UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Enhancing collaboration and agility in data-centric AI projects

Fabian Stieler, Bernhard Bauer

Angaben zur Veröffentlichung / Publication details:

Stieler, Fabian, and Bernhard Bauer. 2024. "Enhancing collaboration and agility in data-centric AI projects." In *Evaluation of Novel Approaches to Software Engineering: 18th International Conference, ENASE 2023, Prague, Czech Republic, April 24–25, 2023, revised selected papers*, edited by Hermann Kaindl, Mike Mannion, and Leszek A. Maciaszek, 321–43. Cham: Springer. https://doi.org/10.1007/978-3-031-64182-4_15.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Enhancing Collaboration and Agility in Data-Centric AI Projects

Fabian Stieler(✉) and Bernhard Bauer

Software Methodologies for Distributed Systems,
University of Augsburg, Augsburg, Germany
{fabian.stieler,bernhard.bauer}@uni-a.de

Abstract. Usually, mature Artificial Intelligence (AI) projects are developed by a team of various members, such as data engineers, data scientists, software engineers and machine learning (ML) engineers. They often pursue highly heterogeneous approaches, leading to new challenges in collaboration, particularly regarding software quality, data versioning and the traceability of model metrics and other resulting artifacts. These challenges are further intensified when AI projects rely on dynamic datasets, introducing an entirely new dimension that teams must deal with. Adopting principles from the machine learning operations (MLOps) paradigm becomes essential in this context. To go beyond existing process models and develop actionable guidelines, our work introduces a Git workflow for AI projects. We present basic instructions for data and code while outlining a minimal infrastructure setup. Building upon abstract concepts, we delve into concrete, actionable steps by examining the proposed branching workflow. Through a case study, we apply the development methodology to two use cases and demonstrate that the principles and approaches positively impact project outcomes.

Keywords: Software engineering for machine learning · Agile development · MLOps · AI development · Data-centric AI

1 Introduction

The attention around AI has recently reached unprecedented levels, so the proliferation of projects in companies across different industries continues. AI-based technologies offer the possibility to make data-driven decisions, optimize operational processes, and show enormous potential for developing innovative solutions and new products.

From a developers perspective, AI projects require novel development approaches compared to traditional software projects due to their inherent characteristics, especially when projects leave the proof-of-concept stage and grow into mature projects. While an ecosystem of methods, concepts, and tools has been emerged for conventional software projects, these solutions cannot be

directly applied to AI projects. A fundamental difference between AI- and traditional software projects, primarily relying on source code to create artifacts, is the involvement of code and data as input.

In this context, data volatility poses a significant challenge in AI projects, as the ML artifacts, which include trained ML models, often need to be recreated more frequently than in conventional software projects. This is required to account for the evolving nature of data and to ensure the ML models' accuracy and relevance and reinforced by the recent trend in the AI community, facing a mindset change towards increasing awareness of data dependencies in implementing powerful AI systems [21]. At the same time, the data dependencies possibly create a certain amount of sluggishness due to computing intensity and therefore, additional resources are required to produce these artifacts [2,27].

To address this, approaches are currently arising in the domain of MLOps. Building upon traditional software engineering processes such as DevOps, the concepts are being supplemented with ML- and data workflows. An example is our approach presented in [31] for a development method for data-centric AI projects focusing on active learning. Here, a Git workflow is introduced, which assists teams in structuring their projects and addressing implementation challenges.

In this article, we build on these concepts and provide detailed principles for code and data in Sect. 3. We abstract the Git workflow and demonstrate in Sect. 4 through a more detailed guide of an exemplary development process that these approaches, besides AL, can be helpful for other data-centric AI projects as well. Furthermore, we extend the evaluation and present a case study in Sect. 5, where two use cases applied the introduced methodology. First, the foundations are presented in the course of the following Sect. 2.

2 Foundations

A paradigm shift has been evident in the research community, which views data as the core and driving factor in developing and implementing AI systems [40]. In this context, high data quality, variety, relevance, and quantity are central to the development process, with data serving as a major building block for the continuous improvement of AI models.

The characteristic of data-centric development is that particular emphasis is placed on collecting, processing, managing, and analyzing data to ensure the integrity and usefulness of the data to train high-quality, high-performance models [13,36]. Looking at implementation in practice, data-centric projects are considered inherently complex and multi-factorial. To address this complexity, methodologies have been established to provide a structured approach to data-heavy projects.

2.1 Process Models

One of the earliest process models hails from traditional data mining. Knowledge Discovery in Databases (KDD), defined as the non-trivial process of iden-

tifying valid, novel, potentially useful, and ultimately understandable patterns in data, comprises five main phases: Selection, Preprocessing, Transformation, Data Mining, and Interpretation/Evaluation [5]. The process is divided into nine sub-steps designed to be interactive and iterative, involving multiple user decisions to achieve optimal results [6]. Within the context of KDD, considerable emphasis is placed on data quality and appropriate data preprocessing, as these factors directly impact the results' efficiency.

The Cross-Industry Standard Process for Data Mining (CRISP-DM) is a process model divided into six phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment [37]. Like KDD, CRISP-DM is iterative and cyclical, accommodating the often nonlinear nature of data mining projects. In addition to KDD, CRISP-DM underscores the importance of business understanding and emphasizes that the project should be closely aligned with business requirements and goals.

In contrast to traditional process models for data mining projects, the Team Data Science Process (TDSP) represents a major change. TDSP is also iterative, but the iterations envisage a more agile procedure, thus enabling effective and efficient collaboration in data science teams. It breaks down the data science process into five clearly defined phases: Business Understanding, Data Acquisition and Understanding, Modeling, Deployment, and Customer Acceptance. Each phase includes specific tasks and milestones and assigns clear roles to individual team members, aimed at boosting the team's productivity through high transparency and coordination. [19]

Recently, more approaches have been added. A prominent concept focusing on ML/AI projects was presented in [33]. The authors propose the cross-industry standard process model for developing machine learning applications with the quality assurance methodology CRISP-ML(Q), which is compatible with CRISP-DM. The main difference with CRISP-DM is that the Business and Data Understanding phases are combined, and a Monitoring & Maintenance phase supplements the overall process.

All of the process models mentioned are agnostic to the underlying problem. Therefore, they can be applied across a broad spectrum of application fields, regardless of the type of data and algorithms involved. It is crucial to remember that the quality of data significantly contributes to the success of a project, irrespective of the process model a team chooses. In this context, concepts from MLOps, an emerging discipline that deals with the systematic approach to managing the lifecycle and artifacts in an AI project, are paramount.

2.2 MLOps

The various process models introduced in Chap. 2.1 assist AI project teams in understanding *what* needs to be done in the individual phases of the lifecycles of data and ML models. MLOps, inspired by the principles of DevOps, pertains to the practices and methods that harmonize the various stages in the lifecycle of AI systems and enable their efficient production, implementation, monitoring, and iteration in an industrial context [16]. Along with several facets, MLOps

aims to answer *how* a team can manage the emerging complexities associated with developing and managing AI systems [22].

