential network effects. If the AI model is used for critical decision-making in the medical sector, deployment strategies which do not influence the system behaviour, clinical performance and safety of the patients should be used. If issues are encountered, rollbacks to already packaged models are necessary as identified during the interviews (R, A and T).

`Monitoring`

The pipeline also monitors the AI model in production and collects necessary information to improve the non-deterministic model over time (Stirbu et al., 2021; Anon, 0000; Leff and Lim, 2021; Wilkiewicz et al., 2019; Gorcenski, 2020). Therefore, the monitoring results should be systematically mapped to the different model versions (Schreiber et al., 2014).

The MLR results reveal that the monitoring can be split up into four aspects. Firstly, the monitoring systems collect input and output data to use for future training (Miao et al., 2017a; Lavin et al., 2021; Breck et al., 2019; Polyzotis et al., 2018; Bachinger and Kronberger, 2020; Kronberger et al., 2020; de la Rúa Martínez, 2020; Raj et al.; Yun et al., 2020; Anon, 2021b). In addition, this helps to quickly identify a drift in the data set due to changes in the statistical characteristics of distribution (Bachinger and Kronberger, 2020; Kronberger et al., 2020; Yun et al., 2020; de la Rúa Martínez, 2020). Secondly, monitoring tools observe the model performance to identify if the performance deteriorates when using real-life data (Lwakatare et al., 2020a; Corbeil and Daudens, 2020; Visengeriyeva et al., 2021). Thirdly, it collects

traditional software monitoring aspects, also called Key Performance Indicator (KPI)s during runtime (Lwakatare et al., 2020a; Corbeil and Daudens, 2020; Castellanos et al., 2021). For instance, KPIs consist of response time, minimal latency and throughput (Lwakatare et al., 2020a; Li et al., 2021; Pentreath, 2019) or resource usage and network statistics, which are especially important if the AI application is deployed as a Service (Zhang et al., 2020; Baroni, 2018). Fourthly, may be collected to identify the impact on business outcomes, such as user engagement (Baroni, 2018; Pentreath, 2019; Sato et al., 2019a). According to participant P, operational telemetry data is essential because although the model improves, the subjective user perception of the model may diminish. To monitor the previously mentioned aspects, participants A, P and T use Google Error Logs and Postman to identify the correct behaviour and performance of API requests.

`Environment and Infrastructure Handling`

AI applications may be deployed in several different environments or operational stages. The environments have varying hardware, operating system and software version dependencies (Hummer et al., 2019; Brumbaugh et al., 2019). Although the model operates on different environments, it must provide persistent output for a specific input (Stumpf et al., 2018). If for instance, multiple computing platforms use the same AI application, the pipeline should support cross-platform abstraction. This allows abstracting the configuration management and low-level API calls. For example, SageMaker expects a custom Docker image including predefined entry points, Watson Machine Learning requires a zip archive which includes the code for training a model. This zip archive is then deployed in a container in the cloud (Hummer et al., 2019).

4.4. Challenges

This section elaborates 25 challenges focusing on the implementation, adaption and usage of pipelines for the continuous development of AI. Fig. 5 maps all identified challenges to general requirements or the presented framework's four stages, *Data Handling*, *Model Learning*, *Software Development*, and *System Operations*. Fig. 5 uses different colours for each challenge to indicate how often the sources state the challenge. For instance, yellow indicates that we extracted the challenge less or equal to 5 times, whereas dark blue indicates that we identified the challenge between 26 and 30 times. These challenges are further summarized in Table 7. This following section also describes challenges that we substracted frequently in more detail.

4.4.1. Data handling

Challenges during the stage *Data Handling* elaborate challenges regarding data collection and integration, preparation, data quality assurance with reasonable and sound alerts and data versioning with the respective data traceability. The most often mentioned data stage issue is the **data collection and integration** with over 16 mentioned cases. The pipeline is challenged with complying with local data regulations (Banerjee et al., 2020). For example, the healthcare sector may require to keep the data sets within the organization. Thus, the pipeline should be able to work with restricted on-premise computation resources (Granlund et al., 2021). In addition, participants R and D stated that the pipeline should handle data governance and ownership constraints.

In addition, the pipeline should automatically transform data from various schema regimes into a universal format. Additionally, the pipeline should store the data transformation and data preparation steps in a machine-readable form (Amershi et al., 2019; Renggli et al., 2021). Moreover, the pipeline faces the issue of automatically encoding data into features with the help of self-labelling consistent, and accurate instrumentation (Polyzotis et al., 2017; Lwakatare et al., 2019).

4.4.2. Model learning

Specific challenges occurring during the stage *Model Learning* elaborate challenges regarding finding the optimal time to update a model to save time and resources, model quality assurance faces the challenges to split the data set, test updated and the potential overfitting and the introduced latency that slows down the pipeline. In addition, most often mentioned was that **versioning** should depict the model evolution and ensure reproducibility (Saucedo, 2020; Srinivasan, 2021). Versioning is a static procedure, however, the pipeline needs to be capable of tracking constantly learning models and the conceptual coupling between the respective data sets, model and pipeline (Sato et al., 2019a; Barrak et al., 2021; Bachinger and Kronberger, 2020; Barrak et al., 2021; Sato et al., 2019b). Additionally, the pipeline's versioning should provide enough information to identify the reason for performance and prediction changes (Yun et al., 2020).

4.4.3. Software development

Specific challenges occurring during the stage *Software Development* elaborate challenges regarding packaging model and software independently to provide scalability which was challenging for participant P. The necessary tracking provenance becomes challenging if a complex model combines several other models which were trained on different data sets. It becomes even more challenging if the training data set's transformations are highly-heterogeneous (Lwakatare et al., 2019). The pipeline should also allow for backwards and forwards compatibility of a data schema (Sato et al., 2019a; Schelter et al., 2018).

4.4.4. System operations

Specific challenges occurring during the stage *System Operations* elaborate challenges regarding the deployment, such as ensuring that the model is compatible with the consuming application, cross-model inference and comparing models. Monitoring related challenges focus on what aspects are worth monitoring and hidden feedback loops. The challenge during the **environment and infrastructure** handling emphasizes the issue of handling multiple environments and embedded systems which was identified in 12 sources. This challenge occurs if AI applications are deployed to the cloud, such as multiple on-premise servers, edge devices, dedicated clusters or any combination (Hummer et al., 2019; Martínez-Fernández et al., 2021; Anon, 0000). Because these environments handle resources and security differently, the pipeline needs to adapt to these specific requirements (Hummer et al., 2019; Martínez-Fernández et al., 2021; Aronchick, 2020). For instance, edge devices provide different hardware architecture, such as arm64X or x86 or sensor setups, thus the pipeline needs to provide personalized docker containers automatically tailored to the architecture. This should allow improving hardware stability (Raj, 2020).

4.4.5. General requirements for the pipeline

General challenges which cannot be mapped to a specific stage of the framework focus on the need of a flexible, customizable, reusable and fault tolerant pipeline. Elastic scaling is also a pipeline specific challenge because capacity changes. The required multi-tenant setting, integration into the already existing security landscape, and missing regulations pose challenges as well.

The pipeline has to be **flexible** and **customizable** to avoid becoming too restrictive (Seyffarth, 2019) and to allow using preferred and established tools, services and engineering practices (Hummer et al., 2019; Renggli et al., 2019a; Spell et al., 2017). Thus, pipelines need to combine quickly evolving tools, libraries, frameworks and platforms in every step of the AI lifecycle management to meet specific requirements (Martel et al.;

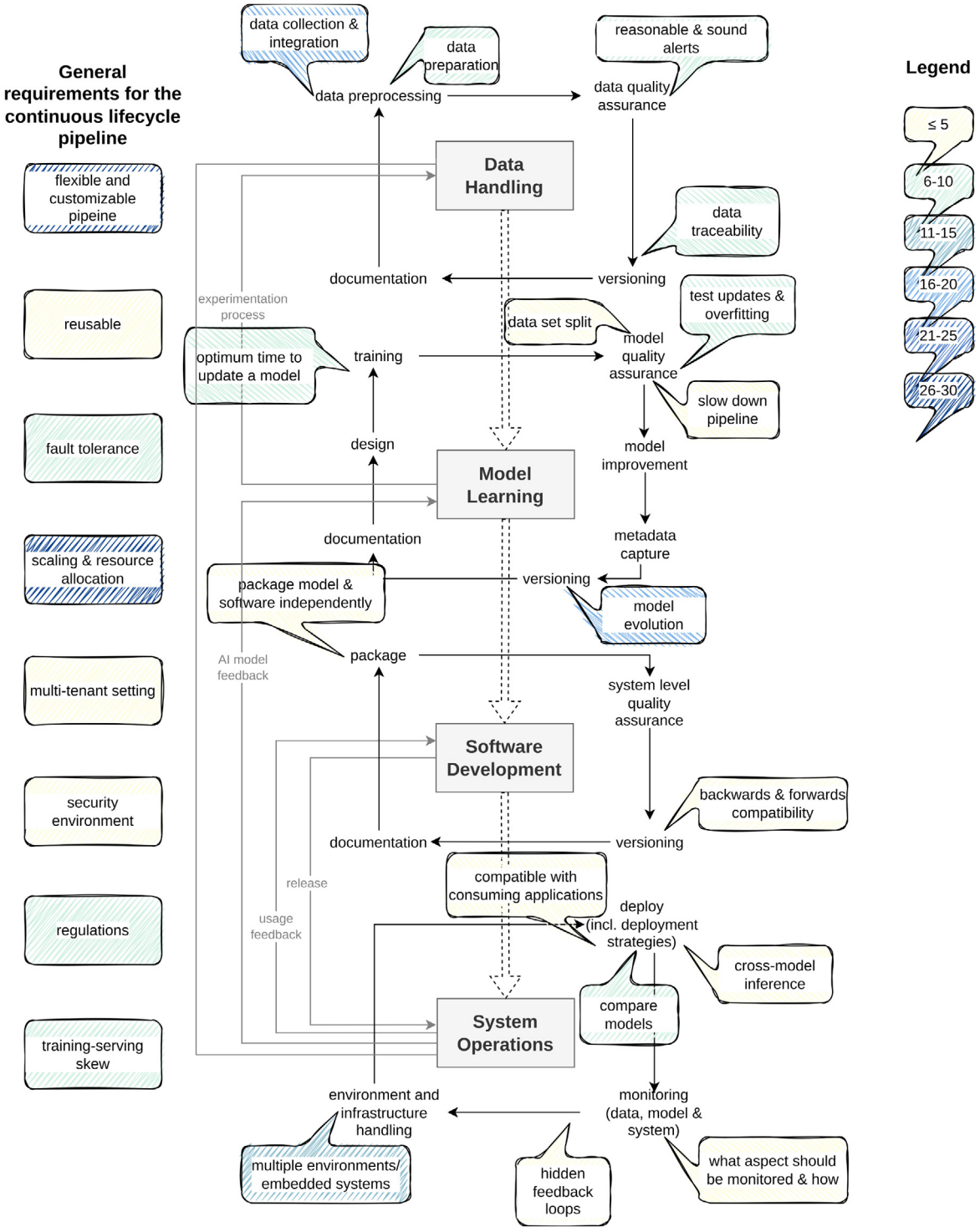

Fig. 5. Challenges occurring during the implementation, adaption and usage of pipelines for the continuous development of AI - labelled challenges based on the number of occurrences starting from yellow (≤ 5) to dark blue (26–30) also indicated in Table 7. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Aronchick et al., 2020; Ammanath et al., 2021; Spell et al., 2017; Castanyer et al., 2021). The need for flexibility and customizability were confirmed by more than half of the participants (R, A, P, Z and C) because pipelines are very context-specific and strongly depend on the use case. Participant V uses TFX for their AI development and they discovered that the infrastructure handling is very difficult for them and for their customers. For example, TFX requires Kubeflow which only works on Linux. Participant V, indicated that his customers wanted to execute the TFX pipeline by themselves but did not have an infrastructure based on Linux.

