LIFEDATA - a framework for traceable active learning projects

Fabian Stieler, Miriam Elia, Benjamin Weigell, Bernhard Bauer, Peter Kienle, Anton Roth, Gregor Müllegger, Marius Nann, Sarah Dopfer

Angaben zur Veröffentlichung / Publication details:

Stieler, Fabian, Miriam Elia, Benjamin Weigell, Bernhard Bauer, Peter Kienle, Anton Roth, Gregor Müllegger, Marius Nann, and Sarah Dopfer. 2023. "LIFEDATA - a framework for traceable active learning projects." In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), 4-5 September 2023, Hannover, Germany*, edited by Kurt Schneider, Fabiano Dalpiaz, and Jennifer Horkoff, 465–74. Piscataway, NJ: IEEE. https://doi.org/10.1109/REW57809.2023.00088.

LIFEDATA - A Framework for Traceable Active Learning Projects

Fabian Stieler[1,3,*], Miriam Elia[1], Benjamin Weigell[1], Bernhard Bauer[1,3],
Peter Kienle[2], Anton Roth[2], Gregor Müllegger[2], Marius Nann[2], Sarah Dopfer[2]
[1]Institute of Computer Science, University of Augsburg, Germany
[2]GS Elektromedizinische Geräte G. Stemple GmbH, Kaufering, Germany
[3]Center for Responsible AI Technologies
[*]Corresponding Author: fabian.stieler@uni-a.de

***Abstract*—Active Learning has become a popular method for iteratively improving data-intensive Artificial Intelligence models. However, it often presents a significant challenge when dealing with large volumes of volatile data in projects, as with an Active Learning loop. This paper introduces LIFEDATA, a Python-based framework designed to assist developers in implementing Active Learning projects focusing on traceability. It supports seamless tracking of all artifacts, from data selection and labeling to model interpretation, thus promoting transparency throughout the entire model learning process and enhancing error debugging efficiency while ensuring experiment reproducibility. To showcase its applicability, we present two life science use cases. Moreover, the paper proposes an algorithm that combines query strategies to demonstrate LIFEDATA's ability to reduce data labeling effort.**

***Index Terms*—Active Learning, Data Labeling, Traceability, Data-Centric AI, Python Framework, Open Source**

I. Introduction

Nowadays, the surge of interest in Artificial Intelligence (AI) systems is driving rapid advancements and the proliferation of Machine Learning (ML) methodologies. These learning techniques are capable of handling increasingly complex data structures. They are generating promising outcomes by learning to make decisions that were impossible, or at least hard to solve using traditional coding methods. However, training these algorithms usually requires large datasets where the labeling process presents various challenges.

Active Learning (AL), a prominent method addressing the issue of data labeling, holds the potential to render this cost-intensive and time-consuming process efficiently. By intelligently selecting instances for labeling, the aim is to enhance model performance and reduce the volume of necessary training data [41]. Furthermore, feedback from human-model interaction can guide the model in correctly learning relevant concepts, thereby increasing acceptance [19].

Despite the advantages, the currently available AL-frameworks frequently lack the provision of transparency and traceability necessary for a comprehensive understanding and replication of the learning process. The dynamic nature of artifacts that emerge along the AI life cycle, driven by continuous learning and evolving datasets, calls for the implementation of techniques for tracking data provenance, versioning code, data, and models, as well as employing methods for model interpretation and stakeholder-specific user interfaces. Traceability in the sense of consistent versioning of all artifacts is a significant MLOps principle that contributes significantly to the compliance and audibility of an AI system [29]. The significance of addressing this gap is further emphasized by the European Commission's requirement for transparency techniques in high-risk domains like finance, aerospace, mobility, and healthcare as part of their approach to building trustworthy AI [23].

This paper introduces LIFEDATA, an innovative framework for traceable AL projects. We provide open-source software consisting of a core framework and a project template, aiming to enhance transparency, explainability, and interactivity in AL projects. LIFEDATA presents a novel approach that empowers researchers and practitioners to comprehend the learning process, from instance selection to interpretable model output, thereby improving the overall understanding of the model's behavior. In Section III, we present the technical design of our framework and introduce a structured approach for AL projects. Subsequently, in Section IV, we showcase two real-world use cases where we applied our framework and demonstrate the feasibility of AL projects with LIFEDATA. Before delving into the technical details, we provide a brief overview of related work in Section II.

II. Background

The research community has previously introduced various open-source frameworks for AL. While some of these are specialized in implementing simulations [1], [46], others focus on providing a rich interface for annotating data [11]. The frameworks proposed in [50] and [13] are characterized by their modular architecture to realize a wide range of AL scenarios. To better understand the methodology, let's take a closer look at AL paradigms.

A. Active Learning

The basic concept behind AL is that a learning algorithm, through intelligent selection of instances for labeling, can lead to a desired outcome, such as an enhancement in predictive performance [41]. This methodology stems from the inherent challenges of supervised learning that necessitate a substantial volume of labeled training data. Within the context of ML,

the acquisition of such data is generally regarded as the most labor-intensive and cost-prohibitive stage.

In AL, there are two main scenarios for data labeling. First, pool-based sampling, where the model, termed 'Learner', selects data points from an unlabeled dataset for labeling by the 'Oracle', or annotator [30]. The second scenario, stream-based sampling, assesses each data point at runtime to determine if its label should be requested from the oracle [12].

Central to this setup is the query strategy (QS), which can involve individual batches or a data stream. Usually, three opposing forces influence the decision which unlabeled samples best generate the desired effect on the model, regardless of the concretely applied QS: informativeness, representativeness [15], and diversity. Various QSs have been proposed, including Uncertainty Sampling [4], which selects the instances where the model is most uncertain. Other approaches use decision theory, calculate samples' information density [48], or employ multiple models [42].

Recent developments have broadened the scope of AL from instance labels to include additional knowledge queries [19]. However, there's a tendency within the ML community to treat annotators as error-free oracles, likely resulting from testing AL algorithms mainly with simulated human input.

Automatic labeling or semi-supervised learning involves using the model as an oracle [53]. After initial human annotations guide the model to achieve a certain accuracy, it can assign labels automatically. But, human involvement remains vital in some domains to create a large ground truth dataset and identify edge cases for quality assurance, justifying the combination of automatic and manual methods [14].