The goal is to create a cooperative and efficient environment in an AI project that combines various roles, such as data engineers, data scientists, software engineers, and ML engineers, and promotes seamless integration and interaction to enhance productivity. Key components are envisaged, such as version control, not only for source code, as in traditional software development, but also for data, configurations, and ML models, facilitating the critical aspect of reproducibility [23].

Another fundamental principle is the high degree of automation for continuous integration (CI), continuous delivery (CD), and -deployment, which also includes continuous training (CT), the iterative retraining of ML models, for example, periodically or through defined triggers such as a changed data basis [14,15,18].

Automated testing is integral to MLOps, broadening the application of CI/CD principles within the realm of ML [30]. By continuously testing the models against predefined metrics and criteria, it is possible to ensure quality and reliability before implementation [3].

In this context, another component is model monitoring. Unlike conventional software, ML models are susceptible to concept drift, where the statistical properties of the target variables can change over time, potentially affecting model performance [7,17]. Monitoring strategies aim to enable early detection of such shifts through continuous evaluation and validation and allow a team timely intervention and model updating through appropriate measures.

Another aspect is the creation of robust infrastructure that ultimately supports the end-to-end process, from data collection and preparation to model development and validation to deployment and subsequent monitoring and maintenance. Factors such as scalability and performance, as well as security issues, are relevant, which in summary, should enable sustainable and efficient implementation of AI projects in practice [41].

In their mixed-method research, the authors in [16] found that various challenges prevail in the MLOps domain. They enumerate ML system, operational, and organizational challenges, with the latter particularly attributable to a cultural change and the understanding of MLOps as a multidisciplinary group process. Currently, this is often hampered by siloed work, necessitating measures for better collaboration [20,34].

While there is already existing work investigating tools in this context [8, 25], we aim to contribute by presenting our proposal for a Git workflow-based development methodology for data-centric AI projects in the following sections.

3 Core Concepts for an Effective Collaboration

Similar to how DevOps principles and best practices methods such as Git-Flow and trunk-based software development have emerged in the past, the research

community has recently published numerous best practices for AI projects and is still working on MLOps principles [28].

Our work presented in [31] introduces an agile development methodology that provides guidelines for a team of developers and data scientists to structure their work, focusing on data-driven development. To implement the basic concept from this, such as the use of runners for CI, CD, and CT for the realization of automated iterations, we introduce the following basic concepts related to data and code and give a brief overview of the infrastructure setup.

3.1 Enable Developers to Develop with Data

After the initial conception phase, from a technical process perspective, AI projects typically start with collecting and preprocessing raw data. Like the program code, the data exhibits a persistent dynamism across the project's duration. These changes can result from adding new data results and new project requirements, which in turn require the collection and processing of additional data. In practical terms, new raw data may need to be imported or new data labels introduced in a subsequent iteration phase. Here, it becomes clear that the artifacts generated by an AI project do not solely depend on the underlying code but are significantly influenced by the data used and processed. This necessitates efficient and traceable data versioning. Although it is possible to do this manually - for example, by assigning timestamps to the different versions of the dataset - a dedicated tool for data versioning is often recommended in practice to ensure optimal transparency and reproducibility, in particular when certification and qualification of AI systems are addressed.

Developing an ML pipeline with complete, often massive datasets can make implementation inefficient for various reasons. Factors such as data selection and extended computation time can significantly slow down the engineer's workflow. For this reason, we recommend creating a so-called development dataset. This provides a representative but significantly smaller subset of the original data and is designed so that the entire ML pipeline only takes a few minutes to perform its calculations. This approach allows developers to use the development dataset for local development on their local systems. Additionally, the runner or the executing system can quickly provide feedback on whether a specific code change leads to a disruption or other problems in the ML pipeline.

The creation of this development dataset must be reproducible, version-safe and should therefore not be done manually. The ideal solution, as shown in Fig. 1, is a standalone code module that draws samples from the original full datasets. This module should allow automatic updates of the partial datasets when the full dataset changes. Assuming this code module offers the option to include specific samples or sample clusters in the development dataset, it could serve as a basis for regression tests. In these tests, samples that have caused problems in the ML pipeline in the past could be continuously monitored and examined from the beginning of model training to identify and fix potential error sources early.

Considering the dynamic nature of the data in some AI projects, further requirements arise for this development dataset, shown in Fig. 1. As seen from the change from the first to the second iteration, the Dev-Dataset-Creator module can mitigate the dynamics. Depending on the use case, the data basis is subject to permanent changes, e.g., due to stream data, which makes development with this data impractical, even if these were processed into batches. However, the development dataset should approximate the distribution of the entire datasets as accurately as possible, which occurs between the second and third iterations. If the distribution of the full dataset changes significantly, a new version of the development dataset is created automatically ($dev_1 \rightarrow dev_2$).

Furthermore, outliers and distorted samples should be included from the beginning to ensure a realistic development environment. If new data sources are tapped during the project's duration, these must be considered in the development dataset, as seen in the fourth iteration.

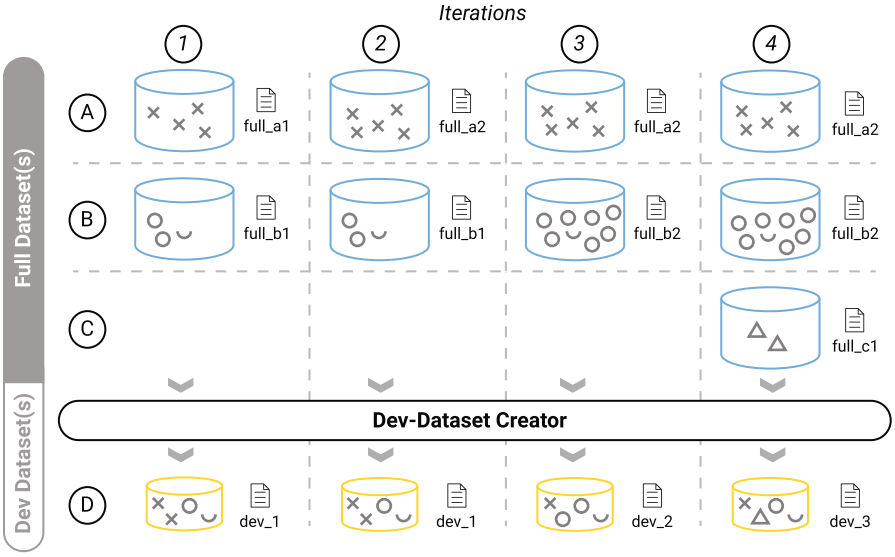

Fig. 1. Development Dataset Concept.

Figure 1 illustrates a simplified outline. A project may require various development datasets that differ in size and content. It's important to understand that this dataset does not claim to serve for training a high-performance model, nor - contrary to common terminology - to test its performance during development time. As a primary target group of the development dataset, the team can use it during the development iteration to check the executability of the ML pipeline code on their local clients before their push and use it, e.g., for integration tests in a CI job, and therefore improve the quality of the committed code.