The demand for computing capacities varies heavily due to the increase in the algorithms' unpredictable complexity (Martel et al.; Brumbaugh et al., 2019). Thus, elastic **scaling** is required to provide the right amount of hardware to handle capacity changes (Martel et al.; Brumbaugh et al., 2019). The pipeline needs to adapt to either horizontal or vertical scaling which is

Table 7
Overview of challenges regarding the implementation, adaption and usage of pipelines for the continuous development of AI.

Stage	Challenge	# Occurances	Description
Data Handling	Data collection & integration	16 – 20	– comply with local data regulations (Banerjee et al., 2020; Granlund et al., 2021) – transform data from various schema regimes into universal format Amershi et al. (2019), Renggli et al. (2021) – automatic feature labelling (Polyzotis et al., 2017; Lwakatare et al., 2019) – how to automatically identify which data source augments features (Polyzotis et al., 2017)
	Data preparation	6 – 10	– pipeline should identify how data preparation effects model quality (Renggli et al., 2021) – cleaning raw data, the pipeline struggles to identify which effect noise or uncertainty have on the model quality (Renggli et al., 2021)
	reasonable & sound alerts	6 – 10	– unclear data anomaly alerts (Baylor et al., 2017; Polyzotis et al., 2017) – correct & reasonably strict alerts (Baylor et al., 2017; Polyzotis et al., 2017) – unclear what optimal amount of alerts is (Baylor et al., 2017) – flexible & automated data validation techniques to avoid unreasonable customization but allow for custom checks (Baylor et al., 2017)
	Data traceability	6 – 10	– automatic versioning methods for data sets and associated model (Rosenbaum, 2020) – store information on data governance (Sato et al., 2019a)
Model Learning	Optimum time to update a model	6 – 10	– what are signs for optimal retraining time (Mäkinen et al., 2021; Mäkinen, 2021; Banerjee et al., 2020) – pipeline ought to handle conceptual coupling between data, model and pipeline (Barrak et al., 2021) – how can it handle online model retraining (pipeline is not triggered) & model is not static artefact (Sato et al., 2019a)
	Data set split	≤ 5	– how to distribute data to training, validation and test data set based on pipeline's knowledge (Schelter et al., 2018)
	test updates & overfitting	6 – 10	– test automation of self-adaptive models throughout lifecycle (Felderer and Ramler, 2021) – how can the pipeline efficiently reuse tests and data set without overfitting the model (Schelter et al., 2018; Garcia et al., 2018; Renggli et al., 2021)
	Slow down pipeline	≤ 5	– how should pipeline handle latency due to data preparation (Raj et al., 2020)
	Model evolution	16 – 20	– how can various versions of AI models and all the dependencies be stored efficiently (Lwakatare et al., 2020a,b; Miao et al., 2017b) – research identified that frequent model updates also require corresponding changes to various software artefacts (e.g. build files). A challenge for the pipeline is to reduce this overhead (Barrak et al., 2021)
Software Development	Package model & software independently	≤ 5	– handle complex models that includes other models (Lwakatare et al., 2019)
	Backwards & forwards compatibility	≤ 5	– how to ensure compatibility of a data schema (Sato et al., 2019a; Schelter et al., 2018)

(*continued on next page*)

strongly dependent on the infrastructure and platform limitations (de la Rúa Martínez, 2020). In addition, the pipeline needs to be capable of efficiently distributing these resources. This is especially important if device capabilities are heterogeneous such as differences in the Central Processing Unit (CPU) complexity, memory, number of cores or bandwidth (Boovaraghavan et al., 2021; Sato et al., 2019b).

5. Discussion

The following section provides a discussion on how to map an existing pipeline for the continuous development of AI to the proposed framework. Whereas the proposed framework provides a thorough depiction of tasks in the lifecycle management pipeline, the implemented pipeline may vary in practice. For instance, pipelines may vary due to resource constraints, difficulties in specific implementations and because a task's benefit does not outweigh the costs. In addition, the execution order of the tasks

Table 7 (*continued*).

Stage	Challenge	# Occurances	Description
System Operations	Compatible with consuming applications	≤ 5	– the pipeline needs to support non-standardized integration strategies for multiple consuming application (e.g. library dependency, separate service, model as data) (Zaharia et al., 2018; John et al., 2020; Raj, 2020)
	Cross-model inference	≤ 5	– How can the pipeline anticipate & prevent cross-model inferences (e.g. latency) (Baylor et al., 2017; Olston et al., 2017)
	Compare models	$6-10$	– How to identify optimum model for deployment based on stored information (when e.g. information regarding model's behaviour with non predictable production stream is not available) (Mäkinen et al., 2021; Miao et al., 2017b,c; Hermann and Del Balso, 2017; Renggli et al., 2021)
	What aspects should be monitored & how	≤ 5	– Which information helps to improve the pipeline (Gharibi et al., 2021) – How should the pipeline most efficiently integrate human interaction (Granlund et al., 2021)
	Hidden feedback loops	≤ 5	– How can pipeline prohibit propagating wrong training data (e.g. bias) (Amershi et al., 2019; Breck et al., 2019; Schleier-Smith, 2015)
	Multiple environments/embedded Systems	$11-15$	– How can pipeline adapt to specific requirements (e.g. resources, security) of varying environments (Hummer et al., 2019; Martínez-Fernández et al., 2021; Anon, 0000; Aronchick, 2020)
	Flexible & customizable pipeline	$26-30$	– How can the pipeline cater to custom selected tools, services, engineering practices, libraries, frameworks and platforms in every step of the AI lifecycle (e.g. data processing, quality control) (Hummer et al., 2019; Renggli et al., 2019a; Spell et al., 2017; Martel et al.; Aronchick et al., 2020; Ammanath et al., 2021; Castanyer et al., 2021)
	Reusable	≤ 5	– Pipeline ought to provide predefined templates and patterns to minimize configuration effort while maintaining flexibility (Brumbaugh et al., 2019; Hummer et al., 2019) (Participant R)
	Fault tolerance	$6-10$	– How can pipeline handle failures in its underlying execution environment due to different tools and infrastructures plugged together without interruption (Hummer et al., 2019; Baylor et al., 2017; Arora et al., 2019) – How can the pipeline recover from intermittent failures (e.g. inconsistent data) (Boag et al., 2017)
	Scaling & resource allocation	$26-30$	– Pipeline needs to automatically adopt & efficiently distribute resources due to the model's unpredictable complexity (Martel et al.; Brumbaugh et al., 2019,?; de la Rúa Martínez, 2020; Boovaraghavan et al., 2021; Sato et al., 2019b) (participant C) – How can the pipeline optimize its different tasks (e.g. data processing could be optimized via declarative abstraction) (Schelter et al., 2018)
	Multi-tenant setting	≤ 5	– How can the pipeline prioritize different users all triggering the same pipeline (Aguilar Melgar et al., 2021; Karlaš et al., 2018) – How should the pipeline allocate resources to the tenants in a non-discriminatory manner (Karlaš et al., 2018)
	Security environment	≤ 5	– How can the company-specific security environment integrate the continuous pipeline (participant Z)
	Regulations	$6-10$	– How can tasks throughout the pipeline support country-specific regulations (e.g. privacy, necessary documentation, GDPR, certifications) (participant A,T,P,R,B,V,D)
	Training–serving skew	$6-10$	– Pipeline should orchestrate data processing regarding training and inference data and its specific characteristics (Baylor et al., 2017; Olston et al., 2017; Polyzotis et al., 2017; John et al., 2020)

strongly depends on the specific context, such as organizational policies for running the pipeline.

The most often mentioned pipeline collected via the MLR is TFX. TFX combines different components to enable a flexible

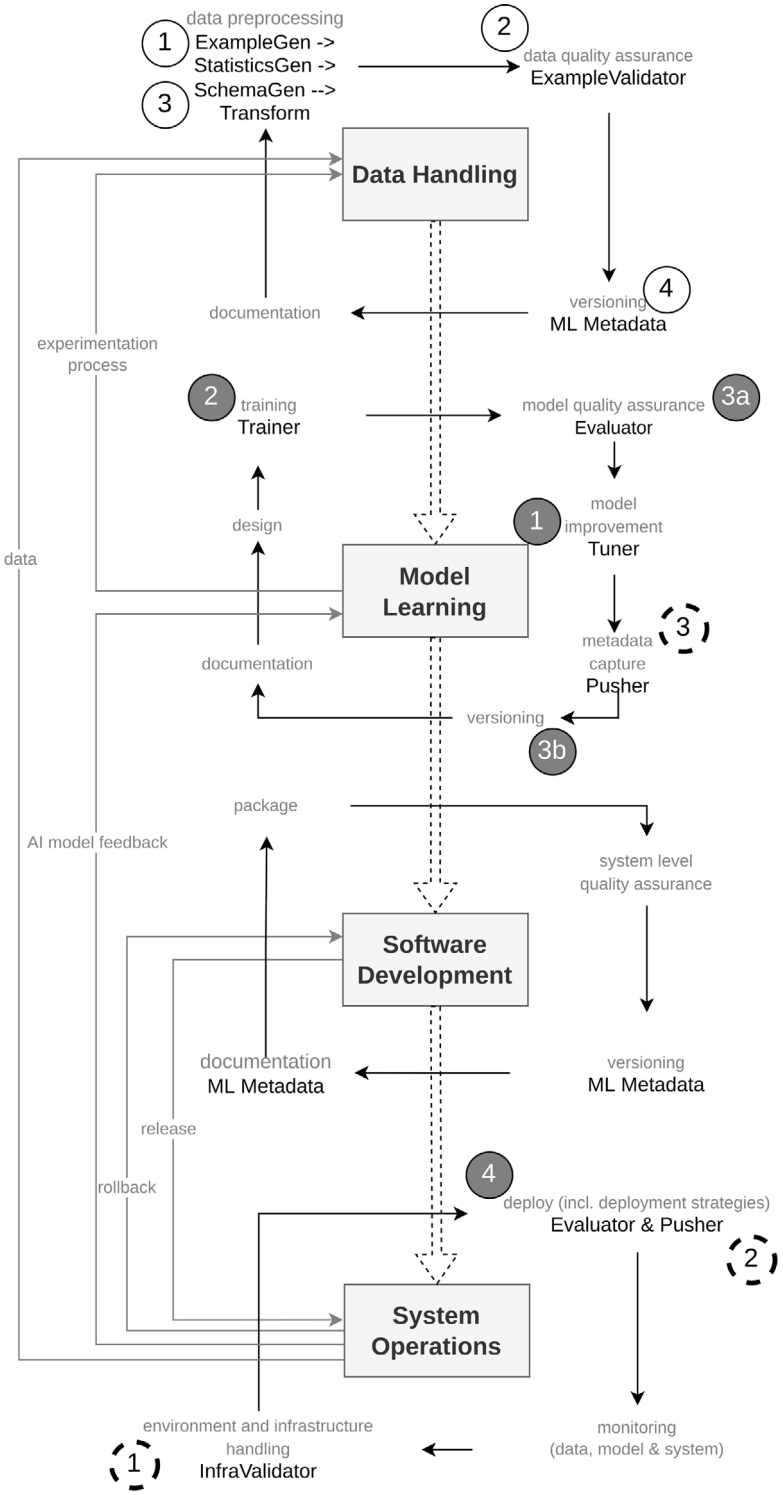

Fig. 6. Map TFX's components to the proposed framework's stages and tasks; The circle with the number illustrates the execution order in TFX for each stage, where the circle without filling belongs to our identified stage *Data Handling*, the filled circle belongs to the stage *Model Learning* and the circle with dotted line belongs to our stage *System Operations*.

and customizable pipeline. An orchestrator, such as Kubeflow, manages and triggers these components. In the proposed framework, the term *tasks* used in this paper refer to the same concept as TFX's components. Fig. 6 illustrates the TFX's components mapped to the framework. Components for the data preprocessing consist of ExampleGen, StatisticsGen, SchemaGen, and Transform, which are explained in the next subsection.