Literature shows that labeling errors are inevitable, even when dealing with simple perceptual tasks [10], [33]. Usually, the real-world applications targeted are those where high-quality labels are expensive to obtain due to knowledge barriers or the effort involved in labeling. Additionally, there are applications where determining the ground truth is not trivial, and different oracles can come to different results while labeling the same sample [6], [7], [26]. Using AL to improve label quality has also been investigated in several studies with promising results [6], [52].

Given this context, it becomes paramount to incorporate data provenance within an AL project design. Especially when multiple annotators are involved in the labeling process, and the model is continuously learning.

B. Traceability in AI Projects

In AI projects, various artifacts, such as the trained model, emerge throughout the entire life cycle. Unlike in traditional software projects where the source code suffices to create the artifacts, the reproduction of artifacts in AI projects requires the source code and the data as input [40]. Accounting for the traceability of these artifacts enables various stakeholders in an AI project to understand the decision-making process of AI systems, thereby fostering trust and increasing acceptance.

On the one hand, this capability significantly facilitates the debugging and improvement of trained models. For instance, if a model produces unexpected output, developers can trace the decision-making process back to the source of the cause, which in turn simplifies model fine-tuning. On the other hand, regarding accountability, traceability should ideally be considered from the onset of training data generation and its annotations [20]. In this context, it is important to maintain an overview of which data instances were involved in which training iterations.

When meticulously recording data provenance (i.e., the origin of the training data and the resulting artifacts along the AI life cycle), the versioning of code and data represents a key aspect in achieving traceability [38]. Besides established source code control systems, techniques such as data repositories and metadata tracking are utilized, which capture information about the data collection process as well as changes in the data, model configurations, and performances [27]. This ensures the reproducibility of results, even if the model is iteratively re-trained with new training data [36].

Moreover, traceability enhances transparency by contributing to the clarity of AI systems. A detailed understanding is critical in high-risk domains where the consequences of AI decisions can be severe. In this context, the interpretability of model outputs is likewise relevant, which can be improved through techniques of explainable AI (XAI). This subfield of AI encompasses techniques that aim to render the decisions of an AI understandable to humans. As more complex and opaque models are prevalent in development, XAI methods are gaining growing importance. [5]

Utilizing these methods increases the transparency in AL projects, thereby providing essential insight into these intricate models. Studies, such as those conducted by [39] and [19], delve deeper into this topic. Their investigations suggest that integrating XAI into an AL loop contributes to the calibration of trust annotators in the ML model. By providing a comprehensive explanation of the model's decisions, XAI aids in building a nuanced understanding, thereby fostering an environment of informed trust.

Such an approach enhances humans' confidence in the model. It ensures that trust is based on a clear understanding of the model's strengths and limitations, paving the way for more reliable and effective collaboration between human annotators and AI models. To this end, we cast the presented concepts into an AL framework, which is presented in the next Section.

III. Framework Design

Due to the vast spectrum of emerging application fields for data-centric AI, projects are shaping up for various ML tasks and algorithmic solution approaches. In order to achieve domain independence of LIFEDATA, the primary approach comprises decoupling a specific ML project from the underlying framework and linking the execution of the ML pipeline in the form of a service. This universal accessibility offers two significant advantages: First, LIFEDATA is independent of the ML task, allowing users to implement algorithms and QSs. Secondly, proven concepts of data versioning in ML projects can be leveraged for traceability.

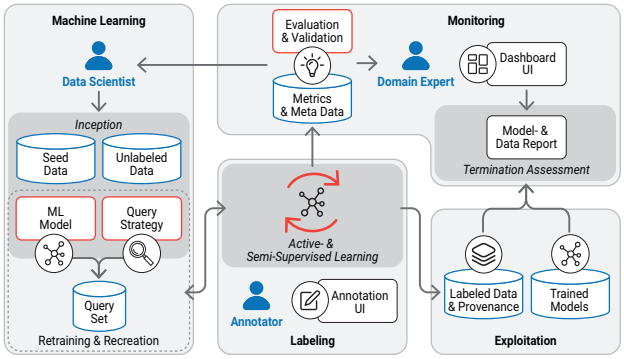

Fig. 1. Process Outline in an Active Learning Project with LIFEDATA

To the best of our knowledge, LIFEDATA is the first open-source framework for AL projects that integrates traceability aspects. Our concept consists of two key elements: The core framework, presented in III-B, comprising various interfaces, a database, and a web-based annotation UI. Additionally, we provide a project template, presented in III-C, that can be used by LIFEDATA users as a kick-start to create an AL project. To gain a comprehensive understanding of the workflow of AL projects using LIFEDATA, we initially provide a process-oriented perspective.

A. Overall Process

Before an AL project can commence, the underlying scope must be defined. The literature proposes varying process models for ML projects. The initial phases vary in specification, ranging from "Business & Data Understanding", "Requirements", to a basic "Data" phase [45]. Assuming these tasks are already completed, we illustrate in Figure 1 the idea of a LIFEDATA project, which depicts the overall process with its four essential activities and hints at the *Inception* as the starting point in the *Machine Learning* phase.

The data scientist or ML engineer begins by providing the unlabeled instances and potentially seed data, an initial set of already annotated data. Based on the ML model of the QS, or a combination of both, a selection of instances to be labeled, the query set, is generated. These are passed on to the oracle in the *Labeling* phase, which can be labeled by a human annotator in an Annotation UI or pseudo-labeled by the model itself. Newly emerged feedback are used to retrain the learning algorithm. This process can be reiterated as often as desired, which is indicated by the bidirectional arrow between these two phases.

From the labeling phase, two other outgoing arrows are visible. The *Exploitation* of an AL project with LIFEDATA is defined as the labeled data, its provenance as well as the trained models. Furthermore, the effects on the model are determined in each labeling iteration and made available for *Monitoring*. Metrics, e.g., the F_1-score, and meta-data are stored, allowing the data scientist and the domain expert to evaluate the prediction performance and validation, for instance, using XAI methods such as Shapley Additive Explanations (SHAP) [31]. This information continuously flows into a Dashboard UI, from which the *Termination Assessment* can finally be derived through data and model reports.