3.2 Agree on a Common Code- And Project Structure

In the early stages, particularly during prototyping, it is common to implement all steps of the ML pipeline in a single script or notebook [26]. However, this approach can lead to limitations as these artifacts are often limited to the workspace of an individual developer and can pose challenges in tracking version differences. Therefore, effective collaboration in an AI project requires splitting the individual steps of an ML pipeline into separate modules to involve a team of developers [1]. This allows the members to focus on specific phases of the pipeline and distribute their work efficiently.

We propose a project structure as outlined in Fig. 2, where it is evident that the individual code modules reflect the different phases of an ML pipeline[1] When designing an ML pipeline, it is crucial to identify the necessary steps and clearly define the dependencies between the various stages.

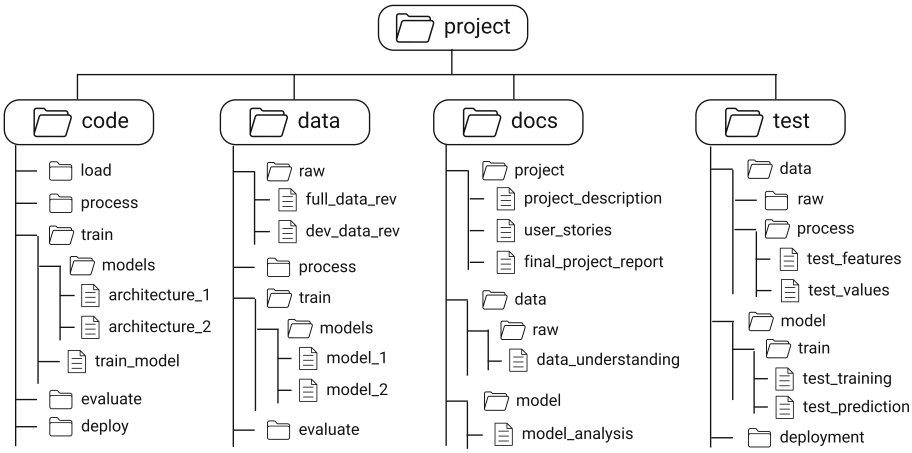

Fig. 2. Proposed Project Structure.

As soon as a team begins the implementation phase, it is essential to ensure that the data passes through the ML pipeline as early as possible, even if it has yet to produce meaningful results or artifacts such as trained models. The development dataset mentioned in Sect. 3.1 can be used to provide data to the code modules during the development phase. By ensuring this early data flow, developers can work simultaneously on different phases and manage their individual tasks.

Therefore, it is necessary to transform all steps of the ML pipeline into an automatically executable form. A sustainable code implementation, e.g., using

[1] A suitable template for projects implemented in Python can be found at: https://www.github.com/ds-lab/lifedata-project-template.

parameters and configuration files, allows for different implementations and settings of each stage. By tracking these through a version control system like Git and using feature flags, a systematic comparison is possible even when multiple developers work on the same ML pipeline stage.

To meet further requirements regarding traceability, the "version number" of the input dataset must be treated as part of the configuration. Some data versioning tools offer this by storing hashes of the datasets as version numbers in the configuration files. Other concepts use references, such as a data catalogue. Moreover, these tools can often cache all ML pipeline artifacts (e.g., preprocessed data, trained models, or evaluation reports). They can restore these artifacts if an execution with identical code and data occurred in the past.

This capability creates synergies in terms of the runtimes of the ML pipeline throughout the AI project, whether in the local development environment, experimenting with the entire dataset, or in the production environment. For example, when new training data flow into the pipeline, the model can be retrained and, if necessary, optimized and evaluated. On the other hand, the computationally intensive preprocessing step can be repeated for only some of the datasets. By skipping this step, the new model can be quickly deployed.

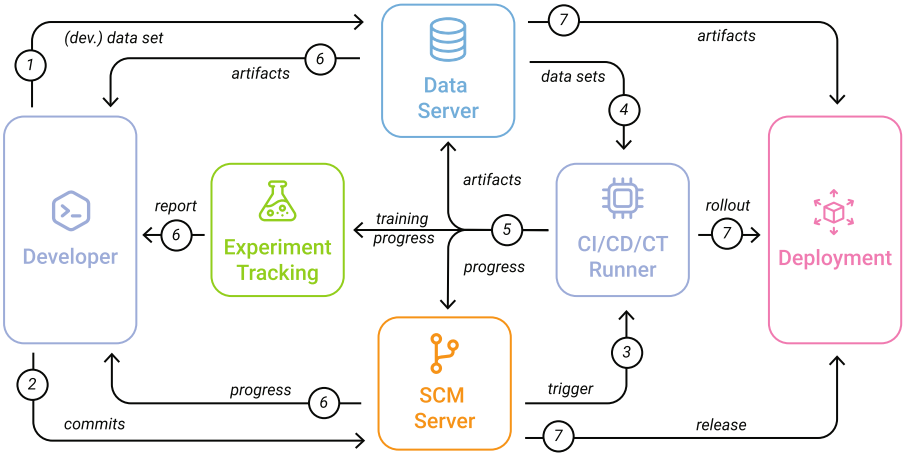

Fig. 3. Infrastructure Setup.

3.3 Setting up the Basic Infrastructure

An important aspect in AI projects is the increased demand for infrastructure. This is reflected in the growing field of MLOps research, which encompasses managing different environments and effectively scaling hardware resources, along with the associated concepts and tools. Figure 3 outlines a simplified infrastructure setup agnostic to the underlying technologies and illustrates the necessary components for implementing a distributed system for AI projects.

In step ①, the raw data and the created development dataset must made available to the data server. Typically, the upload is not done directly from the developer's client, but it is imported from the corresponding data source for larger datasets. The development dataset remains on the developer's client and should be small enough to enable fast iterations during ongoing work. Once the required datasets are available on the data server, at step ②, developers commit and push their code to the Source Control Management (SCM) system. An SCM application is able to trigger a runner, the CI/CD/CT runner, at step ③. This runner should be hosted on a powerful machine (e.g., with GPUs) and takes into account the planning, management, and scaling of the computational resources needed for executing the ML pipeline. This becomes necessary when the resources exceed the capacities of the developer's computer, such as training the model on the entire dataset.

Each job is uniquely associated with a specific commit, facilitating easy decision-making on whether certain long-running jobs can be terminated. Existing tools and frameworks from traditional software projects can be reused to manage and monitor the servers where the runners are executed. In step ④, the runner checks out the version of the dataset specified in the configuration files and runs the ML pipeline.