5.1. Data handling

TFX's ExampleGen component stores consumed data from different file types (e.g., csv) in an appropriate format for the following components. With the help of Apache Beam, a unified programming model for defining and executing data processing workflow, ExampleGen can read different data formats and sources. For instance, ExampleGen handles file-based data

(e.g. CSV, Amazon S3, Google Cloud Storage, general text files), Messaging (e.g. Apache Kafka, Kinesis streams, JMS, Amazon Simple Queue Service), FileSystem-based data (e.g. Hadoop, Google Cloud Storage), Databases (e.g. Apache Cassandra, MongoDB, Google Cloud Datastore) and others. TFX's ExampleGen transforms the ingested data to customizable spans, versions and splits. A span groups together data based on context-specific characteristics, such as a day. Spans include several versions when the data is processed, resulting in a new version within the span. ExampleGen then splits each version into training and evaluation data sets.

StatisticsGen uses ExampleGen's output as input and calculates descriptive statistics for the data set.

SchemaGen uses these statistics to construct a schema automatically. The schema includes information about the (absence/presence of) data types, ranges, categories, distribution etc. Developers can modify and adapt the generated schema.

ExampleValidator uses the generated schema as input to validate the data set via the TensorFlow Data Validation to ensure data quality. As suggested in the MLR, it compares the data statistics against the constructed schema, compares training and serving data to identify training–serving skews, and identifies data drifts by evaluating a series of data.

Opposed to our proposed framework, TFX's Transform component handles feature engineering after the data validation. Transform converts the ExampleGen's data set via SchemaGen's data schema using TensorFlow Transform. Feature transformations include embedding (mapping features to a low dimensional space), vocabulary generation (convert non-numeric features into integers), value normalization (scale numeric data without distorting differences in its range), and enriching text feature (extract features from raw data, e.g. n-grams).

5.2. Model learning

Opposed to our proposed framework, TFX firstly improves the model via hyperparameter tuning, then trains the model and then validates it. Regarding model learning, TFX improves the model by tuning hyperparameters via their Tuner that uses the Python KerasTuner. As input, Tuner requires the training and evaluation data set, tuning logic such as the model definition, hyperparameter search space and the Protocol buffers that include instructions for serializing structured data.

Trainer trains an AI model via Python's TensorFlow API. This task uses ExampleGen's data set and the transformed features to train two models. One model gets deployed to production for inference, and TFX uses one model for evaluation. TFX also allows warm-starting the training by using an existing model for further training. In addition, during model training, developers can simultaneously compare multiple model runs, due to saved information in the ML Metadata Store.

TFX's Evaluator performs analysis on the training results and allows to look at the model's behaviour for individual slices of data sets. This ensures identifying well-performing models for the entire data set but poorly for a data point. This task also allows comparing models to find the optimum model that does not necessarily need to be the lastly trained model. Moreover, standard Keras metrics calculate the accuracy, precision, recall etc.

5.3. Software development

TFX does not explicitly handle any software development-specific tasks in their primary pipeline, such as packaging or system-level quality assurance. ML Metadata versions and documents software-specific information.

5.4. System operations

TFX'S InfraValidator checks whether the model complies with the production environment. To do so, it launches a sand-boxed model server based on manually configured environment details, such as type of CPU, memory, or accelerators to evaluate the compatibility between the model server binary and the model that should get deployed.

After successfully evaluating the model concerning the specific production serving environment, TFX's Pusher pushes the validated model to the appropriate deployment environment. Depending on the deployment target Pusher supports model repositories such as Tensorflow Hub, Javascript environments (tensorflow.js), native mobile applications (TensorFlow Light), or server farms (Tensorflow Serving).

TFX versions all the artefacts produced by every component over several executions. Therefore, it uses a ML Metadata Store, a relational database that stores the properties of trained models and the respective data set, evaluation results, execution records and the provenance of data objects. Thus, the metadata store allows identifying which features impacted the evaluation metrics. It also allows only rerunning necessary components, such as skipping data processing when the developer only adapts the model hyperparameters.

6. Threats to validity

This section discusses the four possible main threats to validity according to Wohlin et al. (2012) as well as how we mitigated these threats. In addition, the section discusses the scope of this study.

Conclusion validity is restricted because we chose a qualitative approach. To improve the interviews' reliability of treatment implementation, we implemented and tested an interview plan. In addition, the semi-structured approach allowed us to ask follow-up and clarification questions to reduce misunderstandings and ensure a thorough understanding.

Internal validity may be weak because only one researcher conducted the MLR. Therefore, as proposed by Kitchenham and Charters (2007), we execute a test–retest approach at the end of the initial source selection to provide internal consistency in the decisions to include or exclude the source as well as information extraction. The random sample comprised 30 papers where only two papers were excluded instead of included. This results in a consistency of 93,3%. We calculate the test–retests value for categorizing the data of the included sources. Therefore, we once more extracted the information from these 30 randomly selected sources. The average difference in the information extraction comprises 23%. This value may seem high, however, some categories only had one supporting source during the test, and during the retest, we identified another source which is a 100% increase from test to retest.

Regarding the **construct validity**, one may argue that the literature review is an interpretation of the meaning of the collected literature. Thus, to minimize construct validity, the selected literature resources were carefully evaluated via two quality checklists proposed by Kitchenham and Charters (2007) for formal literature and Garousi et al.'s checklist (Garousi et al., 2016) for grey literature. In addition, we avoided construct irrelevance or construct under-representation as described by Messick (1995) and applied to literature studies by Dellinger (2005) via six countermeasures. (1) The selection process was rigorously documented, (2) the studies were assigned a 5-point Likert scale to identify the amount of contribution. (3) Over-representation was avoided by combining similar sources from the same authors and companies. (4) The described checklists identified unwarranted

sources which were excluded. (5) Contrary findings were also included as well as (6) the search strings allowed to include many relevant study findings.

Regarding the **external validity**, we applied appropriate countermeasures, such as extensive search terms, including sources from academic studies as well as grey literature provided by industry, in addition to a two-stage selection process of the interview partners. Regarding the two-stage selection process, the target group consists of three different categories of interview partners to incorporate several points of view. The categories comprise people from academia and industry. Interview partners from the industry can already use a continuous end-to-end lifecycle management pipeline for AI or are start-ups that develop AI applications and the associated lifecycle pipeline. These interviews also helped to evaluate and extend the results obtained from the MLR to provide a general depiction of tasks necessary for the continuous development of AI.

The scope of this study is to cover a comprehensive framework for the continuous development of AI models. Thus, this paper combines research on AI, ML and DL. The paper does not differentiate between them because results from the MLR treated them similarly and only the implementation of tasks in data handling or model learning slightly varies. In addition, the scope comprises different learning types, such as supervised, unsupervised and reinforcement learning. However, we did not consider special cases of learning techniques, such as federated learning or transfer learning.

7. Conclusion

In this paper, we aimed to provide a comprehensive, evidence-based foundation of established research on pipelines for the continuous development of AI. Thus, the main goal was to systematically identify relevant conceptual ideas, as well as to synthesize and structure the research in the area. To achieve this goal, we extracted 151 relevant formal and informal sources via a Multivocal Literature Review (MLR) and we executed nine semi-structured interviews (Steidl et al., 2022). Based on this information, we identified and compared five primarily used terms, such as DevOps for AI, CI/CD for AI, MLOps describing an extension of DevOps, the term end-to-end lifecycle management to describe the continuous execution of development and deployment tasks, and CD4ML describing the technical implementation of MLOps.

The paper also investigated potential main **triggers**, such as feedback and alert systems, orchestration service with a scheduled time, traditional repository updates, and manual triggers.

These triggers start the execution of the **pipeline**, which consists of four stages: (1) *Data Handling*, (2) *Model Learning*, (3) *Software Development* and (4) *System Operations*. The stage *Data Handling* comprises the repetitive end-to-end lifecycle of data-related tasks, such as pre-processing, quality assurance, versioning, and documentation. The stage *Model Learning* uses the output of the data handling and illustrates the tasks associated with the model development, such as model design, training, quality assurance, model improvement metadata capture, versioning, and documentation. After the pipeline handles the model learning, the stage *Software Development* prepares the model for deployment via packaging, software level quality assurance, and system versioning. The final stage *System Operations* deploys the AI model to a specific environment via different deployment strategies and monitors the system.

Furthermore, the paper maps 25 potential **challenges** regarding the implementation, adaption, and usage of pipelines for the continuous development of AI to the previously mentioned four stages. *Data Handling* related challenges encompass data collection and integration. *Model Learning* related challenges identify that versioning should depict the model evolution and ensure reproducibility. *Software Development* requires the pipeline to provide backward and forward compatibility of the model and data schema. The stage *System Operations* needs to handle multiple environments where cross-model inferences may occur. Ultimately challenges resulting from the pipeline requirements, such as the need for a flexible, customizable, and scalable pipeline are elaborated.

Regarding **future work**, we plan to further explore, evaluate and compare the pipeline concepts by prototypical implementations of demonstrators using available platforms that provide a pipeline for the continuous development of AI. Additional candidate platforms are, for instance, MLFlow, Uber's lifecycle platform for AI, called Michelangelo, ModelDB, or Facebook's FBLearner. The identified tasks and resulting taxonomy in this paper may then be compared with the already available tasks supported by the mentioned platforms. In addition, challenges for implementing, adapting, and using the pipeline for the continuous development of AI can also be tracked during the implementation and demonstration to complement our findings from the literature.

CRediT authorship contribution statement

Monika Steidl: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Michael Felderer:** Conceptualization, Methodology, Validation, Investigation, Data curation, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Rudolf Ramler:** Conceptualization, Methodology, Writing – review & editing, Funding acquisition.

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

I have shared the link to the data in the paper.

Acknowledgements

This work was supported by the Austrian Research Promotion Agency (FFG) in the frame of the project ConTest [888127] and the COMET competence center SCCH/INTEGRATE, Austria[865891, 892418]. We also thank all interview participants for their valuable input and feedback.

References

Alnafessah, Ahmad, Gias, Alim Ul, Wang, Runan, Zhu, Lulai, Casale, Giuliano, Filieri, Antonio, 2021. Quality-aware DevOps research: Where do we stand? IEEE Access 9, 44476–44489. http://dx.doi.org/10.1109/ACCESS.2021.3064867.

Baier, Lucas, Jöhren, Fabian, Seebacher, Stefan, 2019. Challenges in the deployment and operation of machine learning in practice.

Boucher, Philip, 2020. Artificial intelligence: How does it work, why does it matter,and what can we do about it?. European Parliament, Brussels.

Dellinger, Amy, 2005. Validity and the review of literature. Res. Schools 12.

Ereth, Julian, 2018. DataOps-Towards a Definition. In: Proceedings of the Conference Lernen, Wissen, Daten, Analysen. LWDA, URL https://ceur-ws.org/Vol-2191/paper13.pdf.

Figalist, Iris, Elsner, Christoph, Bosch, Jan, Olsson, Helena Holmström, 2020. An end-to-end framework for productive use of machine learning in software analytics and business intelligence solutions. In: Morisio, Maurizio, Torchiano, Marco, Jedlitschka, Andreas (Eds.), Product-Focused Software Process Improvement. In: LNCS sublibrary, SL 2, Programming and Software Engineering, vol. 12562, Springer, Cham, pp. 217–233. http://dx.doi.org/10.1007/978-3-030-64148-1_14.

