Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Context-aware Machine-Learning-based Error Detection

Ali Salaheddine

Course of Study:	Informatik
Examiner:	Prof. Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Daniel Del Gaudio, M.Sc.
Commenced:	March 1, 2023
Completed:	October 27, 2023

Abstract

The rapid growth of Internet-enabled devices in recent years has resulted in immense amounts of data. However, the data generated from this also carries an inherent risk, namely the presence of erroneous data. Such errors in the data lead to degradation of the underlying application. Eliminating errors from data is therefore a widely studied area of research. With the breakthrough, of machine learning, a new opportunity has been created to develop error detection systems. However, such systems require large amounts of labeled data to achieve the desired power. Since labeling data is an extremely repetitive and time-consuming task, such datasets are rarely available in practice. 'Active learning' as a subfield of machine learning manages to deal with this limitation. Using a small amount of labeled data, this involves training models for error detection that achieve promising results. In our work, we address how to improve the development of such models using information about the context in which the data was generated. We present three different pipeline approaches for developing error detection models, each based on a two-stage architecture: The first stage gathers context knowledge using a tool called RTClean, and the subsequent stage uses this knowledge to support error detection driven by active learning. Evaluation results showed that this two-stage structure could increase accuracy by 10-40% in optimal scenarios. However, certain cases showed significant performance deficits, indicating potential inefficiencies. A key finding was that RTClean needs to be improved to ensure the success of pipelines built upon it. In conclusion, the utilization of RTClean to enhance active learning-based error detection offers promising avenues for minimizing human intervention in the process.

Kurzfassung

Der rasante Zuwachs an internetfähigen Geräten in den letzten Jahren führte zu immensen Datenmengen. Jedoch bergen die daraus generierten Daten auch ein inhärentes Risiko, nämlich die Präsenz von fehlerhaften Daten. Solche Fehler in den Daten führen zu einer Beeinträchtigung der zugrunde liegenden Anwendung der Daten. Fehler aus Daten zu eliminieren, ist daher ein weit untersuchtes Forschungsfeld. Mit dem Durchbruch, des maschinellen Lernen wurde eine neue Möglichkeit geschaffen, Fehlererkennungssysteme zu entwickeln. Solche Systeme benötigen jedoch große gelabelte Datenmengen, um die gewünschte Mächtigkeit zu erreichen. Da das Labeln von Daten eine äußerst repetitive und zeitaufwendige Arbeit ist, sind in der Praxis nur selten solche Datensätze verfügbar. 'Aktives Lernen' als Teilgebiet des maschinellen Lernens schafft es, mit dieser Einschränkung umzugehen. Mithilfe von wenigen gelabelten Daten werden dabei Modelle zur Fehlererkennung trainiert, die vielversprechende Ergebnisse erreichen. In unserer Arbeit beschäftigen wir uns damit, die Entwicklung solcher Modelle mithilfe von Informationen über den Kontext in dem die Daten generiert wurden, zu verbessern. Wir präsentieren drei verschiedene Pipeline-Ansätze zur Entwicklung von Fehlererkennungsmodellen, die jeweils auf einer zweistufigen Architektur basieren: Die erste Stufe sammelt Kontextwissen mittels eines Tools namens RTClean, und die nachfolgende Stufe nutzt dieses Wissen, um die durch aktives Lernen gesteuerte Fehlererkennung zu unterstützen. Die Evaluierungsergebnisse zeigten, dass diese zweistufige Struktur in optimalen Szenarien die Genauigkeit um 10-40% steigern konnte. Bestimmte Fälle weisen jedoch erhebliche Leistungsdefizite auf, was auf mögliche Ineffizienzen hinweist. Eine wichtige Erkenntnis ist, dass RTClean verbessert werden muss, um den Erfolg der darauf aufbauenden Pipelines zu gewährleisten. Zusammenfassend lässt sich sagen, dass die Verwendung von RTClean zur Verbesserung der auf aktivem Lernen basierenden Fehlererkennung vielversprechende Wege zur Minimierung menschlicher Eingriffe in den Prozess bietet.

Contents

List of Figures

List of Tables

1 Introduction

The dawn of the 21st century witnessed an explosion in the amount of data generated, marking the inception of the 'Big Data' era [SS13]. With the build-up of internet connectivity, the rise of social media platforms, the growth of the Internet of Things (IoT), and increased digitalization of businesses, vast amounts of data are produced at unprecedented rates [ABW18]. By some estimates, 90% of the world's data was generated in the last two years alone, showing the large volume and velocity of data generation [AKG21]. The surge in the deployment of IoT devices across numerous sectors has undeniably transformed the way we collect, process, and interpret data [NRP+18]. However, with the large amount of data generated by these devices comes an inherent risk: the presence of erroneous data.

These errors arise for different reasons. From false sensor measurements, to data transmission glitches, integration errors, and even malicious tampering [BCC20] are all factors responsable for erroneous data. Such inaccuracies are not just minor discrepancies; when data is integrated into sophisticated systems, like Machine Learning (ML) pipelines, even a minor inaccuracy can cascade into significant computational errors or skewed interpretations. This becomes even more pivotal when considering the reliance on data accuracy for the development and fine-tuning of ML models [AKG21; GSDH17; LGEC17]. Advanced ML models, particularly deep learning architectures, are often compared to multi-storied buildings. The data serves as the foundation .If this foundation is unstable, i.e., because of poor quality, the entire structure risks collapse.

Human intervention for error correction, while effective to some extent, is severely limited by the large volume of data and the rapidity at which real-time applications operate. In the realm of Big Data, where datasets grow rapidly to the order of petabytes or more [ABW18], relying on manual or human-centric approaches for error detection becomes not only impractical but also grossly inefficient. First and foremost, the expansive volume of data is beyond human capacity to process in a reasonable timeframe [SS13]. Even if substantial human resources were dedicated, the effort would be time-consuming and susceptible to oversight due to the inherent limitations of human attention span and cognitive load. Moreover, the complexity of Big Data, characterized by its variety, velocity, and volume [AKG21; Sch22], means that errors can manifest in different ways, ranging from subtle inconsistencies to glaring anomalies. Expecting humans to discern such intricate patterns and anomalies amidst the vast ocean of data points is comparable to finding a needle in a haystack. Additionally, the fatigue and monotony of cleaning through such colossal datasets can increase the risk of human errors, ironically adding to the problem rather than solving it. In essence, while human intuition and expertise remain invaluable in many domains, the manual detection of errors in the Big Data landscape is an endeavor set up for incapability and failure.

Furthermore, the detection of erroneous data often necessitates specialized domain knowledge, which complicates scalable and broad implementation. Factors such as data complexity, contextual comprehension, risk of misinterpretation, discernment of subtle nuances, minimization of false positives, and the intricacies of interdependent relationships underscore the importance of domain expertise in error detection [ASW05; MCSG21; NKR20].

In recent years, ML-based error detectors have gained significant attention [AHS23]. These detectors, equipped with statistical pattern-recognition capabilities, offer robust results in error identification, even when tasked with analyzing vast datasets. However, the challenge lies in architecting ML systems that emulate the intricate insights of domain experts [KS22]. Even when architects overcome this challenge they are impeded by the necessity for vast, labeled datasets. Such data sets are rather rare in practice. Therefore, in reality, one often falls back on experts who label small amounts of data. However, this limits the scalability of such approaches because in general these ML models require large amounts of data. Thus, one needs a lot of labeled data from domain experts, which in turn takes a lot of time and limits the scalability [ASW05].

The field of active learning, which is a subfield of machine learning, reduces the amount of labeled data needed to train models [Set22]. A sophisticated model is trained with relatively little data that is labeled by an expert, also known as an oracle. The amount of data labeled by the Oracle is called the labeling budget. The goal is to develop the best possible model with the lowest possible labeling budget. However, the more the oracle labels, the better the models accuracy [AKG+14]. Active learning approaches for error detection have achieved promising results in the past [AHS23]. However, they are held back by the need for domain experts and long training times.

Furthermore, training ML models is often very time-consuming [GBC16]. Training of competent models can take from hours to weeks to months which highlights again how important it is to design this process in a scalable manner. In dynamic IoT ecosystems, where sensor locations might experience frequent alterations, retraining an ML model becomes impractical, particularly if the training process is too lengthy [Zho22]. The domain of context-aware computing has addressed the mutable nature of environments by introducing methodologies rooted in context models and ontologies [SAW94]. While these systems excel in incorperating expert knowledge, they, when deployed in isolation, may not deliver optimal results compared to other methods [AHS23; DSA23].

Consequently, our research aims to combine the strengths of context-aware error detection systems with active learning techniques, to address the challenge of scalable error detection in IoT-centric datasets.

Previous research has shown that RTClean [DSA23] achieves promising performance on the error detection task. RTClean is a context-aware system designed to harness the potential of context models for error detection [DSA23; GABS22]. It does so by leveraging Ontology Functional Dependencies (OFDs) derived from these context models. RTClean presents a novel approach, from the context-aware computing realm, to data error identification [BKC+17]. This thesis, thus, embarks on an exploration to further harness RTClean's potential, specifically focusing on its capability to suggest positive data instances for labeling, thereby optimizing the labeling budget of active learning-based error detectors. Additionally, we propose a method to use RTClean's provided positive instances with the aspiration to achieve more efficient and rapid feature generation on a well performing active learning-based error detector, namely ED2 [NMA19].

Our contributions include:

- An error detection pipeline that uses RTClean to pre-label and pre-detect errors, which are later used for ML-based error detection. This includes three approaches how design this pipeline in order to reduce the labeling budget and increase training times. The approaches are referred to as: Prelabeling, Ensemble, and Feature Extension.
- An optimization to the pipeline which allows users to intervene with the error detection done by RTClean.
- A proof of concept implementation of the proposed pipeline and approaches, followed by an evaluation of the concept on real-world IoT data in terms of the reduced labeling budget and the improved feature generation time.

Example Scenario

To aid the readers understanding, we introduce an illustrative example scenario depicted in Figure 1.1. Let us consider a scenario in which running a restaurant where operational expenses, especially those related to heating and cooling, play a significant role in overall profitability. In order to optimize these expenses, it becomes imperative to forecast the energy consumption associated with heating and cooling systems. To gather the necessary data for this task, a variety of temperature sensors are installed throughout the restaurant premises: some within the main dining area, some outside to gauge external temperatures, and others strategically placed inside the kitchen. The intent is to achieve a holistic understanding of temperature variations and their impact on energy consumption. All of these temperature measurements are meticulously collected and stored within a database housed in the restaurant's smart heating system. This system not only logs the real-time temperature data, but it also pulls in current weather forecasts and archives historical data related to both heating power and past temperature readings. With the amassed data, the goal is to develop a ML model capable of predicting the energy expenses for the upcoming days. This predictive model is designed to process all the accumulated temperature readings, weather forecasts, and historical heating power data. However, after careful deployment and monitoring, it becomes evident that the model's predictions deviate considerably from actual energy expenditures. Delving into the source of this mismatch, it's discovered that the primary cause is the unreliable transmission of data via wireless connections.