In step ⑤, the runner reports the status to the SCM system and uploads artifacts, such as the trained model or preprocessed data, to the data server. This can occur when the developer submits new code, conducts experiments on the entire dataset, or triggers an automated job, such as scheduled nightly retraining. The SCM application allows access to the runner's logs, providing the simplest solution for monitoring the progress of the ML pipeline. This can be further enhanced by using appropriate tools like MLflow [39], which enables the collaborating team of data scientists at step ⑥ to make structured comparisons of experiments or monitor the model in the production environment.

The presented infrastructure concept is minimalistic and can be expanded in various ways: Separate runners for CI, CD, and CT can be replaced by powerful clusters, the data server can be enhanced by a feature store and/or a model registry, and many more possibilities. Additionally, the deployment implementation, as shown in step ⑦, can be further refined and is intended to indicate the interconnections of the individual components.

4 Git Workflow for AI Projects

One of the core ideas presented in [31] is to utilize the CI/CD/CT runners introduced in Sect. 3.3 for automation purposes using a branch-based workflow concept. This involves introducing different namespaces for branches, with each namespace focusing on either the code or data dimension. The runners behave differently based on the namespace of the branch in which they are triggered.

This organizational structure provides a structured environment for a team collectively working on an AI project repository. Furthermore, it serves as a catalyst for agile development by efficiently adapting the feedback cycles to the

specific implementation types, whether application- or data-oriented tasks. To further investigate the branching workflow, we will illustrate the concept through an example project in the subsequent chapter.

4.1 Establishing a Robust ML Pipeline

After the scoping phase is completed and the setup described in Sect. 3 is in place, the code for the initial ML pipeline is first transferred to the repository's main branch in the SCM application. Similar to traditional software projects, the **main branch**, contains the most complete version of the code. It is advisable to follow the proposed code structure from the beginning, even if, in the initial steps, the focus should still be on the exploratory phase of the data. At the beginning of an AI project, the rest of the code may consist solely of stubs and interfaces for each downstream phase of the ML pipeline. Subsequently, the primary requirement for the main branch is to provide a clean, executable, and stable version of the ML pipeline.

In this regard, the focus of the CI/CD/CT runners in its branch namespace lies on code quality: they execute the ML pipeline using the development dataset as a form of integration testing. Given that the configuration files contain information about the allowable values of all parameters (cf. Sect. 3.2), they can verify the code across the permissible combinations, which, even with the development dataset, can prove to be time-consuming.

Furthermore, traditional unit tests and other code analysis steps should be performed to maintain the desired high quality in the main branch. Similar to traditional software development practices, static analysis tools can be employed to uncover potential issues in the code, such as poor programming practices or unused variables. These additional steps contribute to the early detection of potential errors and ensure a robust codebase for the ML pipeline. By incorporating quality assurance aspects, all team members are empowered to address potential issues and provide smooth executability of the ML pipeline at all times.

4.2 Development Procedure

As soon as the development team envisages a new feature or the need arising from altered requirements, a **feature branch** is created, adhering to the established concept of traditional software projects. Usually, this need is documented in an issue within the SCM application, and the feature branch, exemplified in Fig. 4 by commit (A1), is generated. The branch where the implementation takes place is at the level of code-focused branches. In this process, the developer generates a new feature flag along with the necessary parameters and implements the new feature, utilizing the development dataset.

After every push commit, the runner verifies the ML pipeline's correct executability, checking out the development dataset for the code execution. Analogous to a CI pipeline in classic software development, static code checks and unit tests are conducted here as well. The rapid executability using the development dataset provides the author with swift feedback about the success of the CI jobs.

However, the training results of the ML pipeline are irrelevant. Hence all services used in the project to track experiments remain deactivated.

Once the developer is convinced that the problem described in the issue has been solved, he initiates a merge request. Such requests enable other team members to provide feedback and review the code before it flows into the main branch and morphs from commit (A2) into Version (0.2). At this point, a performance increase of the model due to the newly implemented function is not necessarily required. As illustrated by the example of the first feature branch, it can sometimes suffice to merely implement and test the function, enabling other team members to build upon the latest features as quickly as possible. In certain cases, combining various features might be necessary to genuinely enhance the ML model's performance. As the project follows the code structure outlined in Sect. 3.2, it is entirely conceivable that several developers are simultaneously implementing different functions in various branches.

Sometimes, the implementation of certain functions in an AI project necessitates their execution on the entire dataset, and this even before the implementing team member initiate a merge request. This case is exemplified in Fig. 4 within the second feature branch (B1). In this scenario, the developer decides to initiate an experiment and branches out with the corresponding code version into one or more experiment branches.

4.3 Experimentation

Many tasks in an AI project are experimental in nature. While the widespread way of implementation for data science related jobs takes place in notebooks, in our concept we introduce custom branch namespace. This special category of branches, called **experiment branches**, runners execute the ML pipeline on full datasets and not only on development datasets as is traditionally the case. This expands the testing space and potentially provides more representative results.

An exemplary situation is the introduction of a new model architecture by a team of developers. The executable code is implemented in the feature branch

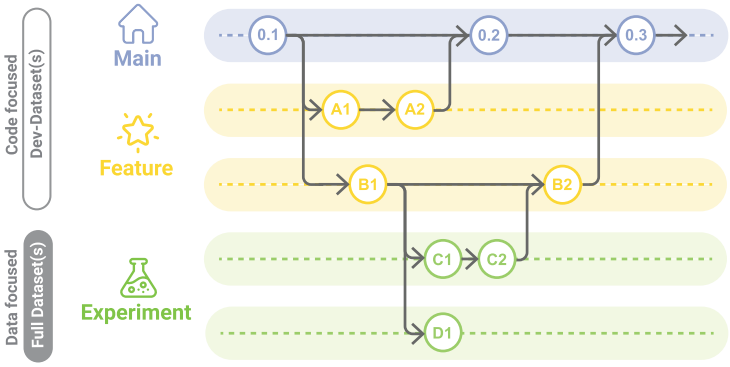

Fig. 4. Git Workflow with Experiment-Branches.

(B1). The next step would be to train and evaluate the new model's performance. This is where the experimental branch comes into play.

A developer creates one or more branches in the experiment namespace and then configures the ML pipeline according to the specific needs of the test. This is done by enabling the feature flag and setting appropriate parameters in the configuration files. Each of these experiment branches can focus on specific aspects of the ML pipeline. For example, one experiment might be specifically reserved for hyperparameter optimization, while another might enable additional data augmentation.

Once a configuration is set, for example, as illustrated in the commit (C1), this triggers the runner to access the full datasets. If the ML pipeline execution is complete, the runner stores the artifacts on the data server using the data versioning system. Consequently, it is ensured that all boundary conditions - from the data version to the code version to the execution environment - are documented for each artifact.

This concept can additionally be used for feasibility experiments. These provide the developers with preliminary evaluative feedback regarding the feasibility and potential efficacy of the current function under consideration. In such cases, it might be helpful to trigger proof-of-concept experiments. In doing so, the focus shifts from fine-tuning the model to roughly evaluating ML performance. If an experiment fails, as visible in the example of commit (D1), the overhead of a merge request and code reviews is avoided. This saves downstream resources and allows for more efficient use of development time.