Fischer, Lukas, Ehrlinger, Lisa, Geist, Verena, Ramler, Rudolf, Sobiezky, Florian, Zellinger, Werner, Brunner, David, Kumar, Mohit, Moser, Bernhard, 2020. AI system engineering key challenges and lessons learned. Mach. Learn. Knowl. Extraction 3 (1), 56–83.

Fitzgerald, Brian, Stol, Klaas-Jan, 2017. Continuous software engineering: A roadmap and agenda. J. Syst. Softw. 123, 176–189. http://dx.doi.org/10.1016/j.jss.2015.06.063, URL https://www.sciencedirect.com/science/article/pii/S0164121215001430.

Fredriksson, Teodor, Bosch, Jan, Olsson, Helena Holmstrm, 2020. Machine Learning Models for Automatic Labeling: A Systematic Literature Review. In: Proceedings of the 15th International Conference on Software Technologies - ICSOFT. pp. 552–561. http://dx.doi.org/10.5220/0009972705520561, URL https://orcid.org/0000-0003-2854-722X.

Garousi, Vahid, Felderer, Michael, Mäntylä, Mika V., 2016. The need for multivocal literature reviews in software engineering. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. EASE, ACM, New York, pp. 1–6. http://dx.doi.org/10.1145/2915970.2916008.

Garousi, Vahid, Felderer, Michael, Mäntylä, Mika V., 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. 106, 101–121. http://dx.doi.org/10.1016/j.infsof.2018.09.006, URL https://www.sciencedirect.com/science/article/pii/S0950584918301939.

Gmeiner, Johannes, Ramler, Rudolf, Haslinger, Julian, 2015. Automated testing in the continuous delivery pipeline: A case study of an online company. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, pp. 1–6.

Hand, David J., Khan, Shakeel, 2020. Validating and verifying AI systems. Patterns (New York, N.Y.) 1 (3), 100037. http://dx.doi.org/10.1016/j.patter.2020.100037.

Jalali, Samireh, Wohlin, Claes, 2012. Systematic literature studies. In: Runeson, Per (Ed.), 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM, IEEE, Piscataway, NJ, p. 29. http://dx.doi.org/10.1145/2372251.2372257.

John, Meenu Mary, Holmström Olsson, Helena, Bosch, Jan, 2021a. Architecting AI deployment: A systematic review of state-of-the-art and state-of-practice literature. In: Klotins, Eriks, Wnuk, Krzysztof (Eds.), Software Business. Springer International Publishing, Cham, pp. 14–29.

John, Meenu Mary, Olsson, Helena Holmstrom, Bosch, Jan, 2021b. Towards MLOps: A Framework and Maturity Model. In: Proceedings - 2021 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021. Institute of Electrical and Electronics Engineers Inc., pp. 334–341. http://dx.doi.org/10.1109/SEAA53835.2021.00050.

Jolliffe, Ian, 2005. Principal component analysis. In: Everitt, Brian, Howell, David C. (Eds.), Encyclopedia of Statistics in Behavioral Science. Wiley, Chichester, http://dx.doi.org/10.1002/0470013192.bsa501.

Karamitsos, Ioannis, Albarhami, Saeed, Apostolopoulos, Charalampos, 2020. Applying DevOps practices of continuous automation for machine learning. Information 11 (7), 363. http://dx.doi.org/10.3390/info11070363.

Kim, Gene, Humble, Jez, Debois, Patrick, Willis, John, Allspaw, John, 2016. The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations, First edition IT Revolution Press LLC, Portland OR.

Kitchenham, Barbara Ann, Charters, Stuart, 2007. Guidelines for performing systematic literature reviews in software engineering, 2.

Kolltveit, Ask Berstad, Li, Jingyue, 2022. Operationalizing Machine Learning Models - A Systematic Literature Review. In: Proceedings - Workshop on Software Engineering for Responsible AI, SE4RAI 2022. Institute of Electrical and Electronics Engineers Inc., pp. 1–8. http://dx.doi.org/10.1145/3526073.3527584.

Kreuzberger, Dominik, Kühl, Niklas, Hirschl, Sebastian, 2022. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. http://dx.doi.org/10.48550/arxiv.2205.02302, arXiv:2205.02302.

Lenarduzzi, Valentina, Lomio, Francesco, Moreschini, Sergio, Taibi, Davide, Tamburri, Damian, 2020. Software quality for AI: Where we are now?.

Lewis, Grace A., Ozkaya, Ipek, Xu, Xiwei, 2021. Software Architecture Challenges for ML Systems. In: Proceedings - 2021 IEEE International Conference on Software Maintenance and Evolution, ICSME 2021. Institute of Electrical and Electronics Engineers Inc., pp. 634–638. http://dx.doi.org/10.1109/ICSME52107.2021.00071.

Lo, Sin Kit, Lu, Qinghua, Wang, Chen, Paik, Hye Young, Zhu, Liming, 2021. ACM Comput. Surv. 54 (5), http://dx.doi.org/10.1145/3450288, arXiv:2007.11354.

Lorenzoni, Giuliano, Alencar, Paulo, Nascimento, Nathalia, Cowan, Donald, 2021. Machine learning model development from a software engineering perspective: A systematic literature review. URL http://arxiv.org/pdf/2102.07574v1.

Martínez-Fernández, Silverio, Bogner, Justus, Franch, Xavier, Oriol, Marc, Siebert, Julien, Trendowicz, Adam, Vollmer, Anna Maria, Wagner, Stefan, 2022. Software Engineering for AI-Based Systems: A Survey. ACM Trans. Softw. Eng. Methodol. (TOSEM) 31 (2), http://dx.doi.org/10.1145/3487043, arXiv:2105.01984.

Mayring, Philipp 1952-, 2015. Qualitative Inhaltsanalyse : Grundlagen und Techniken, 12., überarbeitete Auflage Pädagogik, Basel: Beltz and [Grünwald]: Preselect.media GmbH, Weinheim, URL https://bibsearch.uibk.ac.at/AC12283426.

Mboweni, Tsakani, Masombuka, Themba, Dongmo, Cyrille, 2022. A Systematic Review of Machine Learning DevOps. In: International Conference on Electrical, Computer, and Energy Technologies, ICECET 2022. Institute of Electrical and Electronics Engineers Inc., http://dx.doi.org/10.1109/ICECET55527.2022.9872968.

Messick, Samuel, 1995. Standards of validity and the validity of standards in performance asessment. Edu. Meas. Issues Pract. 14 (4), 5–8. http://dx.doi.org/10.1111/j.1745-3992.1995.tb00881.x.

Miles, Matthew B., Huberman, A. Michael, Saldaña, Johnny, 2014. Qualitative data analysis: A methods sourcebook, Edition 3 Sage, Los Angeles and London and New Delhi and Singapore and Washington DC.

Mishra, Alok, Otaiwi, Ziadoon, 2020. DevOps and software quality: A systematic mapping. Comp. Sci. Rev. 38, 100308. http://dx.doi.org/10.1016/j.cosrev.2020.100308, URL https://www.sciencedirect.com/science/article/pii/S1574013720304081.

Munappy, Aiswarya Raj, Mattos, David Issa, Bosch, Jan, Olsson, Helena Holmström, Dakkak, Anas, 2020. From Ad-Hoc data analytics to DataOps. In: Proceedings - 2020 IEEE/ACM International Conference on Software and System Processes, ICSSP 2020, vl. 20. Association for Computing Machinery, Inc, pp. 165–174. http://dx.doi.org/10.1145/3379177.3388909.

Nascimento, Elizamary, Nguyen-Duc, Anh, Sundbø, Ingrid, Conte, Tayana, 2020. Software engineering for artificial intelligence and machine learning software: A systematic literature review. URL http://arxiv.org/pdf/2011.03751v1.

Ng, Andrew, 2011. Sparse autoencoder. CS294A Lecture Notes 72 (2011), 1–19.

Nguyen-Duc, Anh, Sundbø, Ingrid, Nascimento, Elizamary, Conte, Tayana, Ahmed, Iftekhar, Abrahamsson, Pekka, 2020. A Multiple Case Study of Artificial Intelligent System Development in Industry. In: ACM International Conference Proceeding Series. Association for Computing Machinery, pp. 1–10. http://dx.doi.org/10.1145/3383219.3383220.

Paleyes, Andrei, Urma, Raoul-Gabriel, Lawrence, Neil D., 2020. Challenges in Deploying Machine Learning: a Survey of Case Studies, URL http://arxiv.org/pdf/2011.09926v2.

Pieters, Wolter, 2011. Explanation and trust: what to tell the user in security and AI? Ethics Inform. Technol. 13 (1), 53–64. http://dx.doi.org/10.1007/s10676-010-9253-3.

Pivarski, Jim, Bennett, Collin, Grossman, Robert L., 2016. Deploying analytics with the portable format for analytics (PFA). In: Krishnapuram, Balaji, Shah, Mohak, Smola, Alex, Aggarwal, Charu, Shen, Dou, Rastogi, Rajeev (Eds.), KDD2016. Association for Computing Machinery Inc. (ACM), New York, NY, pp. 579–588. http://dx.doi.org/10.1145/2939672.2939731.

Robinson, Oliver C., 2014. Sampling in interview-based qualitative research: A theoretical and practical guide. Qual. Res. Psychol. 11 (1), 25–41. http://dx.doi.org/10.1080/14780887.2013.801543.

Rodriguez, Manuel, Pires de Araújo, Luiz Jonatã, Mazzara, Manuel, 2020. Good practices for the adoption of DataOps in the software industry. J. Phys. Conf. Ser. 1694, 012032. http://dx.doi.org/10.1088/1742-6596/1694/1/012032.

Sculley, D., Holt, Gary, Golovin, Daniel, Davydov, Eugene, Phillips, Todd, Ebner, Dietmar, Chaudhary, Vinay, Young, Michael, Crespo, Jean-François, Dennison, Dan, 2015. Hidden technical debt in machine learning systems. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, vol. 28. Curran Associates, Inc, URL https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf.

Ståhl, Daniel, Bosch, Jan, 2014. Modeling continuous integration practice differences in industry software development. J. Syst. Softw. 87, 48–59. http://dx.doi.org/10.1016/j.jss.2013.08.032, URL https://www.sciencedirect.com/science/article/pii/S0164121213002276.

Steidl, Monika, Felderer, Michael, Ramler, Rudolf, 2022. Multivocal Literature Review & Interviews: Continuous End-to-End Lifecycle Management Pipeline for Artificial Intelligence. http://dx.doi.org/10.5281/zenodo.6384341, URL https://zenodo.org/record/6384341.

Stol, Klaas-Jan, Ralph, Paul, Fitzgerald, Brian, 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In: Proceedings of the 38th International Conference on Software Engineering. ACM, New York, NY, USA, http://dx.doi.org/10.1145/2884781, URL http://dx.doi.org/10.1145/2884781.2884833.

Stone, Peter, Brooks, Rodney, Brynjolfsson, Erik, Calo, Ryan, Etzioni, Oren, Hager, Greg, Hirschberg, Julia, Kalyanakrishnan, Shivaram, Kamar, Ece, Kraus, Sarit, Leyton-Brown, Kevin, Parkes, David, Press, William, Saxenian, AnnaLee, Shah, Julie, Tambe, Milind, Teller, Astro, 2016. Artificial intelligence and life in 2030: One hundred year study on artificial intelligence: Report of the 2015–2016 study panel. Stanford University URL https://ai100.stanford.edu/2016-report.