Recognizing the need for enhanced precision in the data, the restaurant's owner embarks on a mission to refine the quality of sensor readings. To this end, an error detection mechanism is put in place, aiming to pinpoint and rectify the inaccuracies. However, traditional error detectors, such as those based on outlier detection, prove inadequate. These basic methods grapple with discerning the subtle variances between indoor and outdoor temperature sensor readings. Given the contrasting temperature dynamics inside the restaurant as opposed to the restaurants kitchen, the owner starts considering more advanced solutions. The owner heard of the effectiveness of ML-based error detection methods and is keen to explore them. Yet, a new challenge presents itself: the absence of labeled data to train an error-detecting machine learning model. The owner contemplates using active learning, which is reputed to work effectively with minimal manually labeled data. However, he soon realizes that the task of labeling erroneous data is both intricate and time-consuming. With the day-to-day operations of the restaurant demanding his attention, he's left in a quandary.

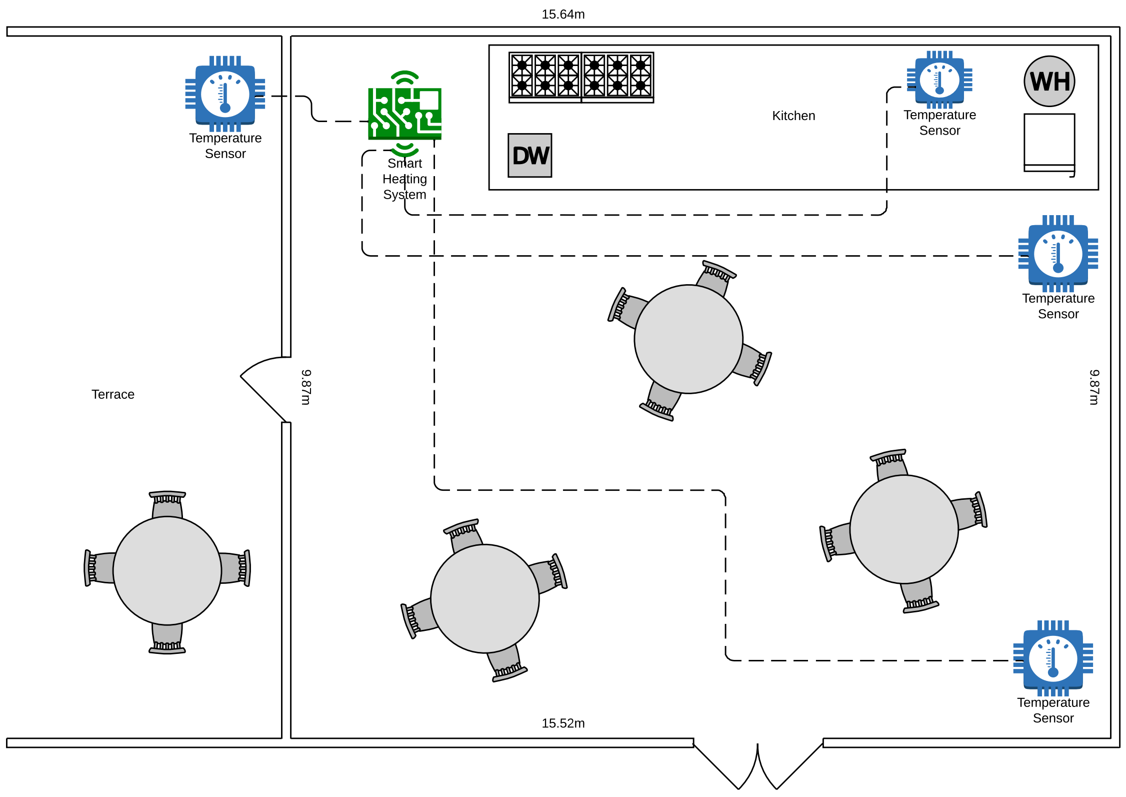


Figure 1.1: Smart Kitchen with Smart Heating

This is the point where our work becomes pivotal. We introduce a comprehensive pipeline tailored to craft an error detection model necessitating zero manual intervention. All that's required from the owner is a blueprint of the restaurant layout, in the form of a context model, and the erroneous data. Armed with this data, we provide a pipeline which uses RTClean toghether with active learning to create from a it sophisticated error detection model to be installed in the restaurant.

Structure

The structure of the thesis is as follows:

Chapter 2 - Fundamentals
Explanation of the foundational concepts related to the identified problem and the corresponding proposed solution.

Chapter 3 - Related Work
An overview of related scientific work. In particular, we focus here on related work from the field of learning and non-based error detection and how our work differs from them.

Chapter 4 - Context Aware ML-based Error Detection
In the core section of the thesis, we present an integrated approach that combines context-aware mechanisms with ML-based error detection techniques to identify errors. Here, we present our three approaches, the optimization step to incorporate user feedback, and how to use our approach for automated retraining of ML-models.

Chapter 5 - Implementation
Relevant details to the implementation of the proof-of-concept. This includes implementation decisions, challenges, and technical specifications.

Chapter 6 - Evalutation
Evaluation of the proof-of-concept in terms of its performance using real-world IoT data, focusing on reductions in the labeling budget and enhancements in scalability. For this, we present several results on the different approaches and discuss the implications of the results.

Chapter 7 - Summary and Future Work
A conclusion to this thesis and an outlook for future research.

2 Fundamentals

This chapter explains the knowledge fundamental to the stated problem and our proposed solution. First Section 2.1, gives an overview over data quality and data errors in general. Afterwards in Section 2.2, we highlight aspects of context aware computing and the importance of context models. We conclude the fundamentals with ML focusing on active learning in Section 2.3.

2.1 Data Quality

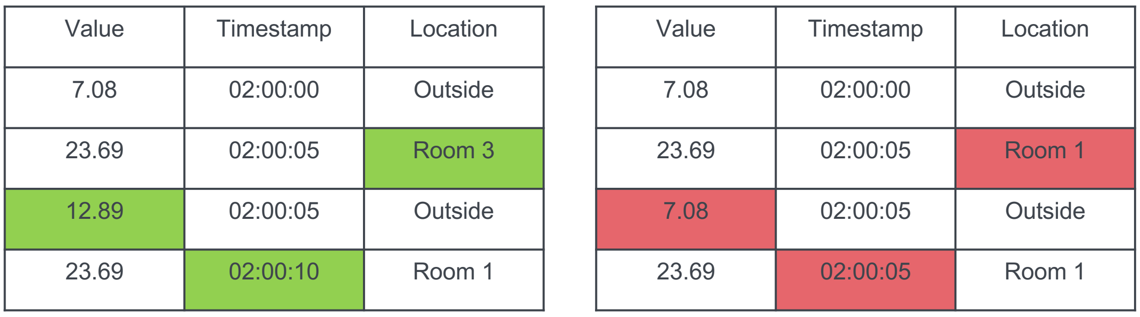
Data quality is fundamental in the realm of data engineering, data science, and business analytics. Poor data quality can lead to misguided insights, inefficient operations, and financial loss. The concept of data quality is broad and includes different definitions and interpretations. All of these have in common that the goal of high-quality data should be a truthful representation of the real-world scenario it aims to capture, free from defects and bias. Data quality can be measured through data quality analysis which can be divided into technical and non-technical analysis. Technical analysis means checking whether the data accurately represents the real world. Non-technical analysis includes aspects like accessibility, believability, relevancy, interpretability, and objectivity of the data [GFS+01; ORH05].

When evaluating data quality, many dimensions can be identified [SSA+12], some technical (T) and non-technical (NT) examples include:

- **(T) Accuracy:** How well does the data represent the real-world phenomena? Are there errors or misrepresentations?
- **(T) Completeness:** Is any essential data missing from the dataset?
- **(T) Consistency:** Are there contradictions within the data, especially when comparing across different data sources?
- **(NT) Timeliness:** How up-to-date is the data? Is it being updated at a frequency appropriate for its use?
- **(NT) Reliability:** Can the data be trusted as a source of truth? Was it sourced from a reputable place or process?
- **(NT) Relevance:** Is the data relevant to the task at hand?

In this work, we propose a pipeline that produces measurements which can be used to for technical data quality analysis with a focus on the accuracy dimension.

Therefore, we need to define what data errors according to the accuracy dimension. We follow the defintion of Cardoso et al. [CPIR14] where a data error is described as an erroneuos alteration from the ground truth value. In this context the ground trouth is defined as a the correct representation of



Value	Timestamp	Location
7.08	02:00:00	Outside
23.69	02:00:05	Room 3
12.89	02:00:05	Outside
23.69	02:00:10	Room 1

Value	Timestamp	Location
7.08	02:00:00	Outside
23.69	02:00:05	Room 1
7.08	02:00:05	Outside
23.69	02:00:05	Room 1

Figure 2.1: Example of Data Rrrors with Ground Truth (left) and Received Data (right)

a value. This allows us to use the ground truth as reference to determine which values of a data cell are erroneous. At the same time the ground truth can be seen as the true representation of values for all data cells in a data set. In Figure 2.1 we can see examples of our definition of a data error. The values in the right table marked in red differ from the ground truth table on the left. This means the received data is erroneous.

Data errors can have different causes. As our work focuses on data related to the IoT domain the cause of IoT related errors is of special importance. In the IoT contex, data errors can occur at the origin where the data is produced. This can be due to sensor malfunctions or unreliable data sources [BCC20]. Additionally, in the IoT domain devices are conncected to the internet and each other, creating large networks where the data is processed and forwarded many times. This can lead to inaccuries i.e. missing, noisy or redundant values [Sch22].

When working with sensors in the IoT context most of the data they produce is numerical and textual. Nambi et al. [CNC+18] define different types of numerical data errors.

- **Spikes:** A change of the values over a brief or extended duration that substantially exceeds expectations.
- **Short:** A data point that differs substantially from the expected value.
- **Stuck-at:** Values not or close to not changing over long period of time.
- **Noise:** Data that shows unexpected fluctuations over a specific time span.
- **Calibration:** Sensor data consistently deviates from the accurate value.