4.4 Release New Versions

Suppose the results of an experiment outperform the current leading model or a completed feature is to be released. In that case, merging from the main branch to the version branch is initiated. Throughout this phase of the model lifecycle, the implemented code as well as the produced artifacts, are subject to thorough inspection and assessment by additional team members.

In the scenario, illustrated in Fig. 5, the acceptance of the merge request of version (1.0) triggers the activation of a new runner on the **release branch**. Due to the caching feature of a data versioning tool, this runner can restore and reactivate the artifacts from an experimental run on the data server. Should errors or undesired states occur during or after the deployment, proven, corrective measures from traditional software development can be applied, such as reverting the release branch to an earlier commit. The same applies to model versions in case of a later concept deviation. The release branch provides traceability for the entire development team, allowing recourse to already released versions of a model registry.

We establish an additional specialized branch type to accommodate the continuous dynamics of data and code within a data-centric AI project. This new branch exists on the data-focused layer of the project and serves as a code-based version twin of the release branch. This is of particular importance as it allows maintaining synchronized versioning of code and data, which is essential for the

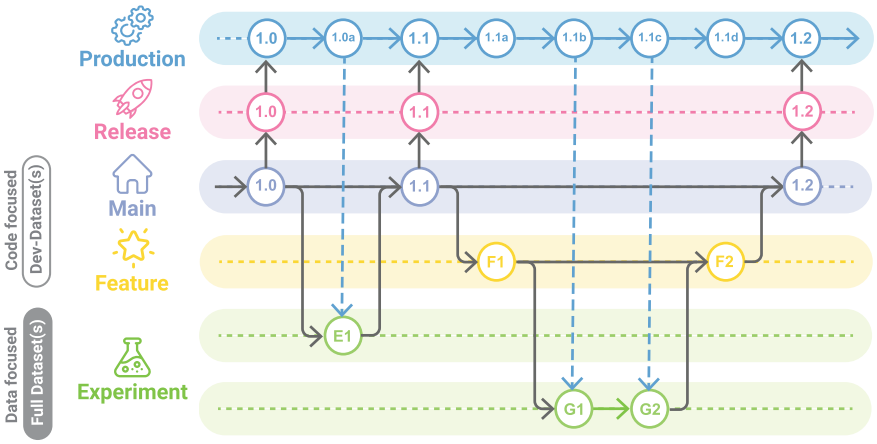

Fig. 5. Git Workflow with Production Environment.

traceability and repeatability of experiments. To make the concept more concrete, we continue the example, where the development team decides to go live with version (1.0).

4.5 Production Environment

The branch referred to as **production branch** is defined as the mirrored, deployed code version of a branch that references a possibly newer version of the corresponding data artifacts. This state is achieved by the consistent use of runners that, in the context of continuous training, are used to transfer the data artifacts to the data server after execution of the ML pipeline and to transfer the version reference of the hashes back to their production branch. Depending on the purpose of the application, this can result in this branch potentially running ahead of all branches under development.

Build upon the example of experiments presented in Sect. 4.3, focusing on the colored blue branch and demonstrating that the experimental branches can be used to reconfigure the ML pipeline. This approach is evident in Fig. 5, where the development team directly branches off a new branch (E1) from the main branch into the experiment namespace to modify the configuration file. Now, we utilize the previously mentioned synergy between the caching function of the data versioning tool and the reuse of artifacts in the other direction. When a runner is triggered in an experiment branch, it can fetch the current version of the data reference file from the production branch and download all existing data artifacts from the data server.

The code of the experiments will be executed with the actual data, providing the development team the opportunity to make decisions based on consistently traceable and reproducible results. As mentioned, runners triggered by the transmission of an experiment branch should, by default, activate an experiment tracking service. This workflow makes the development team's decision to reconfigure transparent and reproducible at any time.

Implementing simulations in the production environment takes it further, especially in applications with active learning or online ML. Here, it is not enough to switch from development datasets to complete datasets. Rather, the runner of the experiment branches must be able to verify the updated, published data version. This process is depicted from commit (F1), where the development team, for instance, implements a new model architecture.

The data scientist initiates a branch at (G1) to evaluate performance under production conditions into the experiment namespace. This time, an automation is activated, ensuring that the runner always uses the latest version from the production environment. This mechanism allows for realistic tests with current data, thus providing immediate insight into the performance of the current system version under real conditions. The experiment branch can coexist parallel to ongoing development. Suppose the team decides to make further changes to the code. In that case, the branching workflow remains the same: a merge request is derived from (F2), and the newer version of the ML configuration is propagated through the merge in the main branch as version (1.2), which is then provided via the release branch in the production branch.

5 Case Study: Active Learning

As part of the proposed develop methodology in [31], we introduced the concepts described and the Git workflow in the context of Active Learning (AL) projects. AL describes a technique where a ML model selects the most beneficial data for training, using new feedback to improve model performance [29]. The collaborative method is one of a popular task in data-centric AI development, with the goal of making the labeling process effective [40].

Within a three-year research program, an AL framework called LIFEDATA was developed where the presented concepts were followed and applied in two use cases[2] Before we evaluate and discuss the results, we start again with the process view and present the lifecycle of an AL project for a better overview.

[2] The LIFEDATA Framework, implemented in Python, is available at: https://www.github.com/ds-lab/lifedata.

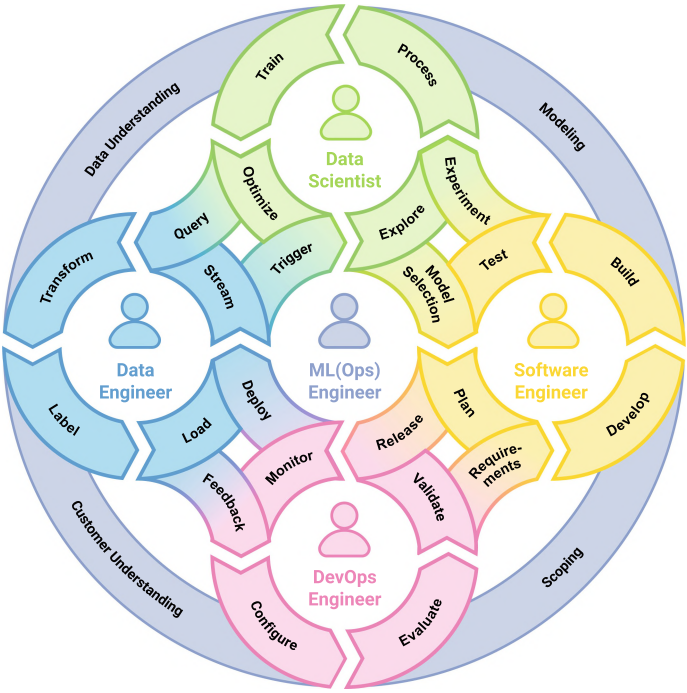

Fig. 6. Active Learning Lifecycle in project context with associated developer key roles.