Tao, Chuanqi, Gao, Jerry, Wang, Tiexin, 2019. Testing and quality validation for AI software–perspectives, issues, and practices. IEEE Access 7, 120164–120175. http://dx.doi.org/10.1109/ACCESS.2019.2937107.

Testi, Matteo, Ballabio, Matteo, Frontoni, Emanuele, Iannello, Giulio, Moccia, Sara, Soda, Paolo, Vessio, Gennaro, 2022. MLOps: A taxonomy and a methodology. IEEE Access 10, 63606–63618. http://dx.doi.org/10.1109/ACCESS.2022.3181730.

Usman, Muhammad, Britto, Ricardo, Börstler, Jürgen, Mendes, Emilia, 2017. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. Inf. Softw. Technol. 85, 43–59.

Washizaki, Hironori, Uchida, Hiromu, Khomh, Foutse, Guéhéneuc, Yann Gaël, 2019. Studying Software Engineering Patterns for Designing Machine Learning Systems. In: Proceedings - 2019 10th International Workshop on Empirical Software Engineering in Practice, IWESEP 2019. Institute of Electrical and Electronics Engineers Inc., pp. 49–54. http://dx.doi.org/10.1109/IWESEP49350.2019.00017, arXiv:1910.04736.

Wohlin, Claes, 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. EASE '14, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2601248.2601268.

Wohlin, Claes, Runeson, Per, Höst, Martin, Ohlsson, Magnus C., Regnell, Björn, Wesslén, Anders, 2012. Experimentation in software engineering. Springer, Berlin, http://dx.doi.org/10.1007/978-3-642-29044-2, URL http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10650365.

Xie, Yuanhao, Cruz, Luís, Heck, Petra, Rellermeyer, Jan S., 2021. Systematic mapping study on the machine learning lifecycle. URL http://arxiv.org/pdf/2103.10248v1.

Xue, Bing, Zhang, Mengjie, Browne, Will N., 2013. Particle swarm optimization for feature selection in classification: a multi-objective approach. IEEE Trans. Cybern. 43 (6), 1656–1671. http://dx.doi.org/10.1109/TSMCB.2012.2227469.

Yasin, Affan, Fatima, Rubia, Wen, Lijie, Afzal, Wasif, Azhar, Muhammad, Torkar, Richard, 2020. On using grey literature and google scholar in systematic literature reviews in software engineering. IEEE Access 8, 36226–36243. http://dx.doi.org/10.1109/ACCESS.2020.2971712.

List of formal references

Aguilar Melgar, Leonel, Dao, David, Gan, Shaoduo, Gürel, Nezihe M., Hollenstein, Nora, Jiang, Jiawei, Karlaš, Bojan, Lemmin, Thomas, Li, Tian, Li, Yang, Rao, Susie, Rausch, Johannes, Renggli, Cedric, Rimanic, Luka, Weber, Maurice, Zhang, Shuai, Zhao, Zhikuan, Schawinski, Kevin, Wu, Wentao, Zhang, Ce, 2021. Ease. ML: A lifecycle management system for machine learning. http://dx.doi.org/10.3929/ETHZ-B-000458916.

Amershi, Saleema, Begel, Andrew, Bird, Christian, DeLine, Robert, Gall, Harald, Kamar, Ece, Nagappan, Nachiappan, Nushi, Besmira, Zimmermann, Thomas, 2019. Software engineering for machine learning: A case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice. IEEE, Piscataway, NJ, pp. 291–300. http://dx.doi.org/10.1109/ICSE-SEIP.2019.00042.

Díaz-de Arcaya, Josu, Miñón, Raúl, Torre-Bastida, Ana I., Del Ser, Javier, Almeida, Aitor, 2020. PADL: A modeling and deployment language for advanced analytical services. Sensors (Basel, Switzerland) 20 (23), http://dx.doi.org/10.3390/s20236712.

Azimi, Shelernaz, Pahl, Claus, 2021. Continuous Data Quality Management for Machine Learning based Data-as-a-Service Architectures.

Bachinger, Florian, Kronberger, Gabriel, 2020. Concept for a technical infrastructure for management of predictive models in industrial applications. In: Moreno-Díaz, Roberto, Pichler, Franz (Eds.), Computer Aided Systems Theory - EUROCAST 2019. In: Lecture Notes in Computer Science, vol. 12013, Springer International Publishing, Cham, pp. 263–270. http://dx.doi.org/10.1007/978-3-030-45093-9_32.

Banerjee, Amitabha, Chen, Chien-Chia, Hung, Chien-Chun, Huang, Xiaobo, Wang, Yifan, Chevesaran, Razvan, 2020. Challenges and experiences with mlops for performance diagnostics in hybrid-cloud enterprise software deployments. In: 2020 USENIX Conference on Operational Machine Learning (OpML 20). USENIX Association, URL https://www.usenix.org/conference/opml20/presentation/banerjee.

Barrak, Amine, Eghan, Ellis E., Adams, Bram, 2021. On the co-evolution of ML pipelines and source code – empirical study of DVC projects. In: Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution, and Reengineering. SANER, Hawaii, USA.

Baylor, Denis, Breck, Eric, Cheng, Heng-Tze, Fiedel, Noah, Foo, Chuan Yu, Haque, Zakaria, Haykal, Salem, Ispir, Mustafa, Jain, Vihan, Koc, Levent, Koo, Chiu Yuen, Lew, Lukasz, Mewald, Clemens, Modi, Akshay Naresh, Polyzotis, Neoklis, Ramesh, Sukriti, Roy, Sudip, Whang, Steven Euijong, Wicke, Martin, Wilkiewicz, Jarek, Zhang, Xin, Zinkevich, Martin, 2017. TFX: A TensorFlow-based production-scale machine learning platform. In: Matwin, Stan, Yu, Shipeng, Farooq, Faisal (Eds.), KDD2017. Association for Computing Machinery Inc. (ACM), New York, NY, pp. 1387–1395. http://dx.doi.org/10.1145/3097983.3098021.

Baylor, Denis, Haas, Kevin, Katsiapis, Konstantinos, Leong, Sammy, Liu, Rose, Menwald, Clemens, Miao, Hui, Polyzotis, Neoklis, Trott, Mitchell, Zinkevich, Martin, 2019. Continuous training for production ML in the TensorFlow extended (TFX) platform. In: 2019 USENIX Conference on Operational Machine Learning (OpML 19). USENIX Association, Santa Clara, CA, pp. 51–53, URL https://www.usenix.org/conference/opml19/presentation/baylor.

Benbya, Hind, Davenport, Thomas H., Pachidi, Stella, 2020. Artificial intelligence in organizations: Current state and future opportunities. SSRN Electron. J. http://dx.doi.org/10.2139/ssrn.3741983.

Boag, Scott, Dube, Parijat, Herta, Benjamin, Hummer, Waldemar, Ishakian, Vatche, Jayaram, K., Kalantar, Michael, Muthusamy, Vinod, Nagpurkar, Priya, Rosenberg, Florian (Eds.), 2017. Scalable multi-framework multi-tenant lifecycle management of deep learning training jobs.

Boovaraghavan, Sudershan, Maravi, Anurag, Mallela, Prahaladha, Agarwal, Yuvraj, 2021. MLIoT: An end-to-end machine learning system for the internet-of-things. In: Proceedings of the International Conference on Internet-of-Things Design and Implementation. ACM, New York, NY, USA, pp. 169–181. http://dx.doi.org/10.1145/3450268.3453522.

Borg, Markus, 2021. The AIQ meta-testbed: Pragmatically bridging academic AI testing and industrial q needs. In: Winkler, Dietmar, Biffl, Stefan, Mendez, Daniel, Wimmer, Manuel, Bergsmann, Johannes (Eds.), Software Quality: Future Perspectives on Software Engineering Quality. Springer International Publishing, Cham, pp. 66–77.

Bourgais, Aurélien, Ibnouhsein, Issam, 2021. Ethics-by-design: the next frontier of industrialization. AI and Ethics http://dx.doi.org/10.1007/s43681-021-00057-0.

Breck, Eric, Polyzotis, Neoklis, Roy, Sudip, Whang, Steven, Zinkevich, Martin (Eds.), 2019. Data Validation for Machine Learning.

Brumbaugh, Eli, Kale, Atul, Luque, Alfredo, Nooraei, Bahador, Park, John, Puttaswamy, Krishna, Schiller, Kyle, Shapiro, Evgeny, Shi, Conglei, Siegel, Aaron, Simha, Nikhil, Bhushan, Mani, Sbrocca, Marie, Yao, Shi-Jing, Yoon, Patrick, Zanoyan, Varant, Zeng, Xiao-Han T., Zhu, Qiang, Cheong, Andrew, Du, Michelle Gu-Qian, Feng, Jeff, Handel, Nick, Hoh, Andrew, Hone, Jack, Hunter, Brad, 2019. Bighead: A framework-agnostic, end-to-end machine learning platform. In: Singh, Lisa (Ed.), 2019 IEEE International Conference on Data Science and Advanced Analytics. IEEE, Piscataway, NJ, pp. 551–560. http://dx.doi.org/10.1109/DSAA.2019.00070.

Castellanos, Camilo, Varela, Carlos A., Correal, Dario, 2021. ACCORDANT: A domain specific-model and DevOps approach for big data analytics architectures. J. Syst. Softw. 172, 110869. http://dx.doi.org/10.1016/j.jss.2020.110869, URL https://www.sciencedirect.com/science/article/pii/S0164121220302594.

Caveness, Emily, G. C., Paul Suganthan, Peng, Zhuo, Polyzotis, Neoklis, Roy, Sudip, Zinkevich, Martin, 2020. TensorFlow data validation: Data analysis and validation in continuous ML pipelines. In: Maier, David, Pottinger, Rachel, Doan, AnHai, Tan, Wang-Chiew, Alawini, Abdussalam, Ngo, Hung Q. (Eds.), Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA, pp. 2793–2796. http://dx.doi.org/10.1145/3318464.3384707.

Chard, Ryan, Li, Zhuozhao, Chard, Kyle, Ward, Logan, Babuji, Yadu, Woodard, Anna, Tuecke, Steven, Blaiszik, Ben, Franklin, Michael J., Foster, Ian, 2019. DLHub: Model and data serving for science. In: 2019 IEEE 33rd International Parallel and Distributed Processing Symposium. IEEE, Piscataway, NJ, pp. 283–292. http://dx.doi.org/10.1109/IPDPS.2019.00038.

Ciucu, R., Adochiei, F.C., Adochiei, Ioana-Raluca, Argatu, F., Serian, G.C., Enache, B., Grigorescu, S., Argatu, Violeta Vasilica, 2019. Innovative devops for artificial intelligence. Sci. Bullet. Electr. Eng. Faculty 19 (1), 58–63. http://dx.doi.org/10.1515/sbeef-2019-0011.

Corbeil, Jean-Philippe, Daudens, Florent (Eds.), 2020. Deploying a Cost-Effective and Production-Ready Deep News Recommender System in the Media Crisis Context.

Derakhshan, Behrouz, Mahdiraji, Alireza Rezaei, Rabl, Tilmann, Markl, Volker, 2019. Continuous deployment of machine learning pipelines. In: EDBT.

Fehlmann, Thomas, Kranich, Eberhard, 2020. A framework for automated testing. In: Yilmaz, Murat, Niemann, Jörg, Clarke, Paul, Messnarz, Richard (Eds.), Systems, Software and Services Process Improvement. Springer International Publishing, Cham, pp. 275–288.

Fursin, Grigori, Guillou, Herve, Essayan, Nicolas, 2020. CodeReef: an open platform for portable mlops, reusable automation actions and reproducible benchmarking. URL http://arxiv.org/pdf/2001.07935v2.