Noise, Shorts, and Spikes can be grouped under the term **outlier** and in this thesis we focus on detecting outliers for numerical errors (and textual).

2.2 Context-aware Computing

Applications are in ever-changing environments, therefore, they need to adapt to these changing environments. This is where the field of context-aware computing comes in. Here applications can adjust themselves based on the context they are in. Context includes location, nearby devices, individuals, and hosts present in the environment. For context aware computing it is important for the system to know where it is, who is in the proximity and what resources are available. Furthermore, such systems need the ability to adapt to changes if the surrounding environment changes.

When it comes to modelling approaches there are different types of techniques which can be used to model context information [PK16]. The following types exist:

- **Key Value Model:** The key-value pair model is a commonly used and straightforward data structure for representing contextual information. The context is modelled by assigning values to context information. This model presents the attributes in a key-value format [SAW94].
- **Markup Scheme Models:** All modeling methods using markup schemes utilize a hierarchical structure made up of markup tags with attributes and content. A prominent example would be to use XML.
- **Graphical Models:** A widely used modelling aproach is to use general-purpose graphical models. A prominent example is to use Unified Modeling Language (UML) diagrams. Given its versatile structure, UML is suitable for context modeling. Similarly, the visually appealing graphical extension of the Object-Role Modeling (ORM) approach also serves as a context model example.
- **Object-Oriented Models:** Object-oriented context modeling leverages the primary advantages of the object-oriented methodology, particularly encapsulation and reusability, to address challenges posed by the dynamic nature of context in context-aware settings.
- **Logic Based Models:** Here, context gets described mathematically which forms the bases for artificial intelligence approaches.
- **Ontology Based Models:** This model illustrates a set of concepts within a domain and the connections among them, visualizing a domain through a concept graph. Relationships can be hierarchical or based on meaning. Since many context modeling strategies are designed for specific application scenarios, their interoperability can be limited. Ontology-based methods capture knowledge about a domain, detailing concepts and their relationships. For instance, when presented with the basic classes of Person and Female, the Male class can be defined as Male $\equiv$ Person $\neg$ Female.
- **Spatial Context Model:** In many context-aware applications, space is the primary context. Context is characterized as "your location, your companions, and nearby resources". Some context modeling methods emphasize space and location. The majority of spatial context models are grounded in facts, arranging their context information according to physical location.

RTClean [DSA23] which we use in our proposed pipeline is based on ontology-based models and spatial context models. Therefore, we will highlight both in the following.

2.2.1 Ontologies

An ontology is like a dictionary and set of rules for a specific domain. It clearly describes the key concepts and how they relate to each other. Just as a dictionary helps us understand words, an ontology helps applications understand certain subjects. In philosophy, ontology discusses what things exist. In computer science, it defines what concepts or things can be represented in a program and how they connect. Ontologies are for organizing information and representing knowledge formally. So they can be used to formally collect and organize expert knowledge from the point of view of having a unified semantic, allowing the expert knowledge to be used.

Ontologies offer an in-depth understanding of current objects, their structure, and their interconnections, making them ideal for depicting context information. This is done by taking definitions and pairing the names of entities (like classes, relations, functions, etc.) with text that people can understand. This text explains the intended meaning of each name. Additionally, specific rules ensure these terms are correctly understood and used.

2.2.2 Ontology Functional Dependencies

In IoT, ontologies are particulary beneficial to represent the complex and diverse structure while still being fine-grained. Now, we need a means which enforces the rules defined inside an ontology towards the data instances which can be done with Ontology Functional Dependencies (OFDs).

OFDs serve as rules that enforce the semantics modelled in an ontology. These rules go beyond checking syntactic errors and encode the logic defined by the ontology.

They are an extension of Functional Dependencies (FDs) which specify a class of constraints on relations (tables). For example, a FD $X \to Y$ means that if two tuples (rows) agree on the attribute in X they must also agree on the attributes in Y.

Given this understanding of FDs, an OFD could be seen as a way of capturing similar constraints but within the more expressive and rich structure of ontologies. This allows to encoperate the knowledge of ontologies into FDs. Afterwards in order to validate a data point it gets validated against the OFDs.

RTClean [DSA23] uses context models to extract OFDs. Therefore we want to highlight context models more in detail in the follwoing.

2.2.3 Context Model

A context model integrates the IoT models, domain knowledge, and environment data. This context model allows to have an understandable and live image of a running application.

The context model we use as a basis to extract context information is based on the IoT context model introduced by [GABS22]. The context model is extended with components from the SSN ontology[1] and IoT-lite ontology [2]. Using these ontologies allows integrating static and dynamic context data. If the IoT environment is already managed in an IoT management platform like the Multi-purpose Binding and Provisioning Platform (MBP), an ontology can be extracted automatically by querying said platform.

[1]https://www.w3.org/TR/vocab-ssn/

[2]https://www.w3.org/submissions/iot-lite/

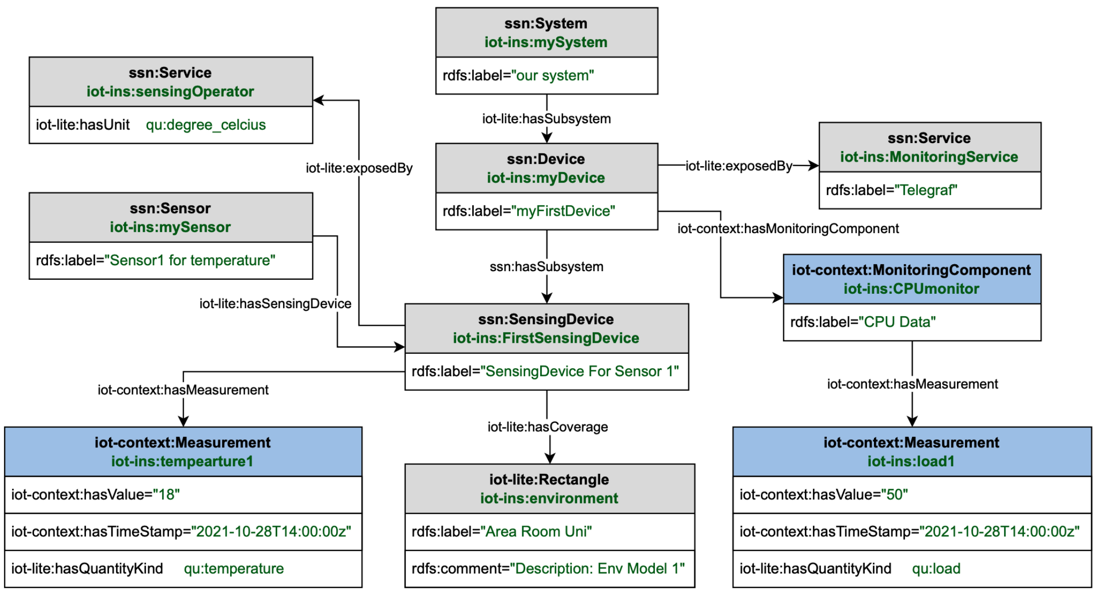

ssn:System
iot-ins:mySystem
rdfs:label="our system"
iot-lite:hasSubsystem
ssn:Device
iot-ins:myDevice
rdfs:label="myFirstDevice"
iot-lite:exposedBy
ssn:Service
iot-ins:MonitoringService
rdfs:label="Telegraf"
iot-context:hasMonitoringComponent
iot-context:MonitoringComponent
iot-ins:CPUmonitor
rdfs:label="CPU Data"
iot-context:hasMeasurement
iot-context:Measurement
iot-ins:load1
iot-context:hasValue="50"
iot-context:hasTimeStamp="2021-10-28T14:00:00z"
iot-lite:hasQuantityKind qu:load
ssn:Service
iot-ins:sensingOperator
iot-lite:hasUnit qu:degree_celcius
iot-lite:exposedBy
ssn:Sensor
iot-ins:mySensor
rdfs:label="Sensor1 for temperature"
iot-lite:hasSensingDevice
ssn:hasSubsystem
ssn:SensingDevice
iot-ins:FirstSensingDevice
rdfs:label="SensingDevice For Sensor 1"
iot-context:hasMeasurement
iot-context:Measurement
iot-ins:tempearture1
iot-context:hasValue="18"
iot-context:hasTimeStamp="2021-10-28T14:00:00z"
iot-lite:hasQuantityKind qu:temperature
iot-lite:hasCoverage
iot-lite:Rectangle
iot-ins:environment
rdfs:label="Area Room Uni"
rdfs:comment="Description: Env Model 1"

Figure 2.2: Example of the IoT Context Model [Sch22]

In Figure 2.2 we can see a snippet of an IoT context. The static information about the environment is grey, and the non-static information is marked in blue. In order to understand the notation we go through an example. We see that the *ssn*:*Device* named “myFirstDevice“ is connected with a monitoring component through the property *iot-context*:*hasMonitoringComponent*. The monitoring component in turn is linked to a concrete measurement through *iot-context*:*hasMeasurement*. This measurement has been taken at the timestamp shown under *iot-context*:*hasTimeStamp* with value 50 shown under *iot-context*:*hasTimeStamp* [GABS22].

2.2.4 RTClean

We need a framework which encoperates all the previous knowledge and can check wether data instances fullfill an ontology representing a particular IoT environment. We use RTClean [DSA23] and as it plays a major role in our proposed pipeline we will explain how it works here.

For our proposed pipeline in phase 1, we need an error detection method that uses context knowledge to derive whether tabular data can be classified as an error. Additionally, it needs to fulfill the requirement of having a short runtime compared to active learning-based approaches. RTClean provides an approach that can extract context knowledge from a context model in order to extract from it dependencies which then can be used to check if particular data point are erroneous or not. In the following, we explain how RTClean achieves this.

Asuming we are given a context model representing an IoT environment and we are given a dirty data set. The requirements for RTClean are an already existing context model and the dirty data. The context model is parsed, and from it, OFDs are derived. The incoming data is then validated against the extracted OFDs. RTClean then checks for every data cell if one of the OFDs is violated.

When a violation exists, the data cell is marked erroneous, otherwise it is marked as correct. The marking of these values is passed to the cleaning framework of RTClean, where the data entries are cleaned.

OFD Extraction

Before the actual evaluation of OFDs and data validation, the OFDs within a dataset must first be identified and extracted.