B. Core Framework

In the following, we transform the process view into an reference architecture, which is displayed in Figure 2 and depicts its two key elements, the core framework, and the project instance. Essentially, the LIFEDATA core framework is built in four layers and provides two interfaces, which for ease of overview, are colored green. Starting with the top layer, we provide a more detailed introduction to each component to give a clear understanding of their functionalities and interrelationships within the overall system.

1) User Management and Interfaces: AL projects typically involve multiple stakeholders with varying technical abilities. As mentioned in III-A, these actors can be data scientists, ML engineers, annotators as human oracles, and domain experts such as specialized physicians, highly qualified engineers, linguists, or financial experts. Each of these stakeholders has a different role within the project and requires access to different functionalities and tools.

For example, data scientists are responsible for building the ML models, while ML engineers are responsible for managing the software infrastructure required to support these models. Domain experts provide domain-specific knowledge and expertise.

To accommodate varying requirements of these stakeholders, LIFEDATA provides both a Command-Line-Interface (CLI) and a Web Application Programming Interface (API). The CLI serves as the data scientists' entry point and allows them the execution of various commands to initialize a project instance and set up the annotation interface. Other commands are useful for the *Experiments Utils*, which allow the implementations of simulations, for instance. The Web API provides a set of functions that ML engineers can use to integrate the *Annotation UI* and *Dashboard UI* and manage user identities as well as access.

2) Annotation Domain Layer: We placed an Annotation Domain Layer between the user-, and data-corpus. It serves as a bridge between these two entities and allows the modeling of associations between labels and samples.

The Annotation Domain Layer provides an abstraction that enables the definition of structural and semantic relationships between information about annotators, annotations, and samples. For instance, it provides the necessary logic to map information about new label addition, sample-annotator assignment, sample skipping, and additional label requests by a different annotator, to realize multiple-view setups as proposed in [32].

Events that pass their information to the subsequent layer are defined to ensure traceability. These events provide a record of the actions taken by users on the data corpus and are crucial for achieving the necessary data provenance of the annotations. Annotation provenance is important for

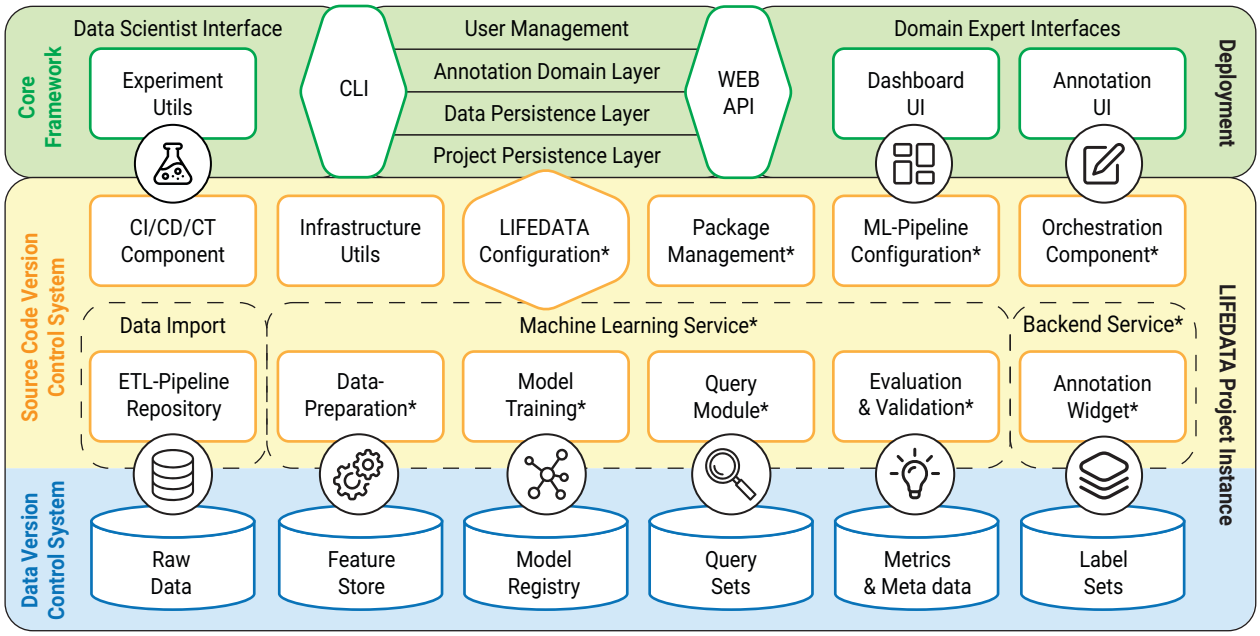

Fig. 2. Reference architecture of Active Learning projects using LIFEDATA. The structure of the core framework (green) is displayed above the project instance. All code modules and configuration files (yellow) are versioned by a source code version control system, while the linked artifacts (blue) are managed by a data version control system. * = Component is part of the provided project template.

several reasons, including quality control, reproducibility, and re-usability of training data in various ML projects.

3) Data Persistence Layer: AL projects involve dealing with fluctuating data. To ensure label traceability, storing the associated information is essential. In this sense, LIFEDATA provides a relational database whose connection management is implemented in the Database Persistence Layer. To ensure the database is reliable and scalable, we followed a container-based approach that isolates the database from the project-specific dependencies.

The main task is to ensure that all generated data, such as annotations, user information, sample information, and event logs, are stored in the database for long-term availability and reuse. This includes all I/O processes resulting from data flows triggered by actions in the user interfaces, as well as the provision of information in- and outside of the ML pipeline.

In addition, LIFEDATA provides functionality for versioned database migrations. If new requirements arise that require a change in the schema, these changes can be implemented during project runtime.

4) Project Persistence Layer: In order to guarantee seamless integration into any ML project, we have introduced the Project Persistence Layer. It provides a standardized interface that serves as a connection between the core framework and the ML project. Its primary components are functions for (re-) triggering the ML pipeline connected to the ML service. For instance, their invocation is necessary when new annotations need to be queried.

Additionally, this layer defines functions for handling the artifacts associated with the ML service's execution. These include exporting the current labels from the database, as well as importing the query set with information about which samples should be labeled next. These functions are summarized in the LIFEDATA API, and their specific implementation is located in form of configurations within the ML project.