Garcia, Rolando, Sreekanti, Vikram, Yadwadkar, Neeraja, Crankshaw, Daniel, Gonzalez, Joseph E., Hellerstein, Joseph M. (Eds.), 2018. Context: The missing piece in the machine learning lifecycle. 114.

Gerostathopoulos, Ilias, Kugele, Stefan, Segler, Christoph, Bures, Tomas, Knoll, Alois, 2019. Automated trainability evaluation for smart software functions. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering : 10-15 November 2019, San Diego, California. IEEE Computer Society, Los Alamitos, CA, pp. 998–1001. http://dx.doi.org/10.1109/ASE.2019.00096.

Gharibi, Gharib, Walunj, Vijay, Nekadi, Raju, Marri, Raj, Lee, Yugyung, 2021. Automated end-to-end management of the modeling lifecycle in deep learning. Empir. Softw. Eng. 26 (2), http://dx.doi.org/10.1007/s10664-020-09894-9.

Hummer, Waldemar, Muthusamy, Vinod, Rausch, Thomas, Dube, Parijat, El Maghraoui, Kaoutar, Murthi, Anupama, Oum, Punleuk, 2019. ModelOps: Cloud-based lifecycle management for reliable and trusted AI. In: 2019 IEEE International Conference on Cloud Engineering. Conference Publishing Services, IEEE Computer Society, Los Alamitos, California and Washington and Tokyo, pp. 113–120. http://dx.doi.org/10.1109/IC2E.2019.00025.

Jackson, Stuart, Yaqub, Maha, Li, Cheng-Xi, 2018. The agile deployment of machine learning models in healthcare. Front. Big Data 1, 7. http://dx.doi.org/10.3389/fdata.2018.00007.

Janardhanan, P.S., 2020. Project repositories for machine learning with TensorFlow. Procedia Comput. Sci. 171, 188–196. http://dx.doi.org/10.1016/j.procs.2020.04.020, URL https://www.sciencedirect.com/science/article/pii/S1877050920309856.

John, Meenu Mary, Olsson, Helena Holmström, Bosch, Jan, 2020. Developing ML/DL models: A design framework. In: Proceedings of the International Conference on Software and System Processes. ICSSP '20, Association for Computing Machinery, New York, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/3379177.3388892.

Junsung, Lim, Hoejoo, Lee, Youngmin, Won, Hunje, Yeon, 2019. MLOp lifecycle scheme for vision-based inspection process in manufacturing. In: 2019 USENIX Conference on Operational Machine Learning (OpML 19). USENIX Association, Santa Clara, CA, pp. 9–11, URL https://www.usenix.org/conference/opml19/presentation/lim.

Karlaš, Bojan, Interlandi, Matteo, Renggli, Cedric, Wu, Wentao, Zhang, Ce, Mukunthu Iyappan Babu, Deepak, Edwards, Jordan, Lauren, Chris, Xu, Andy, Weimer, Markus, 2020. Building continuous integration services for machine learning. In: Gupta, Rajesh (Ed.), Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. In: ACM Digital Library, Association for Computing Machinery, New York,NY,United States, pp. 2407–2415. http://dx.doi.org/10.1145/3394486.3403290.

Karlaš, Bojan, Liu, Ji, Wu, Wentao, Zhang, Ce, 2018. Ease.ml in action: towards multi-tenant declarative learning services. Proc. VLDB Endow. 11 (12), 2054–2057. http://dx.doi.org/10.14778/3229863.3236258.

Kronberger, Gabriel, Bachinger, Florian, Affenzeller, Michael, 2020. Smart manufacturing and continuous improvement and adaptation of predictive models. Procedia Manuf. 42, 528–531. http://dx.doi.org/10.1016/j.promfg.2020.02.037.

Leff, Deborah, Lim, Kenneth T.K., 2021. The key to leveraging AI at scale. J. Rev. Pricing Manag. http://dx.doi.org/10.1057/s41272-021-00320-3.

Li, Zhuozhao, Chard, Ryan, Ward, Logan, Chard, Kyle, Skluzacek, Tyler J., Babuji, Yadu, Woodard, Anna, Tuecke, Steven, Blaiszik, Ben, Franklin, Michael J., Foster, Ian, 2021. DLHub: Simplifying publication, discovery, and use of machine learning models in science. J. Parallel Distrib. Comput. 147, 64–76. http://dx.doi.org/10.1016/j.jpdc.2020.08.006, URL https://www.sciencedirect.com/science/article/pii/S0743731520303464.

Liu, Wei-Chen, Chiang, Yu Ting, Liang, Tyng-Yeu, 2019. A development platform of intelligent mobile APP based on edge computing. In: 2019 Seventh International Symposium on Computing and Networking Workshops. CANDARW, IEEE, [Place of publication not identified], pp. 235–241. http://dx.doi.org/10.1109/CANDARW.2019.00048.

Lopez Garcia, Alvaro, de Lucas, Jesus Marco, Antonacci, Marica, zu Castell, Wolfgang, David, Mario, Hardt, Marcus, Lloret Iglesias, Lara, Molto, Germen, Plociennik, Marcin, Tran, Viet, Alic, Andy S., Caballer, Miguel, Plasencia, Isabel Campos, Costantini, Alessandro, Dlugolinsky, Stefan, Duma, Doina Cristina, Donvito, Giacinto, Gomes, Jorge, Heredia Cacha, Ignacio, Ito, Keiichi, Kozlov, Valentin Y., Nguyen, Giang, Orviz Fernandez, Pablo, Sustr, Zdenek, Wolniewicz, Pawel, 2020. A cloud-based framework for machine learning workloads and applications. IEEE Access 8, 18681–18692. http://dx.doi.org/10.1109/ACCESS.2020.2964386.

Lwakatare, Lucy Ellen, Crnkovic, Ivica, Bosch, Jan, 2020a. DevOps for AI – challenges in development of AI-enabled applications. In: Begušić, Dinko (Ed.), 2020 28th International Conference on Software, Telecommunications and Computer Networks (SoftCOM). IEEE, [Piscataway, NJ], pp. 1–6. http://dx.doi.org/10.23919/SoftCOM50211.2020.9238323.

Lwakatare, Lucy Ellen, Crnkovic, Ivica, Rånge, Ellinor, Bosch, Jan, 2020b. From a data science driven process to a continuous delivery process for machine learning systems. In: Morisio, Maurizio, Torchiano, Marco, Jedlitschka, Andreas (Eds.), Product-Focused Software Process Improvement. Springer International Publishing, Cham, pp. 185–201.

Lwakatare, Lucy Ellen, Raj, Aiswarya, Bosch, Jan, Olsson, Helena Holmström, Crnkovic, Ivica, 2019. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In: Kruchten, Philippe, Fraser, Steven, Coallier, François (Eds.), Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing, Cham, pp. 227–243.

Makarov, Vladimir A., Stouch, Terry, Allgood, Brandon, Willis, Chris D., Lynch, Nick, 2021. Best practices for artificial intelligence in life sciences research. Drug Discov. Today.

Mäkinen, Sasu, Skogström, Henrik, Laaksonen, Eero, Mikkonen, Tommi, 2021. Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?. URL http://arxiv.org/pdf/2103.08942v1.

Martel, Yannick, Roßmann, Arne, Sultanow, Eldar, Weiß, Oliver, Wissel, Matthias, Pelzel, Frank, Seßler, Matthias, 2021. Software architecture best practices for enterprise artificial intelligence. In: INFORMATIK 2020. pp. 165–181. http://dx.doi.org/10.18420/INF2020_16.

Martínez-Fernández, Silverio, Franch, Xavier, Jedlitschka, Andreas, Oriol, Marc, Trendowicz, Adam, 2021. Developing and operating artificial intelligence models in trustworthy autonomous systems. In: Cherfi, Samira, Perini, Anna, Nurcan, Selmin (Eds.), Research Challenges in Information Science. In: Lecture Notes in Business Information Processing, vol. 415, Springer International Publishing, Cham, pp. 221–229. http://dx.doi.org/10.1007/978-3-030-75018-3_14.

Maskey, Manil, Molthan, Andrew, Hain, Chris, Ramachandran, Rahul, Gurung, Iksha, Freitag, Brian, Miller, J.J., Ramasubramanian, Muthukumaran, Bollinger, Drew, Mestre, Ricardo, Cecil, Daniel, 2019. Machine learning lifecycle for earth science application: A practical insight into production deployment. In: IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium. IEEE, pp. 10043–10046. http://dx.doi.org/10.1109/IGARSS.2019.8899031.

Miao, Hui, Chavan, Amit, Deshpande, Amol, 2017a. ProvDB, : Lifecycle management of collaborative analysis workflows. In: Proceedings of the 2nd Workshop on Human-in-the-Loop Data Analytics. ACM, New York, NY, USA, pp. 1–6. http://dx.doi.org/10.1145/3077257.3077267.

Miao, Hui, Li, Ang, Davis, Larry S., Deshpande, Amol, 2017b. ModelHub: Deep learning lifecycle management. In: ICDE 2017. IEEE, Piscataway, NJ, pp. 1393–1394. http://dx.doi.org/10.1109/ICDE.2017.192.

Miao, Hui, Li, Ang, Davis, Larry S., Deshpande, Amol, 2017c. Towards unified data and lifecycle management for deep learning. In: ICDE 2017. IEEE, Piscataway, NJ, pp. 571–582. http://dx.doi.org/10.1109/ICDE.2017.112.

Nashaat, Mona, Ghosh, Aindrila, Miller, James, Quader, Shaikh, Marston, Chad, 2019. M-lean: An end-to-end development framework for predictive models in B2B scenarios. Inf. Softw. Technol. 113, 131–145. http://dx.doi.org/10.1016/j.infsof.2019.05.009, URL https://www.sciencedirect.com/science/article/pii/S0950584919301247.

Olston, Christopher, Fiedel, Noah, Gorovoy, Kiril, Harmsen, Jeremiah, Lao, Li, Li, Fangwei, Rajashekhar, Vinu, Ramesh, Sukriti, Soyke, Jordan, 2017. TensorFlow-serving: Flexible, high-performance ML serving.

Peili, Yang, Xuezhen, Yin, Jian, Ye, Lingfeng, Yang, Hui, Zhao, Jimin, Liang, 2018. Deep learning model management for coronary heart disease early warning research. In: 2018 the 3rd IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA 2018). IEEE, Piscataway, NJ, pp. 552–557. http://dx.doi.org/10.1109/ICCCBDA.2018.8386577.

Pölöskei, István, 2020. MLOps approach in the cloud-native data pipeline design. Acta Tech. J. http://dx.doi.org/10.14513/actatechjaur.00581.

Polyzotis, Neoklis, Roy, Sudip, Whang, Steven Euijong, Zinkevich, Martin, 2017. Data management challenges in production machine learning. In: Chirkova, Rada, Yang, Jun, Suciu, Dan (Eds.), SIGMOD'17, May 14-19, 2017, Chicago, IL, USA. ACM, New York, NY, USA, pp. 1723–1726. http://dx.doi.org/10.1145/3035918.3054782.

Polyzotis, Neoklis, Roy, Sudip, Whang, Steven Euijong, Zinkevich, Martin, 2018. Data lifecycle challenges in production machine learning. ACM SIGMOD Record 47 (2), 17–28. http://dx.doi.org/10.1145/3299887.3299891.

Raj, Aiswarya, Bosch, Jan, Olsson, Helena Holmstrom, Wang, Tian J., 2020. Modelling data pipelines. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 13–20. http://dx.doi.org/10.1109/SEAA51224.2020.00014.

Raj, Emmanuel, Westerlund, Magnus, Espinosa-Leal, Leonardo, 2021. Reliable fleet analytics for edge IoT solutions. CLOUD COMPUTING 2020. Nice, France, pp. 55–62, URL http://arxiv.org/pdf/2101.04414v1.