The process of extracting OFDs is achieved using SPARQL, which is a query language designed for querying data in the RDF format. RDF is a specialized format that represents information as directed, labeled graphs and is often used when working with ontologies. RDF describes data using a triple consisting of $(subject, predicate, object)$. The subject is an entitiy identifier, the predicate is the attribute name and the object is the attribute value either as literal data or an entity identifier. For example given the triple (emp3, title, "Vice President") would mean the employee with id emp3 has the title Vice President. The subject, predicate and object are represented using Uniform Resource Identifiers (URIs). In RDF, the URIs can be represented with the so-called turtle syntax which works as a placeholder for the URI. When converting tabular data to RDF the row identifier can be used as the subject, the column name as the predicate and the value as the object. By doing this one can get a triple for every fact in the table.

When querying this data, SPARQL seeks to match specific patterns with the triples.. When executing these queries, specific conditions are set using the WHERE clause, and specific parameters, designated with the prefix '?', can be chosen for extraction using the SELECT clause.

2.3 Active Learning

Active learning is a subfield within ML. To understand where it fits into the broader landscape of ML, let us first categorize the major areas of ML and then delve into active learning specifically [Set22].

1. **Supervised Learning:** This is the most common technique where the algorithm is trained on a labeled dataset, meaning the data includes both the input and the desired output. Examples include regression and classification problems.
2. **Unsupervised Learning:** Here, the algorithm is trained on data without explicit labels, trying to infer the underlying structure from the data. Examples include clustering and dimensionality reduction.
3. **Semi-supervised Learning:** This approach sits between supervised and unsupervised learning. The training process uses labeled and unlabeled data. The idea is to leverage the unlabeled data to enhance the learning process, often leading to improved model performance compared to using just the labeled data.
4. **Reinforcement Learning:** In this paradigm, an agent learns by interacting with an environment and receiving feedback in the form of rewards or penalties. It's about making a sequence of decisions to maximize some notion of cumulative reward.

5. **Transfer Learning:** This involves leveraging the knowledge gained while solving one problem and applying it to a different but related problem.

In the landscape of ML, active learning falls in the realm of semi-supervised learning. It takes advantage of having a limited labeled dataset and actively seeks to expand this set in the most informative way possible.

In active learning, the algorithm selects the instances for which it wants labels. Instead of receiving a static labeled dataset, the active learning algorithm interacts with an oracle to request the labels for specific examples. The goal is to select those instances that, once labeled, will most improve the models performance. Active learning is beneficial when labeling instances is expensive or time-consuming, so the model aims to learn with as few labeled examples as possible[Set22].

To visualize, imagine you are trying to teach a model to recognize a rare species of bird. Instead of randomly selecting thousands of photos and asking an expert to label them (bird or not bird), the model itself selects specific photos it is unsure about and asks the oracle only about those. By doing this, the model can rapidly improve its accuracy without requiring labels for every single photo.

Active learning is a paradigm yielding good results for error detection. The principle of active learning is illustrated in Figure 2.3. Let us break down the typical architecture and process of an active learning system based on [Set22]. The components of an active learning approach are:

- **Model:** This is a ML model that is being trained. It can be any kind of model: a deep neural network, a support vector machine, a decision tree, etc.
- **Labeled Dataset:** These are the data points for which labeled data already exists. At the beginning of the active learning process, this dataset might be small.
- **Unlabeled Dataset:** This is a pool of data points that the algorithm can query to get unlabeled data points. It is much larger than the initial labeled dataset.
- **Selection Strategy:** This is the core of active learning. It determines which data points from the unlabeled dataset should be queried for labels. The strategy is based on the principle of querying the instances where the model is most uncertain or most informative.
- **Oracle/Expert:** An oracle, often a human expert, provides the true label for the data points queried by the active learning algorithm.

In the beginning, active learning begins with a small dataset with known labels. Utilizing this data, an initial model gets trained. This can be any ML model. Afterwards, the model engages in a phase named Selection Strategy. In this phase, the model evaluates the unlabeled dataset, employing a specific set of criteria to calculate scores or evaluations to each data instance. The selected data gets forwarded to an oracle often represented through a human annotator. The oracle is presented with the selected data and is asked to label this data. The labeled data is placed then in the set of labeled data and the process repeats itself. It terminates if the labeling budget is reached. It is the number of times the oracle gets asked. This labling budget can be set as a hyperparameter by the user. The higher the labeling budget the more accurate the model should be and the less fast it is because the oracle has to lable more data[Set22].

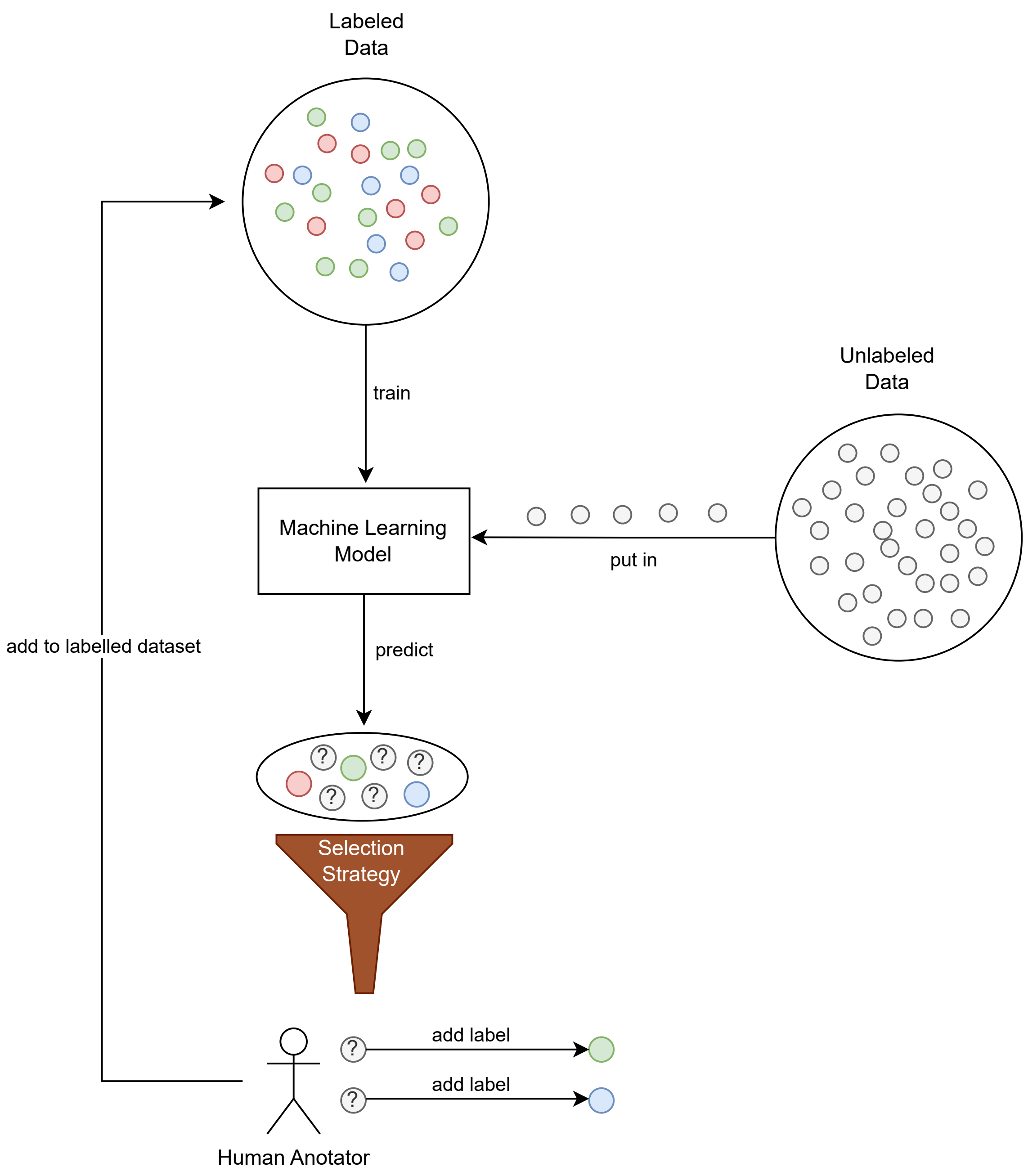


Figure 2.3: General Workflow of Active Learning Approaches [Set22]

3 Related Work

Automated detection and removal of errors are a pivotal task in data analytics. In Section 3.1, we present related work regarding non-learning based error detectors. Afterwards in Section 3.2, we present our related work regarding ML-based error detectors.

Given the increasing volume of data, reliance solely on human-centric error mitigation becomes economically and operationally unsustainable. The authors of [CIKW16] highlight the challenges of this paradigm as one of the persistent challenges in data analytics. In their study, the authors highlight major challenges that prevent the integration of automated strategies in key business contexts, with scalability identified as a primary concern. This thesis introduces a new error detection framework designed to reduce the need for human intervention, thereby improving scalability.

Our examination of related work regarding error detection identified two categories: non-learning-based error detectors and ML-based error detectors.

The non-learning-based error detection methods detect errors based on expert knowledge, rules, constraints, or statistical analysis [AHS23]. Each method tackles specific errors such as duplicates, outliers, or missing values. Machine learning-based error detectors approach error detection as a classification problem, automatically extracting features and training classifiers. Typically, these methods require labeled data. Both approaches have advantages and disadvantages depending on the use case of error detection. In the following, we will review some of the methods of both categories.

3.1 Non-learning based Error Detectors

Non-learning based error detectors have gained prominence for their ability to efficiently and accurately identify data errors without the need for extensive training data. This section reviews the existing literature and tools that fall under this category. These detectors encompass a broad spectrum, ranging from knowledge and rule-based methodologies to visual and statistical tools and even methods specifically tailored for concrete errors like duplicate detection.

At the forefront of knowledge-based approaches is **KATARA** [CMI+15]. This tool harnesses the power of knowledge bases to discern pattern violations, matching data tables with these bases to filter out the inaccurate entries. Of similar kind, the **NADEEF** system [DEE+13] provides a platform where users can formulate rules in the form of functional dependencies. Once these rules are set, NADEEF scans the data for any violations, ensuring compliance. Merging the best of both worlds, **HoloClean** [RCIR17] integrates pattern and rule violations. By melding denial constraints with correlations, it offers a more holistic choice for spotting errors.

For users who prefer visualization tools, **OpenRefineTool** [Ham13] presents an attractive option. This visual tool aims for simplifying error detection in dense datasets. It gives the user the ability to employ filtering and faceting on the data, permitting users to take a deep dive into data subsets and visually identify discrepancies. Supplementing this is **dBoost** [PMHM16], which employs tuple expansion to saturate table data with intricate details. By doing so, it paves the way for statistical outlier detectors to more effectively pinpoint deviations.