C. Project Instance

The second essential part, alongside the core framework, is the project-specific structure which we summarize in Figure 2 as the LIFEDATA Project Instance. As illustrated, the framework demands all code modules to be managed by a *Source Code Version Control System*. The calculated artifacts, such as the trained model or the created query set for a specific database state, are managed by a *Data Version Control System*.

A project instance represents the specific implementation of an AL project based on LIFEDATA. For initial creation, LIFEDATA provides a project template that guides developers to adapt the structure to their specific use case. Concretely, Git is used as a distributed version-control system [9], and DVC is provided to synchronize the source code and the resulting data versions [28].

1) Components Overview: Our proposed AL project structure in Figure 2 includes several components that facilitate the development, deployment, and management of a LIFEDATA project. We propose to use a separate repository where the data engineering pipeline with Extract, Transform and Load (ETL) takes place. Once these steps are performed, there are several *Data Import* functions for data insertion.

```
#ml_config.py
RANDOM_STATE = 123

#...
class QUERY:
  QUERY_SET_FILE = "data/query/queryset.csv"
  QUERY_BATCH_SIZE = 200
  QS_STRATEGIES = ["uncertainty", "random"]
  QS_RATIO = [2,1]
  class UNCERTAINTY:
    CONFIDENCE_THRESHOLD = 0.9
    #...
```

Listing 1: Example configuration of the query stage

Infrastructure Utilities include tools and scripts for managing a project's infrastructure-related tasks. This comprises infrastructure provisioning, monitoring, and scaling resources such as GPU usage. They are at all times highly dependent on the hardware or applied cloud platform, just like libraries for model training, they have to be customized for each project.

For the project's software dependency management, the project instance provides a *Package Management*, which is realized by virtual environments. This ensures that the system is built with the correct versions regarding libraries and packages and ensures its consistency as well as reproducibility across different environments and platforms.

The *LIFEDATA Configuration* contains the project-specific implementation of functions, as described in Section III-B4. In the provided project template, these functions include the logic for retraining the model on a certain data version, which could happen when new labels are present in the training set after an annotation iteration, for instance. The execution of individual functions is managed by the *Orchestration Component*, which coordinates the various modules. It schedules tasks and monitors their progress, while the *CI/CD/CT Component* is responsible for automating the resulting artifacts' building, retraining, testing, and deployment. This includes the software as well as the AL-related artifacts, such as the query set and the trained model.

2) Machine Learning Service: A machine learning pipeline usually consists of different sub-steps executed in a specified order. It has become best practice to split these steps of a pipeline [35], hence this principle is followed by the LIFEDATA project template. The initial ML pipeline starts with separate stages for data preparation, model training, query, as well as evaluation and validation. The individual stages are agnostic to the underlying ML problem and can be extended with the respective logic.

Following the code principles of [43], the different steps of a ML pipeline are usually realized by individual code modules configurable via feature flags and parameters. We want to describe the implementation in more detail using the *Query Module* as an example.

Listing 1 demonstrates the relevant part of a *ML Configuration*. Here, a separate class is set for each stage of the ML pipeline. It specifies the existing parameters and defines paths for code and data artifacts. If certain parameters are to be assigned to a sub-module, following this structure and

```
vars:
- ml_config.py

stages:
  #...
  query:
    cmd: python myproject/query/query.py
    deps:
    - ${EXTERNAL_DATA.UNLABELED_FILE}
    - ${EXTERNAL_DATA.LABELED_FILE}
    - ${MODEL_TRAINING.MODEL_FILE}
    - myproject/query/
    outs:
    - ${QUERY.QUERY_SET_FILE}
    params:
    - ml_config.py:
      - QUERY
  #...
```

Listing 2: Query stage in ML-pipeline definition

defining them in an inner class is advisable. Parameters that apply across multiple stages are set outside each class, as illustrated by the example for `RANDOM_STATE`. Given the correct usage, this example ensures that the libraries imported (such as Numpy, Tensorflow or Pytorch) are initialized with the same random state, even if they are called by different modules.

This implementation allows access to a single configuration from each code module along the ML pipeline. As this is tracked by the source code version control system, a systematic comparison of different configurations and settings becomes possible at project runtime, while this provides the basis for consistent reproducibility of resulting artifacts later on.

To enable the automated execution of the ML pipeline in form of a *Machine Learning Service*, defining all tasks at all stages as well as their dependencies with their configurations of the ML pipeline is necessary. This definition manages the data versioning control system, specifies the input and output at each step, as well as possible interdependencies and thus ensures that the creation of all artifacts is transparent and reproducible. Some use cases may require customization of the initial ML pipeline design by adding or removing specific stages. In order to achieve this flexibility, these stages and their interdependencies can be identified by the Data Scientist and adapted to the ML pipeline definition.

We continue with the example of the query module as an essential part of the ML service. As such, it provides the selection of samples for annotation and enables the automatic generation of query sets. It uses one or more QSs to select samples whose annotation should lead to the intended model change. Typically, QSs leverage the models, which is why their execution occurs after the training is completed. This condition is illustrated in Listing 2, which displays the related snippet of the ML-pipeline definition.

The described structure manifests as follows: Given a valid ML Configuration, the individual stages are defined. The command to be executed is specified, as well as related dependencies in form of paths of code and data artifacts. In the case of the query module, these consist of files containing in-

formation about the samples, whether they are already labeled or unlabeled. Further, the trained model and the entire project folder of the related query module are required, containing all code files.

Computed artifacts consist of a list of samples for annotation selected by the QS and are stored under the corresponding output path. Furthermore, the class of the related stage from the ML configuration is specified. If a modification occurs in one of the specified artifacts produced by a particular stage, or if the reconfiguration requires the re-creation of individual artifacts, the related parts are calculated and saved by triggering the ML service. In more concrete terms, new annotations may arrive, leading to the model's re-training and, consequently, the query set's recreation.

This effect is achieved through the data versioning system and covers the entire ML pipeline in the LIFEDATA project instance. As a final stage, the *Evaluation and Validation* module is located, whereby the calculation of the model metrics is performed at the end of the execution. For a deeper analysis of the trained model, the implementation of XAI methods is included at this stage. These methods are intended to facilitate an interpretation of model predictions, and the generated outputs, akin to all other artifacts, can be stored with their respective versions. Consequently, these outputs can be utilized for an UI or for reporting purposes.