Rausch, Thomas, Dustdar, Schahram, 2019. Edge intelligence: The convergence of humans, things, and AI. In: 2019 IEEE International Conference on Cloud Engineering. Conference Publishing Services, IEEE Computer Society, Los Alamitos, California and Washington and Tokyo, pp. 86–96. http://dx.doi.org/10.1109/IC2E.2019.00022.

Rausch, Thomas, Hummer, Waldemar, Muthusamy, Vinod, Rashed, Alexander, Dustdar, Schahram, 2019. Towards a serverless platform for edge AI. In: 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19). USENIX Association, Renton, WA, URL https://www.usenix.org/conference/hotedge19/presentation/rausch.

Renggli, Cedric, Hubis, Frances Ann, Karlaš, Bojan, Schawinski, Kevin, Wu, Wentao, Zhang, Ce, 2019a. Ease.ml/ci and ease.ml/meter in action: towards data management for statistical generalization. Proc. VLDB Endow. 12 (12), 1962–1965. http://dx.doi.org/10.14778/3352063.3352110.

Renggli, Cedric, Rimanic, Luka, Kolar, Luka, Wu, Wentao, Zhang, Ce, 2020. Ease.ml/snoopy in action: Towards automatic feasibility analysis for machine learning application development. Proc. VLDB Endow. 13 (12), 2837–2840. http://dx.doi.org/10.14778/3415478.3415488.

Rivero, Luis, Diniz, João, Silva, Giovanni, Borralho, Gabriel, Braz Junior, Geraldo, Paiva, Anselmo, Alves, Erika, Oliveira, Milton, 2020. Deployment of a machine learning system for predicting lawsuits against power companies: Lessons learned from an agile testing experience for improving software quality.

In: Machado, Ivan, Costa, Heitor, Albuquerque, Adriano, Sampaio, Suzana, Santos, Rodrigo, Reinehr, Sheila, Santos, Gleison, Bezerra, Sandro, Viana, Davi, Albuquerque, Adriano Bessa, da Rocha, Ana Regina Cavalcanti, Thiry, Marcello, Barcellos, Monalessa Perini, Conte, Tayana, Schots, Marcelo (Eds.), 19th Brazilian Symposium on Software Quality. ACM, New York, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/3439961.3439991.

Sangiovanni, Mirella, Schouten, Gerard, van den Heuvel, Willem-Jan, 2020. An IoT beehive network for monitoring urban biodiversity: Vision, method, and architecture. In: Dustdar, Schahram (Ed.), Service-Oriented Computing. Springer International Publishing, Cham, pp. 33–42.

Schelter, Sebastian, Biessmann, F., Januschowski, Tim, Salinas, David, Seufert, Stephan, Szarvas, Gyuri, 2018. On challenges in machine learning model management. IEEE Data Eng. Bull. 41, 5–15.

Schleier-Smith, Johann, 2015. An architecture for agile machine learning in real-time applications. In: Cao, Longbing, Zhang, Chengqi, Joachims, Thorsten, Webb, Geoff, Margineantu, Dragos D., Williams, Graham (Eds.), Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 10 - 13, 2015, Sydney, Australia. ACM, New York, NY, pp. 2059–2068. http://dx.doi.org/10.1145/2783258.2788628.

Schreiber, Marc, Barkschat, Kai, Kraft, Bodo, 2014. Using continuous integration to organize and monitor the annotation process of domain specific corpora. In: 2014 5th International Conference on Information and Communication Systems. ICICS, IEEE, Piscataway, NJ, pp. 1–6. http://dx.doi.org/10.1109/IACS.2014.6841958.

Spell, Derrick C., Zeng, Xiao-Han T., Chung, Jae Young, Nooraei, Bahador, Shomer, Richard T., Wang, Ling-Yong, Gibson, James C., Kirsche, Daniel, 2017. Flux: Groupon's automated, scalable, extensible machine learning platform. In: Nie, Jian-Yun, Obradovic, Zoran, Suzumura, Toyotaro, Ghosh, Rumi, Nambiar, Raghunath, Wang, Chonggang (Eds.), 2017 IEEE International Conference on Big Data. IEEE, Piscataway, NJ, pp. 1554–1559. http://dx.doi.org/10.1109/BigData.2017.8258089.

Tamburri, Damian A., 2020. Sustainable mlops: Trends and challenges. In: 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. SYNASC, pp. 17–23. http://dx.doi.org/10.1109/SYNASC51798.2020.00015.

Vartak, Manasi, Subramanyam, Harihar, Lee, Wei-En, Viswanathan, Srinidhi, Husnoo, Saadiyah, Madden, Samuel, Zaharia, Matei, 2016. ModelDB: a system for machine learning model management. In: Binnig, Carsten, Fekete, Alan, Nandi, Arnab (Eds.), Proceedings of the Workshop on Human-in-the-Loop Data Analytics - HILDA '16. ACM Press, New York, New York, USA, pp. 1–3. http://dx.doi.org/10.1145/2939502.2939516.

Vuppalapati, Chandrasekar, Ilapakurti, Anitha, Chillara, Karthik, Kedari, Sharat, Mamidi, Vanaja, 2020. Automating tiny ML intelligent sensors devops using microsoft azure. In: 2020 IEEE International Conference on Big Data (Big Data). IEEE, pp. 2375–2384. http://dx.doi.org/10.1109/BigData50022.2020.9377755.

Wachsmuth, Sven, Schulz, Simon, Lier, Florian, Siepmann, Frederic, Lütkebohle, Ingo, 2012. The robot head "flobi": A research platform for cognitive interaction technology.

Yun, Fan, Shibahara, Toshiki, Ohsita, Yuichi, Chiba, Daiki, Akiyama, Mitsuaki, Murata, Masayuki, 2020. Understanding machine learning model updates based on changes in feature attributions.

Zaharia, Matei, Chen, Andrew, Davidson, Aaron, Ghodsi, Ali, Hong, Sue Ann, Konwinski, Andy, Murching, Siddharth, Nykodym, Tomas, Ogilvie, Paul, Parkhe, Mani, 2018. Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng. Bull. 41 (4), 39–45.

Zhang, Huaizheng, Li, Yuanming, Huang, Yizheng, Wen, Yonggang, Yin, Jianxiong, Guan, Kyle, 2020. MLModelCI: An automatic cloud platform for efficient mlaas. In: Wen Chen, Chang (Ed.), Proceedings of the 28th ACM International Conference on Multimedia. In: ACM Digital Library, Association for Computing Machinery, New York,NY,United States, pp. 4453–4456. http://dx.doi.org/10.1145/3394171.3414535.

Zhou, Yue, Yu, Yue, Ding, Bo, 2020. Towards MLOps: A case study of ML pipeline platform. In: 2020 International Conference on Artificial Intelligence and Computer Engineering. ICAICE, IEEE, pp. 494–500. http://dx.doi.org/10.1109/ICAICE51518.2020.00102.

List of informal references

Ammanath, Beena, Farrall, Frank, Kuder, David, Mittal, Nitin, 2021. MLOps: Industrialized AI: Scaling model development and operations with a dose of engineering and operational discipline. URL https://www2.deloitte.com/us/en/insights/focus/tech-trends/2021/mlops-industrialized-ai.html.

Windheuser, Christop, 0000. Continuous Delivery for Machine Learning (CD4ML) Webinar Series, URL https://www.thoughtworks.com/continuous-delivery-for-machine-learning.

Anon, 2021a. Architecture for mlops using TFX, kubeflow pipelines, and cloud build. URL https://cloud.google.com/architecture/architecture-for-mlops-using-tfx-kubeflow-pipelines-and-cloud-build.

Anon, 2021b. MLOps: Continuous delivery and automation pipelines in machine learning. URL https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

Arnold, Matthew, Boston, Jeffrey, Desmond, Michael, Duesterwald, Evelyn, Elder, Benjamin, Murthi, Anupama, Navratil, Jiri, Reimer, Darrell, 2020. Towards automating the AI operations lifecycle. URL http://arxiv.org/pdf/2003.12808v1.

Aronchick, David, 2020. CI/CD + ML==MLOps - the way to speed bringing machine learning to production - david aronchick: DeliveryConf. URL https://www.youtube.com/watch?v=uOCR4Xw-BZ8.

Aronchick, David, Buss, Marvin, Matheson, Zander, Haviv, Yaron, 2020. Git-based CI / CD for machine learning & mlops. URL https://www.iguazio.com/blog/git-based-ci-cd-for-machine-learning-mlops/.

Arora, Akshay, Nethi, Arun, Kharat, Priyanka, Verghese, Vency, Jenkins, Grant, Miff, Steve, Chowdhry, Vikas, Wang, Xiao, 2019. ISTHMUS: Secure, scalable, real-time and robust machine learning platform for healthcare. URL http://arxiv.org/pdf/1909.13343v2.

Baroni, Kate, 2018. Getting AI/ML and DevOps working better together | blog e aggiornamenti di azure | microsoft azure. URL https://azure.microsoft.com/it-it/blog/getting-ai-ml-and-devops-working-better-together/,

Breuel, Cristiano, 2020. ML ops: Machine learning as an engineering discipline: As ML matures from research to applied business solutions, so do we need to improve the maturity of its operation processes. URL https://towardsdatascience.com/ml-ops-machine-learning-as-an-engineering-discipline-b86ca4874a3f.

Castanyer, Roger Creus, Martínez-Fernández, Silverio, Franch, Xavier, 2021. Integration of convolutional neural networks in mobile applications. URL http://arxiv.org/pdf/2103.07286v1.

de la Rúa Martínez, Javier, 2020. Scalable architecture for automating machine learning model monitoring.

Duvall, Paul, 2018. Continuous delivery for machine learning with AWS CodePipeline and amazon SageMaker: Aws summit. URL https://dzone.com/articles/screencast-continuous-delivery-for-machine-learnin.

Erb, Dillon, 2019. CI/CD for machine learning & AI: FirstMark's data driven NYC. URL https://www.youtube.com/watch?v=JpWUw4co75s.

Ettun, Yochay, 2019. In: cnvrg.io (Ed.), Making your machine learning operational with CI/CD. URL https://cnvrg.io/ci-cd-machine-learning/.

Felderer, Michael, Ramler, Rudolf, 2021. Quality assurance for AI-based systems: Overview and challenges. URL http://arxiv.org/pdf/2102.05351v1.

Gorcenski, Emily, 2019. Continuous delivery for machine learning (CD4ml): Keeping models fresh with continuous intelligence: Data engineering meetup berlin. URL https://www.youtube.com/watch?v=UzVa5azAHkc.

Gorcenski, Emily, 2020. Continuous delivery for machine learning: Patterns and pains - emily gorcenski: DeliveryConf. URL https://www.youtube.com/watch?v=bFW5mZmj0nQ.

Granlund, Tuomas, Kopponen, Aleksi, Stirbu, Vlad, Myllyaho, Lalli, Mikkonen, Tommi, 2021. MLOps challenges in multi-organization setup: Experiences from two real-world cases. URL http://arxiv.org/pdf/2103.08937v1.

Guo, Yu, Ashmawy, Khalid, Huang, Eric, Zeng, Wei, 2020. Under the hood of uber atg's machine learning infrastructure and versioning control platform for self-driving vehicles. URL https://eng.uber.com/machine-learning-model-life-cycle-version-control/,

Gupta, Abhishek, Galinkin, Erick, 2020b. Green lighting ML: Confidentiality, integrity, and availability of machine learning systems in deployment. URL http://arxiv.org/pdf/2007.04693v1.

Haakman, Mark, Cruz, Luís, Huijgens, Hennie, van Deursen, Arie, 2020. AI lifecycle models need to be revised. an exploratory study in fintech. arXiv preprint arXiv:2010.02716.