5.1 Active Learning Lifecycle

The structure of the AL lifecycle, as depicted in Fig. 6, harmonizes established practices derived from the fields of DevOps, Machine MLOps, and DataOps. These practices are integrated within an AL loop and contextualized within the framework of a project. The primary cycle includes the training of the ML model, the development phase, and operational tasks specifically tailored to the unique demands of an AI project.

The blue-colored data iteration phase of the AL lifecycle corresponds with the responsibilities traditionally assigned to a data engineer. The three key stages of the Extract, Transform, and Load (ETL) process are systematically executed within this phase. The extraction step is further subdivided into batch and stream-based processes in accordance with the specific requirements of the AL query scenarios. During the transformative phase, data quality is enhanced by implementing rule-based tasks to mitigate potential inconsistencies or inaccuracies. The Label phase, unique to an AL project, involves annotating new data instances by a human or automated system, referred to as an "oracle". The final step, Load, delivers the processed data in a standardized format for integration into other stages of the AL lifecycle.

The ML iteration, colored in green, incorporates procedures generally within the purview of a data scientist. The necessary ML pipeline is typically comprised of diverse substeps, which may differ based on the problem domain- and application specifics. These tasks may be executed periodically, triggered by specific events, or conducted experimentally, each catering to the differing demands of the project. It commences with the exploration phase, where the data required for the model is subjected to detailed analysis. This is followed by the process step, in which data preprocessing is conducted, subsequently leading to the model's training. The optimization phase entails tasks such as tuning hyperparameters and adjusting the ML pipeline by implementing suitable methodologies. An ML iteration is typically triggered by initiating an experimental procedure, or in the case of automated systems, through the periodic intervals or occurrence of a specific event.

The development phase (yellow) encapsulates the principal responsibilities of a software engineer. Implementation is initiated once planning is completed or when new requirements emerge from the operational phase. After a successful build phase, the testing phase ensues, encompassing both traditional and ML-specific tests. This testing is conducted in an automated, continuous manner, adhering to the principles of CI, CD and CT.

Upon release, the pink-colored operations phase is initiated, serving as the bridging link between the development- and the production environment. This phase encompasses tasks related to monitoring the currently deployed model. Continuous reconfigurations, evaluations, and validations are integral components of maintaining a productive AL system.

Viewing the AL lifecycle clockwise, an AL iteration, in the form of an increment or a new model version, can be observed. However, when viewing counterclockwise, the connectivity of the respective phases and roles becomes explicitly apparent, highlighting the cyclic and interconnected nature of the AL lifecycle.

5.2 Use Cases

During the implementation of LIFEDATA, we applied the concepts presented in Sect. 3 and the Git workflow outlined in Sect. 4 to projects in the following two life sciences use cases, each built with separate ML pipelines. This domain poses challenges to AI projects regarding data annotation and serves as an exemplary field to demonstrate data-centric AI development.

Project A) Skin Image Analysis. Skin cancer is globally one of the most prevalent forms of cancer, where early detection significantly impacts patient survival. Various ML challenges from the International Skin Imaging Collaboration (ISIC) have shown encouraging results for AI-supported skin cancer diagnosis [4]. This technology could serve as a support system in clinical decision support systems (CDSS) for medical professionals.

Implementing this use case tackles the challenge of classifying skin lesions from image data using deep learning techniques, considering a constraint to

single-label classification. Our ML pipeline was implemented with two deep neural networks (DNN) classifiers, DenseNet [11], and MobileNet [10], which were trained on the seed data from HAM10000 [35], a publicly available collection of dermatoscopic images, comprising 10,015 annotated JPEG files. An excerpt of results from this case study can be found in [32].

Project B) ECG Classification. Cardiovascular diseases are the most common cause of death, so early detection and treatment of cardiac arrhythmias are crucial for improving treatment outcomes. AI methods have been used to develop CDSS for, e.g., heart monitoring, aiming to enhance diagnostic accuracy and efficiency. The research community has developed various ML models that expect different data types as input. A notable example is PhysioNet, which in recent years has focused on exploring algorithms for the classification and analysis of electrocardiograms (ECGs) [9].

We applied LIFEDATA to this use case and implemented an ML pipeline with a ResNet-DNN classifier to predict cardiac anomalies in 12-lead ECG recordings, as suggested in [38]. In this example, we used time-series data provided by the PhysioNet Challenge 2021 [24], which comes from multiple sources and includes 131,155 ECG recordings with one or more labels.

Moreover, in this use case, we introduced a productive environment, which included a web user interface in which a team of cardiologists could annotate additional yet unlabeled data selected by the training model. The retraining of the model was executed on a nightly schedule, resulting in a new query batch of samples provided to the web server.

5.3 Evaluation

Complementary to the evaluations already conducted in [31], which scrutinized the developmental principles for their adherence to established best practices from the relevant literature, and incorporated the viewpoint of industry experts through semi-structured interviews, our aim in this article is to provide a particular practical assessment and to scrutinize the two projects previously introduced.

First, we analyze the technical specifications, which were maintained consistently across both projects. For Project A and Project B alike, GitLab served as our choice for SCM. The CI/CD/CT runners were hosted on a GPU server. In terms of data versioning, we utilized Data Version Control (DVC) [12], a robust tool designed to handle large-scale datasets and machine learning models, and implemented MLflow [39] to track and manage our experimental endeavors tasks.

In Project A, where the ML pipeline for skin image analysis was implemented, two Nvidia Quadro RTX 6000 GPUs were available for model training, capable of parallel processing various tasks, and exclusively available to the team during implementation. The data remote for storing raw data and all artifacts, such as preprocessed data and trained models, was realized using a file server and SSH access.

A total of 16 developers contributed to this project. The experienced core team consisted of five developers of various roles who applied the described

development method for the first time. We set the size of the development dataset to 100 samples. This amount was large enough to contain instances of each class and to realize training/test splits, yet small enough to execute the ML pipeline code on the developers' client computers in a reasonable amount of time.

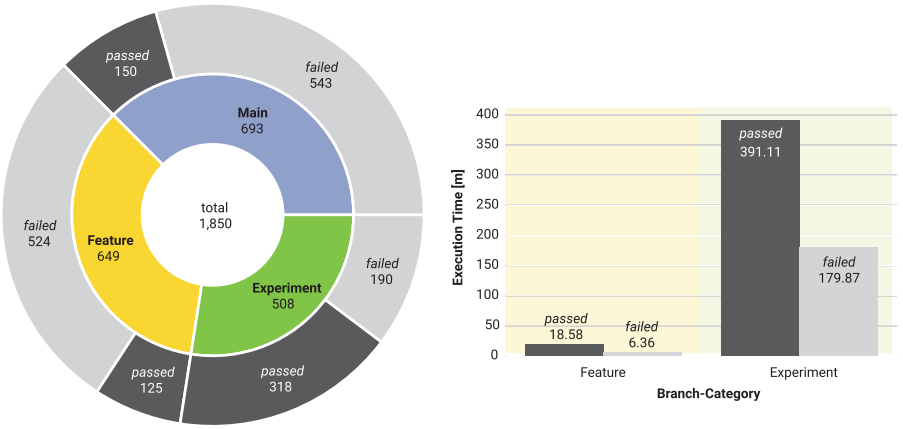

Fig. 7. Overview of CI-pipelines of Project A) Skin Image Analysis.