Less complex methods for outlier detection such as **Interquartile Range** [Zha13] and **Isolation Forest** [LTZ12] provide effective and easy to implement solutions. The Interquartile Range method identifies outliers by focusing on values that fall outside the first and third quantiles. In contrast, Isolation Forest employs an ensemble of decision trees to isolate and identify anomalous data points.

Duplicates are addressed by tools such as **Key Collision** [LRB+19]. By relying on user provided unique key attributes, this method identifies records that have similar key attributes as duplicates. Taking this a step further is **ZeroER** [WCS+20], which, although targeting duplicate detection, aspires to emulate the performance of supervised learning, all without labeled data. It utilizes similarity features and delving into underlying distributions to identify matching data tuples.

While all these aforementioned approaches exhibit promising results in error detection, they often fall short when it comes to leveraging information about the specific environment in which they operate. This is where **RTClean**, introduced in Section 2.2.4, distinguishes itself. It adeptly harnesses environmental knowledge, enhancing the precision and relevance of error detection in context-specific scenarios. While RTClean offers a robust solution, its standalone nature has traditionally hindered its integration with other methodologies. Previously, RTClean was not conceptualized to function within a broader pipeline. In our research, we enhanced RTClean, enabling it to seamlessly integrate within diverse pipelines and cooperate with various frameworks. This adaptability now permits us to harness its capabilities more effectively, particularly in data engineering scenarios.

3.2 Machine Learning based Error Detectors

Shifting our focus from traditional methods, ML-based error detectors have risen to prominence, utilizing advanced algorithms to detect data anomalies with increased performance. These detectors span a wide spectrum, from supervised learning approaches that require labeled data for training, to unsupervised techniques that identify patterns without explicit guidance, and even semi-supervised systems that combine the best of both worlds. In our examination, we prioritize ML-based detectors that employ the semi-supervised approach. Additionally, we focus on detectors that aim to minimize human intervention and lessen the users workload.

The first approach we examine is **RAHA** [MAC+19]. It streamlines the error detection process by minimizing manual configuration and tuning, which is a time intensive task in common ML and error detection systems. It does so through a two step process. First it employs a diverse set of error detection strategies to create a set of potential error candidates. Once various strategies flag potential errors, RAHA uses a ML model to rank these potential errors. This is done over multiple iterations. After each iteration, users can verify or correct a small number of predictions through user input, and RAHA refines its model based on this feedback. This helps to differentiate genuine errors from false positives.

Next, we explore **ED2** [NMA19]. Similar to RAHA, ED2 uses a two-stage active learning approach, combining random sampling in Stage 1 with uncertainty sampling in Stage 2. This ensures that the model gets a broad understanding of the data in the first stage and then improve its accuracy by focusing on the more challenging instances in the second stage. In a first step ED2 samples a small subset of the dataset and requests the user to label this subset for errors. It then trains an initial classifier on this labeled subset. The main aim of this stage is to build an initial understanding of the data and the kind of errors present. In the next stage the model is refined. Based on the initial model from Stage 1, ED2 identifies tuples in the data that are most ambiguous or where the model is most uncertain regarding their correctness. The system then queries the user specifically about these ambiguous instances to get their labels. This targeted querying is more efficient as it focuses on the hard to classify instances, ensuring that the users effort is most effectively utilized.

In our examination of error detection methodologies, a notable contribution is that of **Picket** [LZR21]. Unlike previous, approaches Picket does not require user input to detect errors it leverages a self supervised learning paradigm. Self-supervised learning is commonly used when labeled data is limited and manual intervention is unfavorable. The foundational idea behind this approach involves artificially obscuring certain portions of the input data, forcing the model to infer or reconstruct the concealed segments by relying on the visible portions. This technique helps the model in understanding the inherent relationships and dependencies within the data without user input. Picket takes advantage of this to detect errors. It analyses which data entries are challenging to unmask and marks them as data errors.

In conclusion, ML based error detectors, from semi-supervised systems like RAHA and ED2 to self-supervised paradigms like Picket, represent a significant advancement in data analytics. However, non-learning based error detectors retain their value, often delivering faster and more tailored error detection. In our work, we enhance ED2 with capabilities to use the information gained from non-learning based error detectors. In the upcoming chapter, we will introduce a pipeline that synergizes both non-learning and ML-based detectors, aiming for efficient error detection with minimal human intervention.

4 Context-aware Machine Learning based Error Detector

The main objective of this thesis is to conceptualize and develop a pipeline which enhances a ML based error detection approach with context information. This integration aims not only to support the strengths of both paradigms but also to reduce their individual limitations. The pipeline should overcome the issue of the lower accuracy of the non-learning based error detection. At the same time, overcome the issue of human labeled data in ML-based error detection. Therefore, we want to use already existing context-aware error detection systems gather from them knowledge and enhance with this knowledge with another already existing ML-based error detector. Our proposed pipeline consists of two stages: a context knowledge stage and ML stage.Through this hybrid approach, the aspiration is to foster a more robust and scalable error detection mechanism. This chapter presents the following approaches and improvements:

- Approach 1: Prelabeling uses RTClean as prelabeling for ML-based error detection without human input.
- Approach 2: Ensemble uses RTClean together with other error detection methods prelabeling for ML-based error detection without human input.
- Approach 3: Feature Extension uses RTClean as prelabeling to extend the features for ML-based error detection with human input.
- An improvement of the pipeline across all approaches through user-defined thresholds.
- A modification to the pipeline for automated retraining of ML algorithms.

We begin by providing an overview of the pipeline we propose in Section 4.1. Subsequently in Section 4.2-4.4, we detail our designed approaches. We first present approaches that operate autonomously, called the prelabeling and ensemble techniques, in Section 4.2 and 4.3. Next in Section 4.4, we explore a strategy that reintroduces human intervention to speed up feature generation called the feature extension method. We then describe an improvment that can be applied to the initial phase of all our strategies, in Section 4.5. Finally in Section 4.6, we outline a method that employs our pipeline to determine the best moments to retrain ML models during their operation.

4.1 Overview of Two-Stage Pipeline

We start with a general overview of our two-stage pipeline and introduce key concepts. Our two-stage pipeline consists of the following components:

1. **Data Preparation:** This initial phase involves aggregating both context information and the 'dirty' data from the environment. It sets the stage for further processing by preparing the data for the first major stage.
2. **Stage 1: Context Knowledge:** The gathered context knowledge is compared against dirty data. Each data cell undergoes an evaluation using the context-aware error detection mechanism to ascertain its validity. Depending on the specific context-aware approach employed, the results of error detection may vary. Subsequently, these results are curated and forwarded to Stage 2 for deeper analysis.
3. **Stage 2: Machine Learning:** The outcomes of Stage 1 are channeled into a ML-based error detection approach where they get the data received its final evaluation. The premise is that the insights from Stage 1 can reduce the computational load of the ML-based approaches by either improving the features they use or by reducing the number of training instances they need. Following this, the resultant findings are cataloged as the definitive error detection outputs, pinpointing the erroneous cells. While these results can be harnessed for further and more advanced data cleaning operations, this step falls beyond the scope of this thesis.

Let us summarize the key definitions from above.

Definition 4.1.1
Erroneous Data: *The data has not been checked on the validity of the values and could contain values which differ from a ground truth value.*

Definition 4.1.2
Dirty Dataset: *A tabular dataset containing numerical and textual data generated automatically.*

Definition 4.1.3
Stage 1 Results: *A version of the dirty dataset where each individual datacell is marked as erroneous or otherwise using a boolean where true signifies the presence of an error.*

Definition 4.1.4
Stage 2 Results: *An updated version of the Stage1 Results where ML techniques have been employed for enhanced error detection. These results are the final output of the proposed pipeline.*

Definition 4.1.5
Cleaning: *The process of rectifying or amending erroneous data entries to align them with their true or intended values.*

We have explored three approaches of this general pipeline which we will present in the following. Each variant will be dissected, explaining its components and their details.

4.2 Approach 1: Prelabeling

The aim of our first approach is to improve the scalability of active learning based error detectors by removing the human component from the detector. As mentioned already in Section 2.3, an oracle, typically manifesting as a human expert, plays a pivotal role in interfacing with the active learning system. This involvement is time consuming and often requires a domain expert.

Additionally, in dynamic environments where retraining of an active learning approach is needed, the continuous availability of such a domain expert becomes impractical. Consequently, we propose in Approach 1: Prelabeling a concept that seeks to remove human involvement from the cleaning process.

We recall that the human intervention has been instrumental in guiding the active learning model, particularly for instances where the model exhibits uncertainty. The human or oracle needs to tell the active learning model in this case what the correct value would be. The Approach 1 innovation replaces this oracle output with the results generated by RTClean. RTClean provides a means which offers a cleaned version of the dirty dataset as described in Section 2.2.4. The 'cleaned' dataset can be used as a mock ground truth which in return can be used by the active learning approach as the oracle. We refer to this output as prelabeling.

The prelabling steps unfold as follows. As shown in Figure 4.1, RTClean uses the context model derived from the environment to extract OFDs (Order Functional Dependencies). These OFDs subsequently operate as rule sets to detect whether the incoming data from the environment violates these rules indicating dirty data. The identified instances are then cleaned generating a version of the dirty dataset with cleaned values by RTClean. An example of the tranformation which happens through RTClean is shown in Figure 4.2. First RTClean detects the errors marked in green. Afterwards RTClean cleans the identified erroneous values. The cleaned version of the input data is then used a prelabling for the second stage where the ML based error detector is used.

As we transition to Stage 2, this prelabeled dataset effectively plays the role of an oracle. The active learning model is then trained on the original dirty dataset. In instances where oracle intervention is typically sought, our prelabeled dataset provides the requisite guidance. Figure 4.3 shows the integrated version in the general architecture of Active Learning proposed in Section 2.3. In Stage 2 the dirty data highlighted in red is used as the unlabeled dataset. Then the active learning algorithm selects instances of the dirty dataset which the active learning algorithm is unsure about. The selected data instances are then passed to the prelabeled data from Stage 1 which is used as oracle. Afterwards, the labeled instances get passed to the set of labeled instances and the active learning loop continues until the labelling budget is reached. At the end of the pipeline we receive a model which can be used for detecting future errors.