3) Query Merging: LIFEDATA's Query Module is highly customizable, and can be extended to include any QS. Section II-A introduced the three opposing forces that affecting the way which samples should be selected for annotation. To reconcile these forces, we introduce `QueryMerging`, a hybrid QS. For this purpose, the relevant QSs are first implemented in separate subcode modules. Then, as indicated in Listing 1, the necessary configuration is set, which includes the parameters mentioned in Algorithm 1. In the case of `QueryMerging`, individual batches are first queried for each of the selected QSs, and then merged according to the desired ratio. The resulting query set is proportionally shuffled, while considering the original rank of the individual samples in each query batch.

Once the query set is created, it is possible to generate annotations by the model through semi-supervised learning, as well as one or more human oracles. The query set is available to the *Annotation Widget* in the latter case.

4) Annotation Widget: In LIFEDATA, the annotation of samples by a human oracle takes place in a web-based user interface. This key feature allows for easy and efficient labeling of samples by multiple annotators. To ensure that LIFEDATA is label and data type agnostic, the annotation widget was designed to enable Data Scientists the implementation a wide range of pool-based AL scenarios.

The link between project instances and the core framework is realized through a REST API. The provided *Backend Service* offers an interface for receiving labels and transmitting samples to be displayed in the Annotation UI. The rendering for the respective data type can be implemented within the annotation widget in the project instance itself or replaced by a third-party app. Thus, seamless communication between the project instance and the core framework is given. Moreover, the flexibility to customize the rendering process ensures that users can tailor their annotation process to their specific requirements.

Algorithm 1: `QueryMerging`. The query set $\mathcal{B}$ of size n would be assembled using the set of query functions $\mathcal{Q}$ in ratio R. $\mathcal{U}$ is the unlabeled data pool and f is the model.

Input: $Q \leftarrow$ set of k query functions
$R \leftarrow$ set of query strategy ratios
$n \leftarrow$ size of the query set
$\mathcal{U} \leftarrow$ the unlabeled dataset
$f \leftarrow$ the model
Output: $\mathcal{B} = \{s_1, s_2, ..., s_n\}$
ensure: $0 < n \leq |\mathcal{U}|$
initialize: $b_1, b_2, ..., b_k \leftarrow \emptyset$ and $B \leftarrow \{b_1, b_2, ..., b_k\}$
foreach *query function* $q_i \in \mathcal{Q}$ **do**
  initialize: ${q_i}^f$
  // acquire samples to temporary batches
  $b_i \leftarrow {q_i}^f(\mathcal{U}, n)$
end
initialize: $j \leftarrow 0$, $\mathcal{B} \leftarrow \emptyset$
// assemble the query set
do
  // Choose samples from each batch based on the given ratio
  $i \leftarrow \arg\max R$
  $m \leftarrow R_i$
  $\mathcal{b} \leftarrow \emptyset$
  foreach *sample* $s_l \in b_i$ **do**
    // add samples to query set only once
    if $s_l \notin \mathcal{B}$ **then**
      $\mathcal{b} \leftarrow \mathcal{b} \cup s_l$
      $j \leftarrow j + 1$
    end
    // stop after sufficient sampling
    if $|\mathcal{b}| = m$ **then**
      break
    end
  end
  $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{b}$
  $R_i \leftarrow 0$
while $j < n$;

D. Usage

The LIFEDATA framework, implemented in Python, offers cross-platform compatibility. It includes an integrated database within a Docker environment for ease of setup and deployment and an adaptable React-based Annotations UI. A project template provides boilerplate code for Data Scientists, enabling initialization of project instances with required framework structures and customizations such as data and label types, ML

pipeline code, QS, and Annotation UI sample display logic. This facilitates project customization to individual needs.

We propose using LIFEDATA in an AL project, following the development methodology published in [43], which describes a minimal infrastructure setup, deployment, and automation to enhance project efficiency. Also proposed is the use of runners for scheduled re-training of the model, improving project execution.

IV. Evaluation

To demonstrate the feasibility of projects based on LIFEDATA, we applied the framework to two life science use cases. This domain provides AL projects with the following key challenges of data labeling:

a) **Imbalanced Data.** There is an unequal distribution of samples across different classes. Related challenges potentially affect the model and the annotation process, as the majority classes dominate the dataset. [2], [17]

b) **Costly Annotations.** A high level of qualification is required to annotate the data, often involving a long period of education for the annotator. This circumstance additionally increases the labeling costs, since the time of the human oracle represents a rare resource. [6], [18]

c) **High Quality Labels.** In addition to the specialized knowledge of domain experts, the degree of accuracy regarding the assigned annotations significantly influences the model's performance in the real-world. In many cases, the quality of the resulting model is directly tied to the quality of the labels it is trained on. [34]

The reproduction of ML artifacts is important for ensuring the integrity and transparency of the models. Therefore, traceability in such AL projects is of high importance. In addition, traceability promotes accountability and responsibility throughout the data labeling process, particularly regarding implementations of a multi-user labeling process, where multiple stakeholders contribute to the same datasets.

A. Skin Image Analysis

Skin cancer counts as one of the most prominent cancerous diseases worldwide, with early detection crucial for survival. Various ML challenges by the International Skin Imaging Collaboration (ISIC) have shown promising results for AI-based skin cancer diagnosis [8]. This technology could assist physicians in clinical decision support systems (CDSS). However, training these models requires significant image data annotated by dermatologists [22].

a) Implementation.: Adapting this use case addresses the problem of classifying skin lesions from image data through deep learning techniques with a single-label classification limitation. Our ML-pipeline was implemented with two DNN classifiers, DenseNet [25] and MobileNet [24], which are trained on seed data from HAM10000 [47], a publicly available collection of multi-source dermatoscopic images annotated by physicians. In this scenario, we simulated the human oracle by providing the true label through a lookup in the already labeled dataset. Further, various XAI methods have been implemented

Fig. 3. Comparison of different QSs: The ordinate shows the number of samples queried by the model with class "Nevus" (majority class) and "Melanoma" (one of the rare classes) in relation to the number of labeled training examples plotted across the abscissa.

to render the model more interpretable, such as a domain-specific approach demonstrated in [44].

b) Results.: The primary objective consisted of a simulation-based QS analysis while attempting to emulate a real-world setting as closely as possible. LIFEDATA's framework design supported experiment execution since computationally intensive steps, such as the preprocessing of image data, had to be performed only once, and the generated artifacts were evaluated with different configurations. Subsequently, it is possible to switch between the different model implementations and QSs, by adjusting the ML configuration.