The analysis of the CI-pipelines in Fig. 7 shows that a total of 1,850 jobs were executed in a period of 12 months, with the majority being triggered in feature branches or the main branch. As the right part of the figure illustrates, using the development dataset proved advantageous.

Developers received feedback on whether the pipeline had failed after an average execution time of 6.36 min. In the case of a successful pipeline, the results were available after an average of 18.58 min. Noteworthy is the significant time saving compared to the average execution time of the experiment branches, which was 179.87 min (factor 28) for failed executions and 391.11 min (factor 21) for successful executions.

We adapted the infrastructure in Project B (ECG classification) to meet the increased requirements. This time, the team had four GPUs - two Nvidia Quadro RTX 6000 and two Nvidia Quadro P6000 - exclusively at their disposal, which could also process various tasks in parallel. An S3 bucked was used as the data remote, which was used for storing and exchanging raw data, preprocessed data, trained models, and other artifacts. The deployment was realized via Kubernetes.

A total of 19 developers contributed to this project. The experienced core team remained unchanged compared to Project A and followed the described development method again. In this project, the development dataset's size was 100 samples, which were compiled from different data sources and met the stated requirements.

Figure 8 shows the analysis of the CI-pipelines over an observation period of 18 months. A total of 3,048 jobs were triggered, including automated nightly

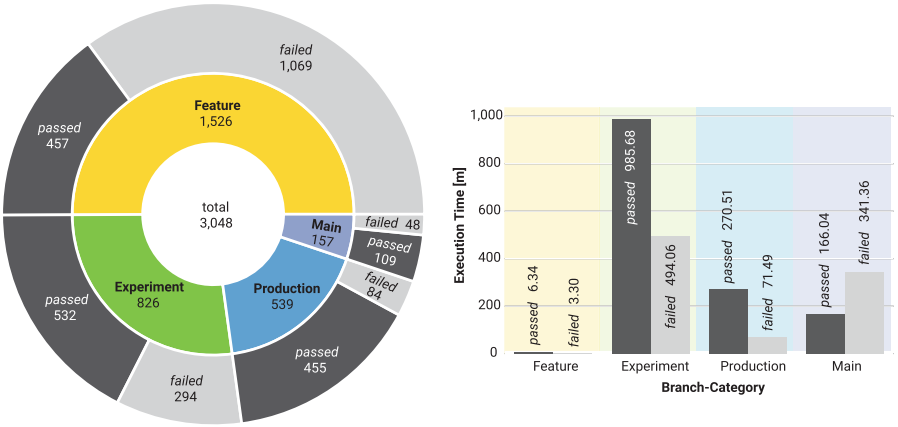

Fig. 8. Overview of CI-pipelines of Project B) ECG Classification.

retraining of the model on the production branch. For this project, we also listed the execution time of the jobs on the main branch, as it provided for the execution of integration tests and demonstrated the positive effects in terms of resource savings with the use of development datasets.

Although the runners in both the production and experiment branches each checked out the entire datasets, there is a clear difference in execution time. This can be explained on the one hand by the use of experiments for more complex simulations and on the other hand by the fact that the data base in the production environment did not necessarily change daily, which allowed the already trained model to be deployed unchanged.

Concerning feedback cycles, there was again a significant time-saving. When implemented in a feature branch, the average duration was 3.30 min (a factor of 149 compared to experiments) for failed and 6.34 min (a factor of 156 compared to experiments) for successful CI-pipelines.

5.4 Discussion

In both projects, adapting the introduced development method has proven beneficial with regard to agility and team collaboration. Due to the project and code structure, several developers could simultaneously implement new features and initiate experiments. Including the development dataset and the different runners resulted in significant differences in the execution times of the CI pipelines, enabling rapid iteration during the code-focused implementation and substantial resource savings throughout the project.

However, some aspects of development practices cannot be dismissed. On the one hand, there are substantial requirements for the development dataset, as its creation can be challenging given often very heterogeneous data sources. It may also become necessary to maintain different development datasets of

various sizes. Furthermore, the other principles, such as using runners for running experiments or modular code practice, represent an inevitable overhead for a team at the outset.

The team requires a member capable of setting up the necessary infrastructure, and every team member must be familiar with both code and data version control systems. Developers or data scientists unfamiliar with a data version control system will face a steep learning curve and common pitfalls, such as overly large merge requests. Feature branches likely require a different configuration than the best model, e.g., fewer training epochs to facilitate the quick feedback loop. These two configurations require additional management.

In addition to improving traceability, teams can benefit from the rapid feedback cycles once the concepts are established. Transparent collaboration is enabled, which, combined with established best practices from agile software development, leads to positive effects in the project.

6 Conclusion

The development methodology presented in this paper offers a structured, agile, and team-oriented approach to developing data-centric AI projects. Alongside established process models for data-intensive projects, we have emphasized the importance of MLOps principles for the effective realization of AI projects throughout the lifecycle. Our work highlights the need for solid infrastructure, establishing a common code and project structure, and implementing an effective Git workflow. The focus of the branching concept involves the implementation of CI/CD/CT runners, as well as specific branching namespaces. Through the guidelines presented, teams are supported in realizing data-centric AI projects.

In a case study, we demonstrated how these concepts can be successfully applied to practical AL projects. Using two specific use cases, we showed how the development methodology was implemented and elucidated the potential in both projects. The integration of development datasets and various runners has resulted in significant differences in the execution time of CI pipelines, leading to faster iteration during the code-centric implementation and considerable resource savings throughout the project. Although the results show positive impacts on data-centric AI projects, challenges require further research. In particular, developing and managing development datasets require additional attention, work, and expertise from developers.

Moreover, setting up the necessary infrastructure represents a substantial initial investment, and every team member needs to familiarize themselves with the systems for code and data version control. Future work could focus on optimizing the methodology concerning these challenges to reduce the setup and maintenance requirements for developers. Overall, our work demonstrates that despite the challenges associated with data-centric AI development, the proposed methods and techniques can improve the efficiency and effectiveness of AI projects. We believe that our work makes a significant contribution to advancing the cultural shift in data-centric AI development teams and hope that it will serve as a valuable resource for other researchers and practitioners.

Acknowledgements. This work was funded by the German Federal Ministry of Education and Research (BMBF) under reference number 031L9196B.