An inevitable question looms: Why not exclusively rely on RTClean's output? Why necessitate an ensuing ML-based error detection phase? Is not the sequential deployment of two error detection paradigms redundant? Our rationale is twofold. Primarily, our intention is to use RTClean as a preprocessing step, thereby simplifying the task for the subsequent ML detector. In essence, while RTClean confidently rectifies certain errors, it delegates ambiguous cases to the ML mechanism. Additionally, this design aids in our exploration of automating the human component traditionally present in active learning paradigms. In summary the overall overview of the pipeline with Approach 1 is shown in Figure 4.4. Over the course of the next approaches we extend this basic architecture.

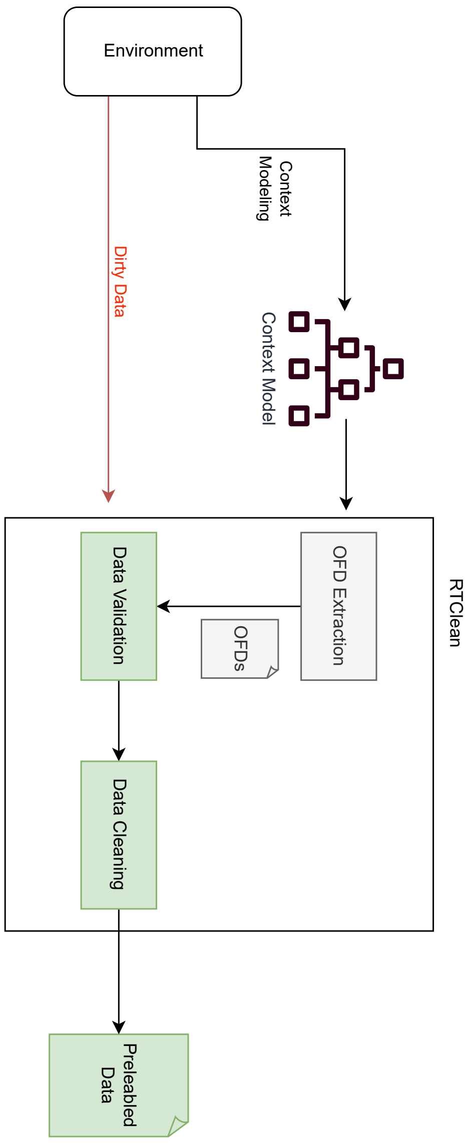
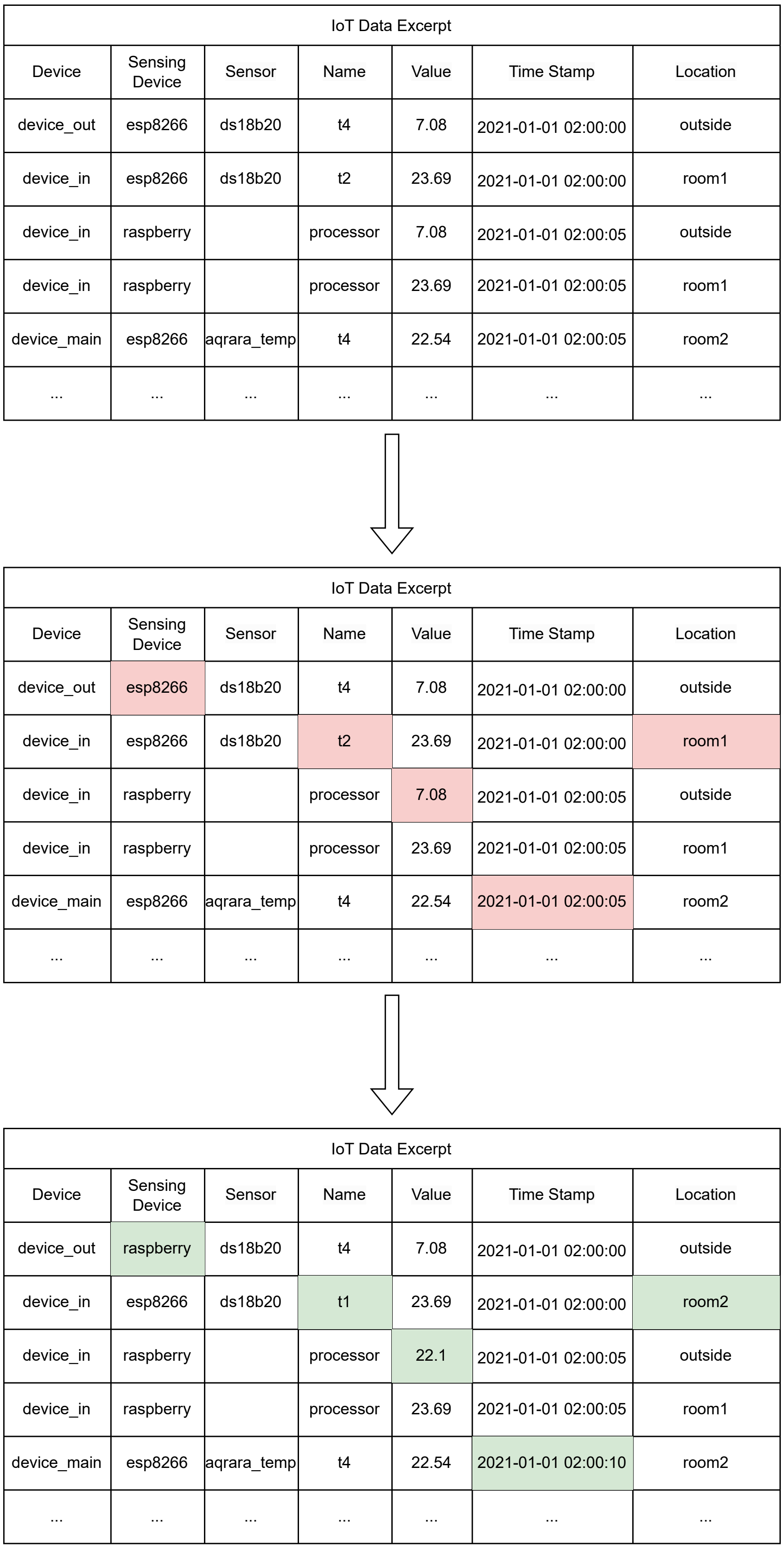


Figure 4.1: Creation of the Prelabeling

4.3 Approach 2: Ensemble

Building upon the foundation established in Approach 1, our second approach integrates an ensemble strategy. We do not only run RTClean but an ensemble of methods covering a wide array of errors. The core objective of this approach is to not only cover the errors of RTClean but further utilize the power of several error detection methods together, thereby ensuring a more comprehensive error identification and rectification mechanism. Upon the independent assessment by each error detection technique, a concurrence mechanism is adopted to collectively ascertain erroneous data instances. The idea is that if more approaches run in parallel and toghether decide whether an error exists, we can amass a more extensive knowledge base. This enriched information reservoir then powers the oracle in Stage 2, further fortifying its error detection capabilities. In this approach, while Stage 1 is appended with an ensemble mechanism, Stage 2 remains consistent with Approach 1.



IoT Data Excerpt

Device	Sensing Device	Sensor	Name	Value	Time Stamp	Location
device_out	esp8266	ds18b20	t4	7.08	2021-01-01 02:00:00	outside
device_in	esp8266	ds18b20	t2	23.69	2021-01-01 02:00:00	room1
device_in	raspberry		processor	7.08	2021-01-01 02:00:05	outside
device_in	raspberry		processor	23.69	2021-01-01 02:00:05	room1
device_main	esp8266	aqrara_temp	t4	22.54	2021-01-01 02:00:05	room2
...	...	...	...	...	...	...

IoT Data Excerpt

Device	Sensing Device	Sensor	Name	Value	Time Stamp	Location
device_out	esp8266	ds18b20	t4	7.08	2021-01-01 02:00:00	outside
device_in	esp8266	ds18b20	t2	23.69	2021-01-01 02:00:00	room1
device_in	raspberry		processor	7.08	2021-01-01 02:00:05	outside
device_in	raspberry		processor	23.69	2021-01-01 02:00:05	room1
device_main	esp8266	aqrara_temp	t4	22.54	2021-01-01 02:00:05	room2
...	...	...	...	...	...	...

IoT Data Excerpt

Device	Sensing Device	Sensor	Name	Value	Time Stamp	Location
device_out	raspberry	ds18b20	t4	7.08	2021-01-01 02:00:00	outside
device_in	esp8266	ds18b20	t1	23.69	2021-01-01 02:00:00	room2
device_in	raspberry		processor	22.1	2021-01-01 02:00:05	outside
device_in	raspberry		processor	23.69	2021-01-01 02:00:05	room1
device_main	esp8266	aqrara_temp	t4	22.54	2021-01-01 02:00:10	room2
...	...	...	...	...	...	...