The experiments conducted in this study demonstrate that AL can significantly reduce annotation effort in skin image use cases. By utilizing an appropriate QS, rarer classes, such as "Melanoma", are preferentially output to the human oracle for annotation at the beginning of the annotation phase, as shown in Figure 3. This approach ensures that the oracle provides the most critical annotations first, resulting in the most effective distribution of resources.

B. ECG Classification

As of now, cardiovascular diseases are the leading cause of death, which yields early detection and treatment of heart arrhythmia critical for improving patient outcomes. AI approaches have been prominently used in developing CDSS for cardiac monitoring, aiming to improve diagnostic accuracy and efficiency. Until now, the research community has developed various ML models expecting different data types as input. A popular example is PhysioNet, which in recent years focused on studying algorithms for electrocardiogram (ECG) classification and analysis [21].

a) Implementation.: We applied LIFEDATA to this use case and implemented a ML pipeline with a DNN classifier based on the ResNet architecture to predict cardiac abnormalities or anomalies in 12-lead ECG recordings, as proposed in [49]. In this example, a large amount of time series signal data is required for training the model. The

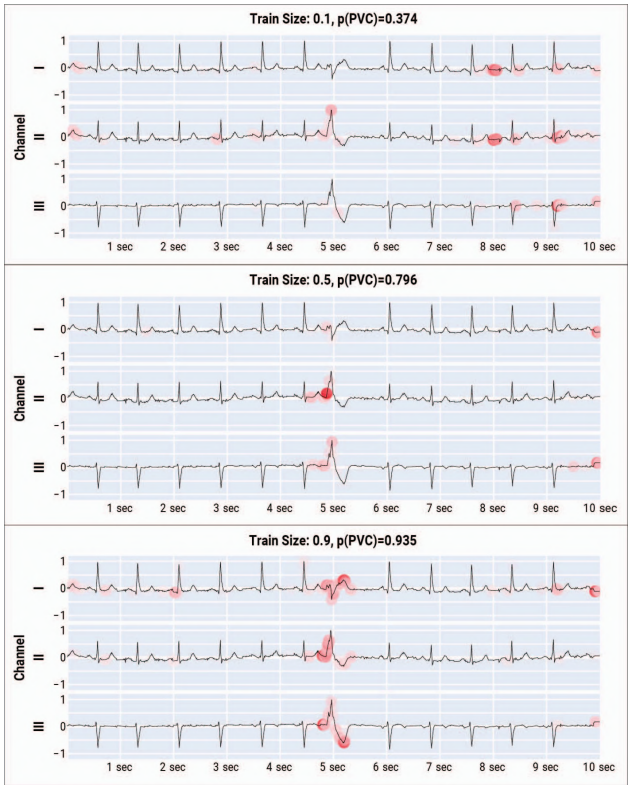

Fig. 4. Model interpretations using SHAP of sample "HR04123" in the course of training the model under supervision of domain experts. For more accessible representation, the sections depict the initial three channels of the 10-second ECG signal, the probability value p for the true class "Premature Ventricular Complex" (PVC), and the proportion of annotated training samples. Points in red denote high Shapley values, with the PVC class's distinctive beat around the 5-second mark.

data provided by the PhysioNet Challenges 2020/21 [3], [37] originate from multiple sources and contain ECG records with one or more labels, which is commonly known as a multi-label classification problem [16].

Similarly to the first use case, we were able to simulate the annotation process by lookup of true labels in the publicly available datasets. However, with gradually evolving experience, we collected additional feedback from real human annotators by providing a Web-UI to a team of cardiologists, so we shifted our investigation's focus to human feedback.

b) Results.: The customizable modules of LIFEDATA's ML pipeline enable the application of XAI methods for ECG classifiers as published in [51]. Further, the approach of end-to-end traceability and its resulting data provenance allows us to trace feedback (given label) to its origin (annotator) and thus investigate the impact of changes along the ML pipeline on human feedback and vice versa.

This approach uncovers valuable insights, fosters an environment of continuous improvement, and ensures that ML models evolve with feedback while maintaining transparency and accountability. Empirical results, as shown in Figure 4 demonstrate that domain expert feedback during iterative training amplifies medically significant concept consideration for class predictions. Systematic versioning facilitated by the XAI method allows for enduring, retrospective validation and analysis of model predictions across the entire AL lifecycle.

This case validates that an AL loop promotes a more robust model and increased acceptance. Traceability proves essential, linking created artifacts in both directions: while it aids domain experts in comprehending algorithm outputs, it also enables developers to trace back annotations to the source data that influenced model interpretation.

C. Discussion

The LIFEDATA framework has proven promising in addressing specific challenges within life science applications. However, it is not exempt from limitations that merit further investigation. As a central element of AL projects with LIFEDATA, traceability paves the way for tracing annotations back to their source, contributing to a more transparent, accountable process. This feature is significant in multi-user scenarios. However, this extensive traceability can also lead to increased complexity and higher administrative overhead, particularly for larger projects, massive datasets, and numerous contributors.

Viewing traceability within a traditional context, it is crucial to remember that it's not just an essential mechanism for maintaining transparency and integrity in AI systems. It also provides an audit trail, holding significant value for subsequent model refinement, stakeholder communication, and potential problem resolution. In this regard, LIFEDATA can be expanded to include concepts that provide stakeholders with a framework for a clear linkage of artifacts in more complex projects.

V. Summary

Especially in domains where often the knowledge of expensive experts is required for labeling training data, the continuous implementation of traceability is essential. To this end, we have introduced LIFEDATA, a publicly available Python framework that supports developers in realizing AL projects. Our framework proposes end-to-end traceability that starts with creating training data and ends with interpreting model results. Secondly, we have introduced `QueryMerging`, a hybrid QS integrated into LIFEDATA, making the annotation process more efficient.