References

1. Amershi, S., et al.: Software engineering for machine learning: a case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291–300. IEEE, Montreal, QC, Canada (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00042
2. Arpteg, A., Brinne, B., Crnkovic-Friis, L., Bosch, J.: Software engineering challenges of deep learning. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 50–59 (2018). https://doi.org/10.1109/SEAA.2018.00018
3. Breck, E., Cai, S., Nielsen, E., Salib, M., Sculley, D.: What's your ML test score? A rubric for ML production systems. In: 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain (2016)
4. Celebi, M.E., Barata, C., Halpern, A., Tschandl, P., Combalia, M., Liu, Y.: Guest editorial skin image analysis in the age of deep learning. IEEE J. Biomed. Health Inform. **27**(1), 143–144 (2023). https://doi.org/10.1109/JBHI.2022.3227125
5. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery in databases. AI Mag. **17**(3), 37 (1996). https://doi.org/10.1609/aimag.v17i3.1230
6. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: Knowledge discovery and data mining: Towards a unifying framework. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. KDD'96, pp. 82–88. AAAI Press (1996)
7. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4), 1–37 (2014). https://doi.org/10.1145/2523813
8. Giray, G.: A software engineering perspective on engineering machine learning systems: state of the art and challenges. J. Syst. Softw. **180** (2021). https://doi.org/10.1016/j.jss.2021.111031
9. Goldberger, A.L., et al.: PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. Circulation **101**(23) (2000). https://doi.org/10.1161/01.CIR.101.23.e215
10. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 [cs] (2017). arXiv: 1704.04861
11. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2261–2269. IEEE, Honolulu, HI (2017). https://doi.org/10.1109/CVPR.2017.243
12. Iterative: DVC: Data version control - git for data & models (2020). https://doi.org/10.5281/zenodo.7848331
13. Jakubik, J., Vössing, M., Kühl, N., Walk, J., Satzger, G.: Data-centric Artificial Intelligence (2022). arXiv:2212.11854 [cs]
14. John, M.M., Olsson, H.H., Bosch, J.: Towards MLOps: a framework and maturity model. In: 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, Palermo, Italy (2021). https://doi.org/10.1109/SEAA53835.2021.00050

15. Karamitsos, I., Albarhami, S., Apostolopoulos, C.: Applying DevOps practices of continuous automation for machine learning. Information **11**(7) (2020). https://doi.org/10.3390/info11070363
16. Kreuzberger, D., Kühl, N., Hirschl, S.: Machine learning operations (MLOps): overview, definition, and architecture. IEEE Access **11**, 31866–31879 (2023). https://doi.org/10.1109/ACCESS.2023.3262138
17. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under concept drift: a review. IEEE Trans. Knowl. Data Eng. (2018). https://doi.org/10.1109/TKDE.2018.2876857
18. Lwakatare, L.E., Crnkovic, I., Rånge, E., Bosch, J.: From a data science driven process to a continuous delivery process for machine learning systems. In: Morisio, M., Torchiano, M., Jedlitschka, A. (eds.) PROFES 2020. LNCS, vol. 12562, pp. 185–201. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64148-1_12
19. Microsoft (2020). https://learn.microsoft.com/en-us/azure/architecture/data-science-process/overview
20. Mäkinen, S., Skogström, H., Laaksonen, E., Mikkonen, T.: Who needs MLOps: what data scientists seek to accomplish and how can MLOps help? (2021). arXiv:2103.08942 [cs]
21. Paleyes, A., Urma, R.G., Lawrence, N.D.: Challenges in Deploying Machine Learning: a Survey of Case Studies (2022). https://doi.org/10.1145/3533378
22. Polyzotis, N., Zaharia, M.: What can Data-Centric AI Learn from Data and ML Engineering? (2021). arXiv:2112.06439 [cs]
23. Renggli, C., Rimanic, L., Gürel, N.M., Karlaš, B., Wu, W., Zhang, C.: A Data Quality-Driven View of MLOps (2021)
24. Reyna, M.A., et al.: Will two do? Varying dimensions in electrocardiography: the PhysioNet/computing in cardiology challenge 2021. In: 2021 Computing in Cardiology (CinC). IEEE, Brno, Czech Republic (2021). https://doi.org/10.23919/CinC53138.2021.9662687
25. Ruf, P., Madan, M., Reich, C., Ould-Abdeslam, D.: Demystifying MLOps and presenting a recipe for the selection of open-source tools. Appl. Sci. **11**(19) (2021). https://doi.org/10.3390/app11198861
26. Rule, A., Tabard, A., Hollan, J.D.: Exploration and explanation in computational notebooks. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. ACM, Montreal, QC, Canada (2018). https://doi.org/10.1145/3173574.3173606
27. Sculley, D., et al.: 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. (2015)
28. Serban, A., van der Blom, K., Hoos, H., Visser, J.: Adoption and effects of software engineering best practices in machine learning. In: Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (2020). https://doi.org/10.1145/3382494.3410681
29. Settles, B.: Active learning literature survey. Computer Sciences Technical report 1648, University of Wisconsin-Madison (2009)
30. Siddique, U.: SafetyOps (2020). arXiv:2008.04461 [cs]
31. Stieler, F., Bauer, B.: Git workflow for active learning: a development methodology proposal for data-centric AI projects:. In: Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering, pp. 202–213. SCITEPRESS - Science and Technology Publications, Prague, Czech Republic (2023). https://doi.org/10.5220/0011988400003464

32. Stieler, F., Rabe, F., Bauer, B.: Towards domain-specific explainable AI: model interpretation of a skin image classifier using a human approach. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops 2021, 19–25 June 2021, Nashville, TN, USA, pp. 1802–1809 (2021). https://doi.org/10.1109/CVPRW53098.2021.00199
33. Studer, S., et al.: Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology (2021)
34. Tamburri, D.A.: Sustainable MLOps: trends and challenges. In: 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 17–23 (2020)
35. Tschandl, P., Rosendahl, C., Kittler, H.: The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data **5**(1), 180161 (2018). https://doi.org/10.1038/sdata.2018.161
36. Whang, S.E., Roh, Y., Song, H., Lee, J.G.: Data collection and quality challenges in deep learning: a data-centric AI perspective. VLDB J. **32**(4), 791–813 (2023). https://doi.org/10.1007/s00778-022-00775-9
37. Wirth, R., Hipp, J.: CRISP-DM: Towards a Standard Process Model for Data Mining (2000)
38. Xu, X., Jeong, S., Li, J.: Interpretation of electrocardiogram (ECG) rhythm by combined CNN and BiLSTM. IEEE Access **8**, 125380–125388 (2020). https://doi.org/10.1109/ACCESS.2020.3006707
39. Zaharia, M.A., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. **41**, 39–45 (2018)
40. Zha, D., et al.: Data-Centric Artificial Intelligence: A Survey (2023). arXiv:2303.10158 [cs]
41. Zhou, Y., Yu, Y., Ding, B.: Towards MLOps: a case study of ML pipeline platform. In: 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE), pp. 494–500. IEEE, Beijing, China (2020). https://doi.org/10.1109/ICAICE51518.2020.00102