Figure 4.2: Example Transformation of Data

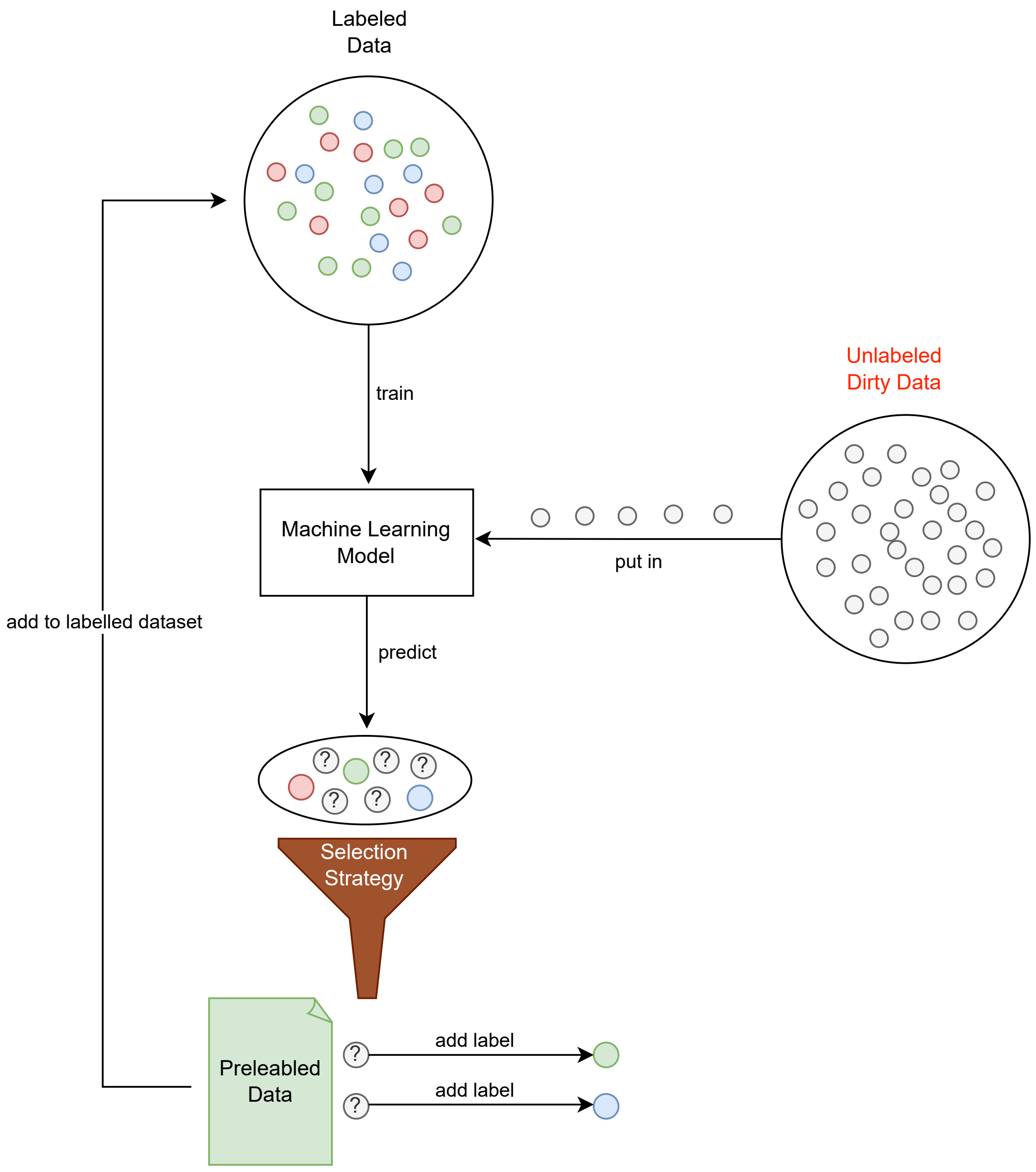


Figure 4.3: Active Learning while using RTClean Prelabeling

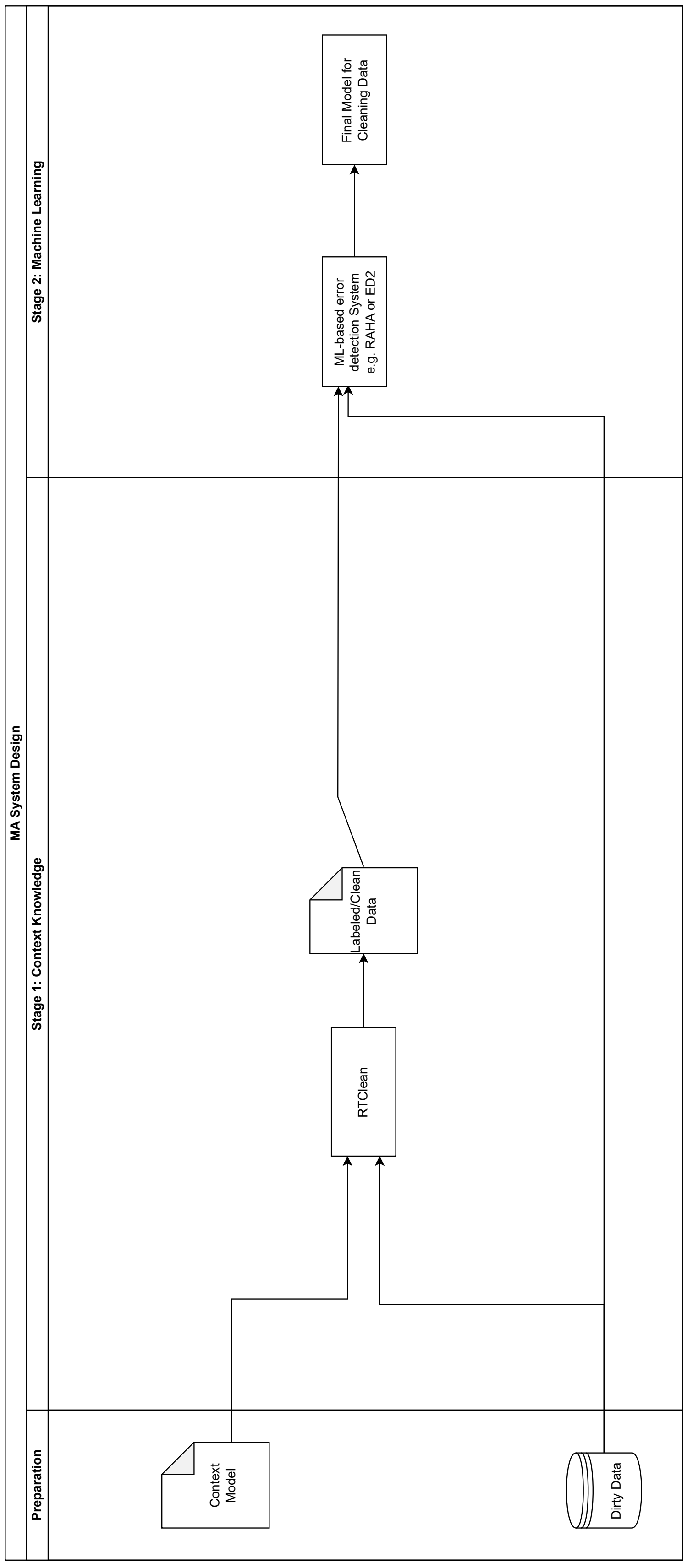

MA System Design
Preparation
Stage 1: Context Knowledge
Stage 2: Machine Learning
Context Model
Dirty Data
RTClean
Labeled/Clean Data
ML-based error detection System e.g. RAHA or ED2
Final Model for Cleaning Data

Figure 4.4: Overall Architecture of Approach 1: Prelabeling

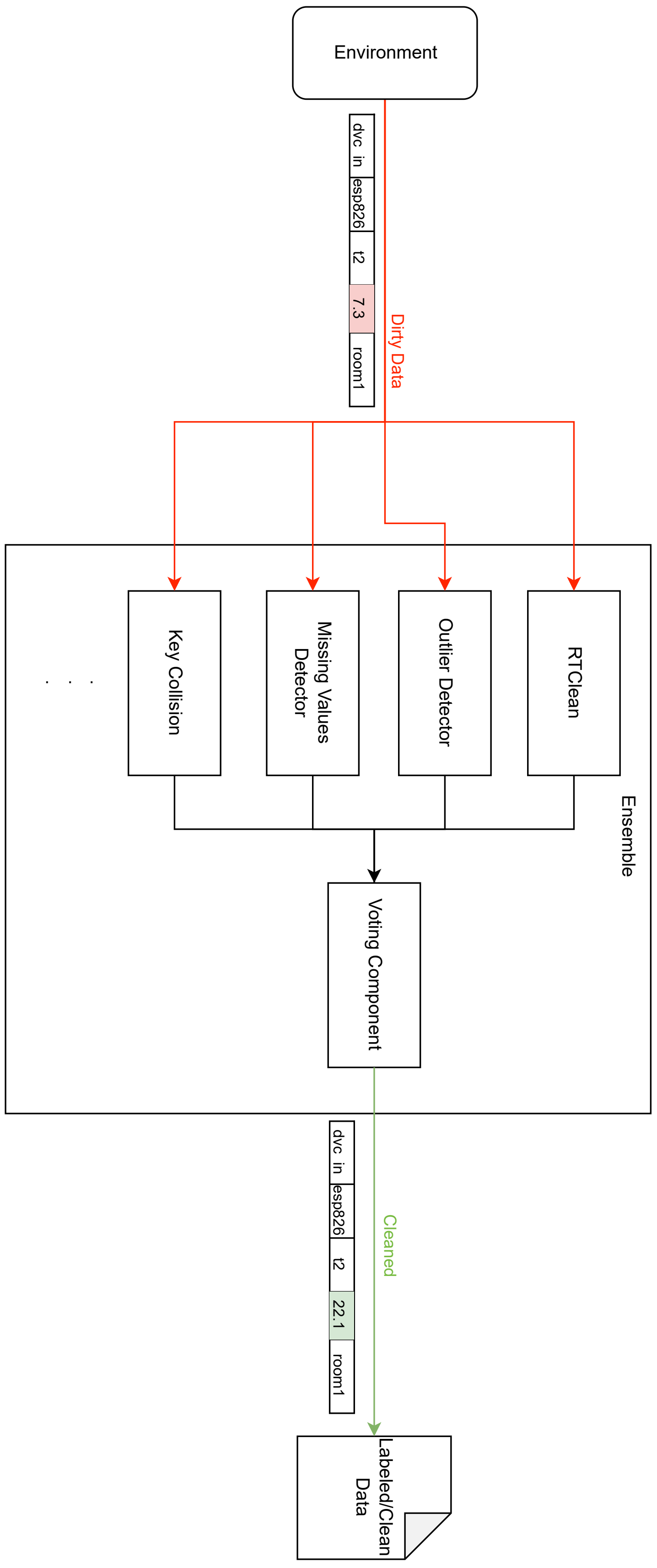


Figure 4.5: Workflow Stage 1 with Approach 2: Ensemble

Therefore, let us take a closer look into the *Ensemble* component as shown in Figure 4.5. Instead of feeding the dirty data only into RTClean we feed it into several error detection methods in parallel. Since one of our goals is to keep the Stage 1 of the two stage architecture comparably light we do not want to increase the computation cost to much. A pivotal consideration is computational efficiency; to preserve the lightweight nature of Stage 1, the selected detection techniques should not impose significant computational overhead.

Naturally, not all error detection methods agree on the same label when deciding on erroneous data instances. This why we added a sampling component afterwards which aggregates the results for each error detection algorithm. The sampling strategy we used is called min-k. It is a majority voting strategy which votes for erroneous if at least k algorithms agreed that a value is erroneous. We are aware that there are other selection strategies like weigthed voting, probability aggreagtation, meta-learninng and bayesian model averaging. Therefore an in-depth analysis of different sampling strategies was not crucial for us. The emphasis was not on optimizing the ensemble's performance but on assessing the pipeline's enhanced scalability. Therefore, a strategy that strikes a balance between simplicity and efficacy was deemed most appropriate, leading us to min-k.

In Figure 4.6, we extended our overall pipeline architecture with the idea of Approach 2. Therefore, we added split gateway which either uses Approach 1 or Approach 2 depending on user input. Depending on the user input one of the approaches is used. Regardless of the selected approach, the pipeline's subsequent stage works identically.