The realization of AL projects with the proposed framework offers a transparent implementation for all stakeholders. Consistent versioning of code and data artifacts enables traceability and reproducibility of training results at any time. At the same time, the provided database for collecting information about the annotation process ensures continuous traceability.

With the introduction of LIFEDATA, we hope to contribute to the broader discourse on enhancing transparency and trust in AI and promote the adoption of traceable and explainable methods in AL projects. Future research can utilize LIFEDATA to reduce annotation effort on the one hand and gain human feedback for improving model performance and increasing acceptance on the other hand.

AVAILABILITY

The source code of LIFEDATA core framework is available on GitHub in the repository https://github.com/ds-lab/lifedata.

The project template to initialize a LIFEDATA project is available under https://github.com/ds-lab/lifedata-project-template.

ACKNOWLEDGEMENTS

This work was funded by the German Federal Ministry of Education and Research (BMBF) under reference number 031L9196B. We would like to thanks the participated physicians from the German Heart Centre of the Technical University of Munich during the ECG use case. Further, we would like to express our sincere thanks to Fabian Rabe for his contribution.

TECHNICAL SPECIFICATION OF EXPERIMENTS

The simulations conducted for evaluation purposes in the context of the two use cases each displayed unique computational requirements and timeframes, which we detail for better understanding.

In the Skin Image Analysis use case, mentioned in IV-A, we utilized two Nvidia Quadro RTX 6000 GPUs capable of processing different jobs in parallel. Considering the three random seed configurations, the computational load associated with this case totaled approximately 270 hours. This timeframe encompasses all tasks performed, starting with data preprocessing, model training, querying, and finally, the evaluation phase. Here, the benefits of data versioning became evident, as previously computed artifacts could be reused via LIFEDATA, thereby skipping certain different steps. We set the query set size to 400 samples, resulting in 20 iterations given a training size of 80% (8,000 samples).

The ECG use case (cf. IV-B) presented higher computational demands due to the larger volume of data. To efficiently handle this, we extended our computational resources to four GPUs. Specifically, we employed two Nvidia Quadro RTX 6000 and two Nvidia Quadro P6000, all operating in parallel. The total computational time for this use case amounted to about 940 hours, again considering three random seed configurations. As in the ECG use case, cached data from the data versioning tool saved computational time. In this scenario, we set the query set size to 2,500, leading to 30 iterations with a training size of 80% (75,000 samples).

REFERENCES

[1] Alexandre Abraham and Léo Dreyfus-Schmidt. Cardinal, a metric-based Active learning framework. *Software Impacts*, 12:100250, May 2022.

[2] Umang Aggarwal, Adrian Popescu, and Celine Hudelot. Active Learning for Imbalanced Datasets. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1417–1426, Snowmass Village, CO, USA, March 2020. IEEE.

[3] Erick A. Perez Alday, Annie Gu, Amit J. Shah, Chad Robichaux, An-Kwok Ian Wong, Chengyu Liu, Feifei Liu, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, Qiao Li, Ashish Sharma, Gari D. Clifford, and Matthew A. Reyna. Classification of 12-lead ECGs: the PhysioNet/Computing in Cardiology Challenge 2020. *Physiological Measurement*, 41(12):124003, December 2020. Publisher: IOP Publishing.

[4] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[5] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[6] Mélanie Bernhardt, Daniel C. Castro, Ryutaro Tanno, Anton Schwaighofer, Kerem C. Tezcan, Miguel Monteiro, Shruthi Bannur, Matthew P. Lungren, Aditya Nori, Ben Glocker, Javier Alvarez-Valle, and Ozan Oktay. Active label cleaning for improved dataset quality under resource constraints. *Nature Communications*, 13(1):1161, March 2022.

[7] Samuel Budd, Emma C. Robinson, and Bernhard Kainz. A Survey on Active Learning and Human-in-the-Loop Deep Learning for Medical Image Analysis. *Medical Image Analysis*, 71:102062, July 2021. arXiv:1910.02923 [cs, eess].

[8] M. Emre Celebi, Catarina Barata, Allan Halpern, Philipp Tschandl, Marc Combalia, and Yuan Liu. Guest editorial skin image analysis in the age of deep learning. *IEEE Journal of Biomedical and Health Informatics*, 27(1):143–144, 2023.

[9] Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

[10] Justin Cheng, Jaime Teevan, and Michael S. Bernstein. Measuring crowdsourcing effort with error-time curves. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 1365–1374, New York, NY, USA, 2015. Association for Computing Machinery.

[11] Rob Chew, Michael Wenger, Caroline Kery, Jason Nance, Keith Richards, Emily Hadley, and Peter Baumgartner. SMART: An Open Source Data Labeling Platform for Supervised Learning. *Journal of Machine Learning Research*, 20(82):1–5, 2019.

[12] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, May 1994.

[13] Tivadar Danka and Peter Horvath. modAL: A modular active learning framework for Python. *arXiv:1805.00979 [cs, stat]*, December 2018. arXiv: 1805.00979.

[14] Michael Desmond, Evelyn Duesterwald, Kristina Brimijoin, Michelle Brachman, and Qian Pan. Semi-Automated Data Labeling. *Journal of Machine Learning Research*, 133:156–169, 2021.

[15] Bo Du, Zengmao Wang, Lefei Zhang, Liangpei Zhang, Wei Liu, Jialie Shen, and Dacheng Tao. Exploring Representativeness and Informativeness for Active Learning. *IEEE Transactions on Cybernetics*, 47(1):14–26, January 2017.

[16] Ibrahim M. El-Hasnony, Omar M. Elzeki, Ali Alshehri, and Hanaa Salem. Multi-Label Active Learning-Based Machine Learning Model for Heart Disease Prediction. *Sensors*, 22(3):1184, February 2022.

[17] Seyda Ertekin, Jian Huang, and C. Lee Giles. Active learning for class imbalance problem. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 823–824, Amsterdam The Netherlands, July 2007. ACM.

[18] Beverly Freeman, Naama Hammel, Sonia Phene, Abigail Huang, Rebecca Ackermann, Olga Kanzheleva, Miles Hutson, Caitlin Taggart, Quang Duong, and Rory Sayres. Iterative Quality Control Strategies for Expert Medical Image Labeling. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, 9:60–71, October 2021.