Haviv, Yaron, 2020. Webinar: Mlops automation with git based CI/CD for ML: Cloud native computing foundation (CNCF). URL https://www.youtube.com/watch?v=VCUDo9umKEQ.

Hermann, Jeremy, Del Balso, Mike, 2017. In: Uber (Ed.), Meet Michelangelo: Uber's Machine Learning Platform. URL https://eng.uber.com/michelangelo-machine-learning-platform/.

Huang, Chong, Nourian, Arash, Griest, Kevin, 2021. Hidden technical debts for fair machine learning in financial services. URL http://arxiv.org/pdf/2103.10510v2.

Hubis, Frances Ann, Wu, Wentao, Zhang, Ce, 2019. Quantitative overfitting management for human-in-the-loop ml application development with ease.ml/meter. URL http://arxiv.org/pdf/1906.00299v3.

Katsiapis, Konstantinos, Karmarkar, Abhijit, Altay, Ahmet, Zaks, Aleksandr, Polyzotis, Neoklis, Ramesh, Anusha, Mathes, Ben, Vasudevan, Gautam, Giannoumis, Irene, Wilkiewicz, Jarek, Simsa, Jiri, Hong, Justin, Trott, Mitchell, Lutz, Noé, Dournov, Pavel A., Crowe, Robert, Sirajuddin, Sarah, Warkentin, Tris Brian, Li, Zhitao, 2020. Towards ML engineering: A brief history of TensorFlow extended (TFX).

Keating, Nate, 2020. An introduction to mlops on google cloud. URL https://www.youtube.com/watch?v=6gdrwFMaEZ0.

Kent, Edward, Doran, Paul, 2019. Continuous deployment of machine learning models by Edward Kent & Paul Doran: Devoxx. URL https://www.youtube.com/watch?v=wnx5yYVf2hQ.
Khan, Asif, 2018. Continuous delivery for AI applications: QCon.ai. URL https://www.youtube.com/watch?v=OgedlOxeT2c.
Lavin, Alexander, Gilligan-Lee, Ciarán M., Visnjic, Alessya, Ganju, Siddha, Newman, Dava, Ganguly, Sujoy, Lange, Danny, Baydin, Atılım Güneş, Sharma, Amit, Gibson, Adam, Gal, Yarin, Xing, Eric P., Mattmann, Chris, Parr, James, 2021. Technology readiness levels for machine learning systems. URL http://arxiv.org/pdf/2101.03989v1.
Liu, Yong, Brooks, Andrew, 2020. Continuous delivery of deep transformer-based NLP models using mlflow and AWS sagemaker for enterprise AI scenarios - databricks: Spark + AI summit 2020. URL https://databricks.com/session_na20/continuous-delivery-of-deep-transformer-based-nlp-models-using-mlflow-and-aws-sagemaker-for-enterprise-ai-scenarios.
Mäkinen, Sasu, 2021. Designing an open-source cloud-native MLOps pipeline.
Meynard, Theodore, Machado, Jean Carlo, 2021. Laying the foundation of our open source ML platform with a modern CI/CD pipeline and mlflow. URL https://inside.getyourguide.com/blog/2021/3/3/-open-source-ml-platform-with-a-modern-ci-cd-pipeline-and-mlflow.
Moesta, Mary Grace, Tamisin, Peter, 2020. Productionalizing models through CI/CD design with mlflow: Spark + ai summit 2020. URL https://databricks.com/de/session_na20/productionalizing-models-through-ci-cd-design-with-mlflow,
Mulkens, Jan, 2020. MLOps, automated machine learning made easy: ML conference. URL https://www.youtube.com/watch?v=4n2SYq0bbnw.
O'Brien, Elle, 2021. Continuous integration for machine learning. URL https://www.youtube.com/watch?v=A3OEaaiGPhk.
Patel, Shivani, 2020. MLOps feature dive: CI/CD with GitHub actions. URL https://www.youtube.com/watch?v=qRi1zy4lNZw.
Patel, Shivani, Edwards, Jordan, 2019. MLOps: Accelerating data science with DevOps - microsoft: DOES19. URL https://www.youtube.com/watch?v=pqppGvTJm-A.
Pentreath, Nick, 2019. Continuous deployment for deep learning: Spark + AI summit europe. URL https://www.youtube.com/watch?v=qwyW0pHz9ag.
Popp, Matthias, 2019. Comprehensive support of the lifecycle of machine learning models in model management systems. http://dx.doi.org/10.18419/OPUS-10690.
Raj, Emmanuel, 2020. Edge MLOps framework for AIoT applications.
Rausch, Thomas, Hummer, Waldemar, Muthusamy, Vinod, 2020. PipeSim: Trace-driven simulation of large-scale AI operations platforms. URL http://arxiv.org/pdf/2006.12587v1.
Renggli, Cedric, Karlaš, Bojan, Ding, Bolin, Liu, Feng, Schawinski, Kevin, Wu, Wentao, Zhang, Ce, 2019b. Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment. URL http://arxiv.org/pdf/1903.00278v1.
Renggli, Cedric, Rimanic, Luka, Gürel, Nezihe Merve, Karlaš, Bojan, Wu, Wentao, Zhang, Ce, 2021. A data quality-driven view of mlops. URL http://arxiv.org/pdf/2102.07750v1.
Rosenbaum, Sasha, 2020. In: InfoQ (Ed.), CI/CD for Machine Learning. QCon, URL https://www.infoq.com/presentations/ci-cd-ml/.
Santhanam, P., Farchi, Eitan, Pankratius, Victor, 2019. Engineering reliable deep learning systems. arXiv preprint arXiv:1910.12582.
Sato, Danilo, 2020. CD4ml and the challenges of testing and quality in ML systems: TensorFlow London meetup. URL https://www.youtube.com/watch?v=adexWmMYLcw,
Sato, Danilo, Wider, Arif, Windheuser, Christoph, 2019a. Continuous delivery for machine learning: Automating the end-to-end lifecycle of machine learning applications. URL https://martinfowler.com/articles/cd4ml.html.
Sato, Danilo, Wider, Arif, Windheuser, Christoph, 2019b. Continuous delivery for machine learning: Getting machine learning applications into production is hard. URL https://www.thoughtworks.com/insights/articles/intelligent-enterprise-series-cd4ml.
Saucedo, Alejandro, 2020. A practical CI/CD framework for machine learning at massive scale: FOSDEM 2020. URL https://archive.fosdem.org/2020/schedule/event/a_practical_cicd_framework_for_machine_learning_at_massive_scale/.
Schruhl, Daniel, Windheuser, Christoph, 2020. Continuous delivery for machine learning applications with open source tools: Ml conference. URL https://www.youtube.com/watch?v=ub9XIgcUMAQ.
Seyffarth, Roman, 2019. Machine learning: Moving from experiments to production. URL https://blog.codecentric.de/en/2019/03/machine-learning-experiments-production/.
Shtelma, Michael, Shiviah, Thunder, 2020. Continuous delivery of ML-enabled pipelines on databricks using MLflow: Spark + AI summit 2020. URL https://www.youtube.com/watch?v=Gjns_Z0zxt8.
Sierra, Xavier, 2018. Our continuous deployment pipeline. URL https://techblog.fexcofts.com/2018/09/03/our-continuous-deployment-pipeline/.
Singhal, Aditi, Kumar, Vivek, 2020. Continuous integration and continuous delivery for machine learning models (CodeLabs tech talk 2020): CodeLabs tech talk. URL https://www.youtube.com/watch?v=YjrkIsDZ0Hc.
Spieker, Helge, Gotlieb, Arnaud, 2019. Towards testing of deep learning systems with training set reduction. URL http://arxiv.org/pdf/1901.04169v1.
Srinivasan, Srivatsan, 2021. An introduction to MLOps: Nuts and bolts of MLOps. URL https://www.youtube.com/watch?v=K6CWjg09fAQ.
Stirbu, Vlad, Granlund, Tuomas, Helén, Jere, Mikkonen, Tommi, 2021. Extending SOUP to ML models when DesigningCertified medical systems. URL http://arxiv.org/pdf/2103.09510v1.
Stumpf, Kevin, Bedratiuk, Stepan, Olcay, Cirit, 2018. Michelangelo PyML: Introducing uber's platform for rapid Python ML model development. URL https://eng.uber.com/michelangelo-pyml/.
Tandon, Rohit, Pati, Sanghamitra, 2021. ML-Oops to MLOps. URL https://www2.deloitte.com/us/en/blog/deloitte-on-cloud-blog/2021/ml-oops-to-mlops.html.
Vadavalasa, Ram Mohan, 2020. End to end CI/CD pipeline for machine learning. Int. J. Adv. Res. Ideas Innov. Technol. 6 (3), URL https://www.ijariit.com/manuscript/end-to-end-ci-cd-pipeline-for-machine-learning/.
Visengeriyeva, Larysa, Kammer, Anja, Bär, Isabel, Kniesz, Alexander, Plöd, Michael, 2021. Machine learning operations. URL https://ml-ops.org/,
Wilkiewicz, Jarek, Haas, Kevin, Doshi, Tulsee, Katsiapis, Konstantinos, 2019. From research to production with TFX pipelines and ML metadata: Google I/O 2019. URL https://blog.tensorflow.org/2019/05/research-to-production-with-tfx-ml.html.
Windheuser, Christoph, Sato, Danilo, 2020. MLOps: Continuous delivery for machine learning on AWS: AWS whitepapers & guides. URL https://d1.awsstatic.com/whitepapers/mlops-continuous-delivery-machine-learning-on-aws.pdf?did=wp_card&trk=wp_card.
Xin, Doris, Miao, Hui, Parameswaran, Aditya, Polyzotis, Neoklis, 2021. Production machine learning pipelines: Empirical analysis and optimization opportunities. URL http://arxiv.org/pdf/2103.16007v1.
Xu, Runyu, 2020. A design pattern for deploying machine learning models to production.
Yasar, Hasan, 2020a. Software Development AI and DevOps. CARNEGIE-MELLON UNIV PITTSBURGH PA PITTSBURGH.
Yasar, Hasan, 2020b. Leveraging DevOps and DevSecOps to accelerate AI development and deployment.
Zweben, Monte, 2021. How optimizing MLOps can revolutionize enterprise AI. URL https://www.infoq.com/articles/optimizing-mlops-enterprise-ai/.

Monika Steidl is doing her PhD at the University of Innsbruck, Austria. In her research she focuses on software quality assurance for microservices and AI with a specific focus on Big Data Applications, Business Intelligence solutions and AI. In addition, she focuses on quality assurance during runtime where CI/CD pipelines and anomaly detection methods in combination with risk analysis are her main research interest.

Michael Felderer is a professor at the Department of Computer Science at the University of Innsbruck, Austria and a guest professor at the Department of Software Engineering at the Blekinge Institute of Technology, Sweden. His fields of expertise and interest include software quality, testing, software processes, AI & Software Engineering as well as empirical software engineering. Michael Felderer holds a habilitation degree from the University of Innsbruck, co-authored more than 150 publications and received 11 best paper awards. He is an internationally recognized member of the software engineering research community and supports it as an editorial board member of the journals Information and Software Technology (IST) and Software Tools for Technology Transfer (STTT) as well as organizer of conferences and regular PC member of premier conferences.

Rudolf Ramler is a research manager at Software Competence Center Hagenberg (SCCH), Austria. Rudolf holds a M.Sc. in Business Informatics from Johannes Kepler University Linz. He has more than 20 years of experience in applied research in the fields of software engineering, software quality assurance and testing, software analytics, and application lifecycle management. He is author of over 100 reviewed publications related to these topics, co-organizer and chair of international conferences and workshops, an ISTQB certified tester, and an IEEE and ACM member. His mission and passion are to support industry in turning research results into practically successful solutions.