[19] Bhavya Ghai, Q. Vera Liao, Yunfeng Zhang, Rachel Bellamy, and Klaus Mueller. Explainable Active Learning (XAL): Toward AI Explanations as Interfaces for Machine Teachers. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW3):1–28, January 2021.

[20] Boris Glavic, Vanessa Braganholo, and David Koop, editors. *Provenance and Annotation of Data and Processes: 8th and 9th International Provenance and Annotation Workshop, IPAW 2020 + IPAW 2021, Virtual Event, July 19–22, 2021, Proceedings*, volume 12839 of *Lecture Notes in Computer Science*. Springer International Publishing, Cham, 2021.

[21] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*, 101(23), June 2000.

[22] Marc Gorriz, Axel Carlier, Emmanuel Faure, and Xavier Giro-i Nieto. Cost-Effective Active Learning for Melanoma Segmentation, November 2017. arXiv:1711.09168 [cs].

[23] High-Level Expert Group on AI. *Ethics Guidelines For Trustworthy AI*. European Commission, 2019.

[24] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017. arXiv: 1704.04861.

[25] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, Honolulu, HI, July 2017. IEEE.

[26] Panagiotis G. Ipeirotis, Foster Provost, Victor S. Sheng, and Jing Wang. Repeated labeling using multiple noisy labelers. *Data Mining and Knowledge Discovery*, 28(2):402–441, March 2014.

[27] Richard Isdahl and Odd Erik Gundersen. Out-of-the-Box Reproducibility: A Survey of Machine Learning Platforms. In *2019 15th International Conference on eScience (eScience)*, pages 86–95, San Diego, CA, USA, September 2019. IEEE.

[28] Iterative. Dvc: Data version control - git for data & models, 2020.

[29] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *IEEE Access*, 11:31866–31879, 2023.

[30] David D Lewis, T Bell Laboratories, and Murray Hill. A Sequential Algorithm for Training Text Classi ers. page 10, 1994.

[31] Scott Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions, November 2017. arXiv:1705.07874 [cs, stat].

[32] Ionq Muslea, Steven Minton, and Craig A. Knoblock. Active Learning with Multiple Views. *Journal of Artificial Intelligence Research*, 27:203–233, 2006.

[33] An Nguyen, Byron Wallace, and Matthew Lease. Combining Crowd and Expert Labels Using Decision Theoretic Active Learning. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, 3:120–129, September 2015.

[34] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks, November 2021. arXiv:2103.14749 [cs, stat].

[35] Katie O'Leary and Makoto Uchida. Common Problems with Creating Machine Learning Pipelines from Existing Code. In *Workshop on MLOps Systems*, 2020.

[36] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program). *Journal of Machine Learning Research*, 22:1–20, 2021.

[37] Matthew A Reyna, Nadi Sadr, Erick A Perez Alday, Annie Gu, Amit J Shah, Chad Robichaux, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, Sardar Ansari, Hamid Ghanbari, Qiao Li, Ashish Sharma, and Gari D Clifford. Will Two Do? Varying Dimensions in Electrocardiography: The PhysioNet/Computing in Cardiology Challenge 2021. In *2021 Computing in Cardiology (CinC)*, pages 1–4, Brno, Czech Republic, September 2021. IEEE.

[38] Sheeba Samuel, Frank Löffler, and Birgitta König-Ries. Machine learning pipelines: Provenance, reproducibility and fair data principles. In Boris Glavic, Vanessa Braganholo, and David Koop, editors, *Provenance and Annotation of Data and Processes*, pages 226–230, Cham, 2021. Springer International Publishing.

[39] Patrick Schramowski, Wolfgang Stammer, Stefano Teso, Anna Brugger, Franziska Herbert, Xiaoting Shao, Hans-Georg Luigs, Anne-Katrin Mahlein, and Kristian Kersting. Making deep neural networks right for the right scientific reasons by interacting with their explanations. *Nature Machine Intelligence*, 2(8):476–486, August 2020.

[40] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[41] Burr Settles. Active Learning Literature Survey. page 47, 2009.

[42] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 287–294, New York, NY, USA, 1992. Association for Computing Machinery.

[43] Fabian Stieler and Bernhard Bauer. Git Workflow for Active Learning: A Development Methodology Proposal for Data-Centric AI Projects. *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 202–213, 2023.

[44] Fabian Stieler, Fabian Rabe, and Bernhard Bauer. 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*, pages 1802 – 1809, 2021.

[45] Stefan Studer, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus-Robert Müller. Towards crisp-ml(q): A machine learning process model with quality assurance methodology. *Machine Learning and Knowledge Extraction*, 3(2):392–413, 2021.

[46] Ying-Peng Tang, Guo-Xiang Li, and Sheng-Jun Huang. ALiPy: Active Learning in Python. Technical report, Nanjing University of Aeronautics and Astronautics, January 2019. arXiv: 1901.03802.

[47] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, 5(1):180161, December 2018.

[48] Tingting Wang, Xufeng Zhao, Qiujian Lv, Bo Hu, and Degang Sun. Density weighted diversity based query strategy for active learning. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 156–161, 2021.

[49] Xue Xu, Sohyun Jeong, and Jianqiang Li. Interpretation of Electrocardiogram (ECG) Rhythm by Combined CNN and BiLSTM. *IEEE Access*, 8:125380–125388, 2020.

[50] Yao-Yuan Yang, Shao-Chuan Lee, Yu-An Chung, Tung-En Wu, Si-An Chen, and Hsuan-Tien Lin. libact: Pool-based Active Learning in Python. *arXiv:1710.00379 [cs]*, October 2017. arXiv: 1710.00379.

[51] Dongdong Zhang, Samuel Yang, Xiaohui Yuan, and Ping Zhang. Interpretable deep learning for automatic diagnosis of 12-lead electrocardiogram. *iScience*, 24(4):102373, April 2021.

[52] Liyue Zhao, Gita Sukthankar, and Rahul Sukthankar. Incremental Relabeling for Active Learning with Noisy Crowdsourced Annotations. In *2011 IEEE Third Int'l Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third Int'l Conference on Social Computing*, pages 728–733, Boston, MA, USA, October 2011. IEEE.

[53] Xiaojin Zhu. Semi-Supervised Learning Literature Survey. 2008.