4.4 Approach 3: Feature Extension

Unlike Approaches 1 and 2, Approach 3 reintroduces a human oracle into the pipeline. This is particularly useful when users are hesitant to rely solely on RTCleans input. However, we still aim to leverage the potential of RTClean's insights. In ML, a pivotal goal is feature optimization, selecting features that optimally describe data for a ML model. In Approach 3: Feature Extension, we augment the feature set used in active learning with results derived from RTClean.

To achieve this, we first obtain cleaned data from RTClean and create a matrix that indicates the presence of errors for each data point. We then expand the feature matrix with the matrix derived from RTClean.

A feature vector **x** for a single data sample is an n-dimensional vector where each dimension represents a feature. In mathematical terms:

$$(4.1)\quad \mathbf{x} = [x_1, x_2, \ldots, x_n]$$

where n is the number of features, and x_i represents the value of the i^{th} feature.

If you have m samples in your dataset, you can represent the entire dataset as a matrix **X** of size $m \times n$, where each row is a feature vector:

$$(4.2)\quad \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{22} & \ldots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix}$$

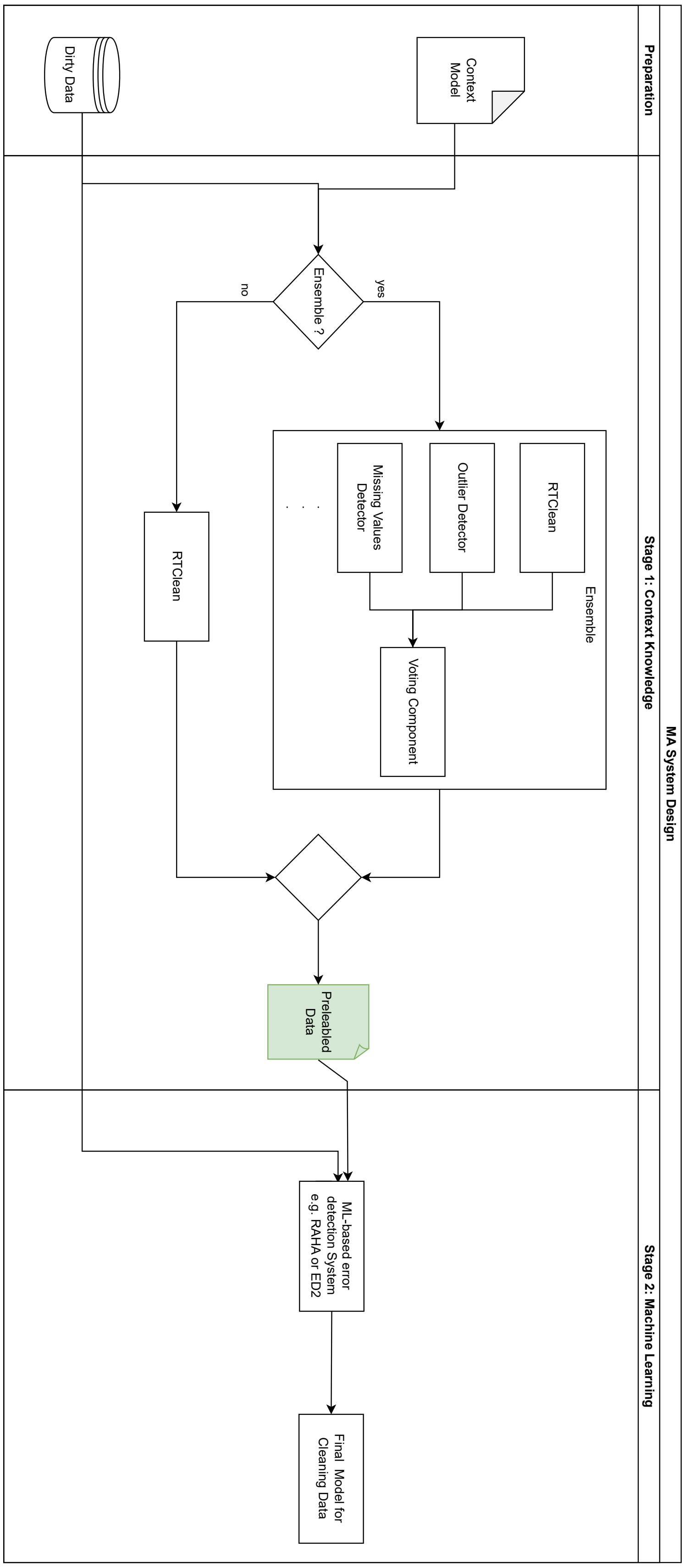


Figure 4.6: Overview of Pipeline including Approach 2: Ensemble

If each of the m samples in your dataset has k many results from RTClean the resulting matrix would be:

$$\textbf{RTCLEAN} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mk} \end{bmatrix} \tag{4.3}$$

In Approach 3: Feature Extension, we concatenate X and RTCLEAN matrices together mathematically described:

$$\begin{aligned} &\textbf{EXTENDED} = \textbf{X}||\textbf{RTCLEAN} \\ &= \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} || \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mk} \end{bmatrix} \\ &= \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} & c_{11} & c_{12} & \cdots & c_{1k} \\ x_{21} & x_{22} & \cdots & x_{2n} & c_{21} & c_{22} & \cdots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} & c_{m1} & c_{m2} & \cdots & c_{mk} \end{bmatrix} \end{aligned} \tag{4.4}$$

The enhanced feature matrix is then employed by the active learning algorithm. Our ambition is to enhance the active learning algorithm by combine RTCleans knowledge while retaining the human oracles input.

In Figure 4.7, the integration of Approach 3 into our proposed pipeline is depicted. We have incorporated a user configuration gateway, represented by a diamond shape, to determine the approachs utilization. If the 'no' option is selected, the Prelabeling serves as the oracle for the active learning model, as in previous setups. However, choosing 'yes' initiates the described feature extension process, wherein the dataset features are augmented with prelabeling results. A human oracle is reintroduced, as the enhanced features now use the prelabeling insights.

Our aspiration with this approach is to improve scalability. By enriching the features, we anticipate that models can accelerate their learning pace, either reducing the learning time or lowering the number of features to achieve good results. This is relevant since several active learning algorithms automatically generate their features, a process that can be time-consuming for large datasets [AHS23; NMA19].

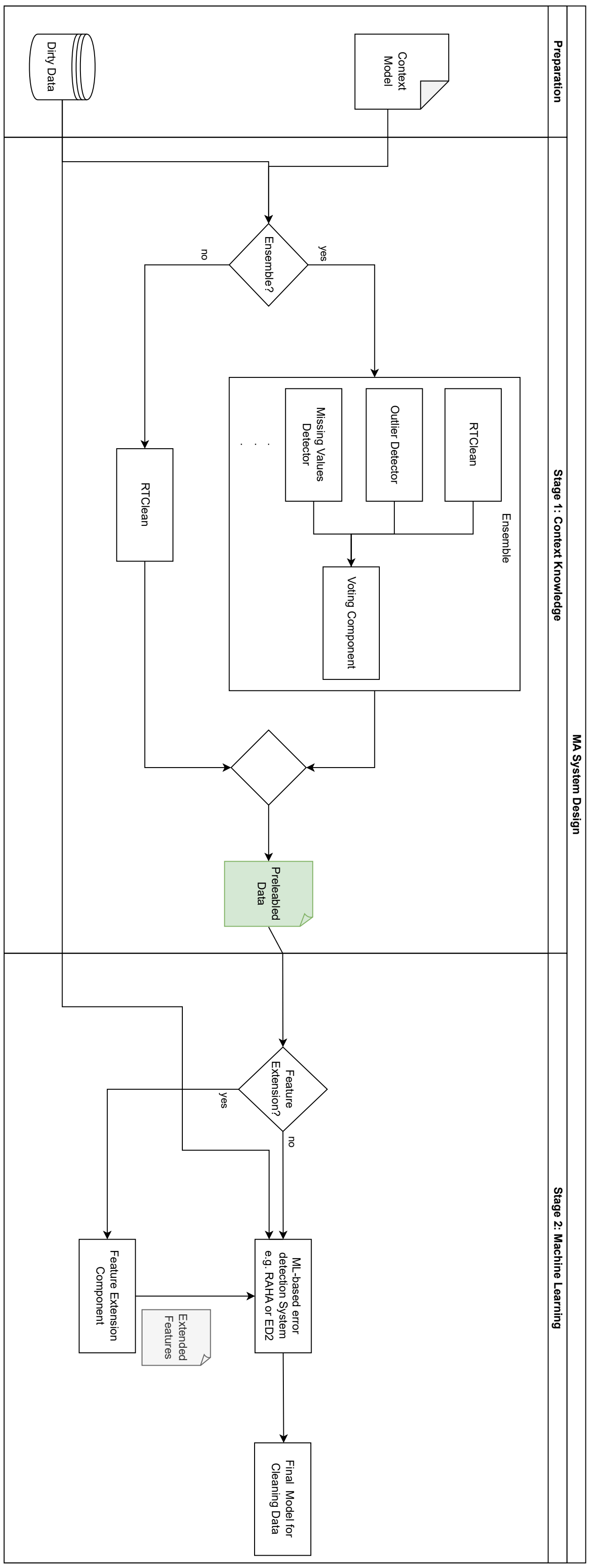


Figure 4.7: Overview Pipeline with Approach 3: Feature Extension

4.5 Improvement on RTClean using a User Threshold

We can enhance our pipeline by empowering users to make decisions regarding RTClean's outcomes. The objective is to offer users the flexibility to review and, if desired, reverse modifications made by RTClean. This is especially important in instances where users may question the accuracy or appropriateness of RTCleans cleaning during Stage 1 prelabeling. Hence, we introduce a User Threshold. We define the following terms:

Definition 4.5.1
Number of Cleaned Values (ncv): *This represents the count of records that RTClean has modified between the original and the cleaned versions of a dataset.*

Definition 4.5.2
Number of Values (nv): *This is the total count of records in a dataset.*

Definition 4.5.3
Fraction of Cleaned Values (fcv): *Represented as the proportion of records cleaned by RTClean. Mathematically, fcv is the ratio of ncv to nv, expressed by:*

$$fcv = \frac{ncv}{nv}$$

Users set a threshold, indicating their allowable limit of changes per column. Following the execution of RTClean or the ensemble method, the *fcv*-value is computed for each column in the pre-labeled dataset. If the computed fcv surpasses the user-defined threshold, all modifications done by RTClean to that column are reverted.

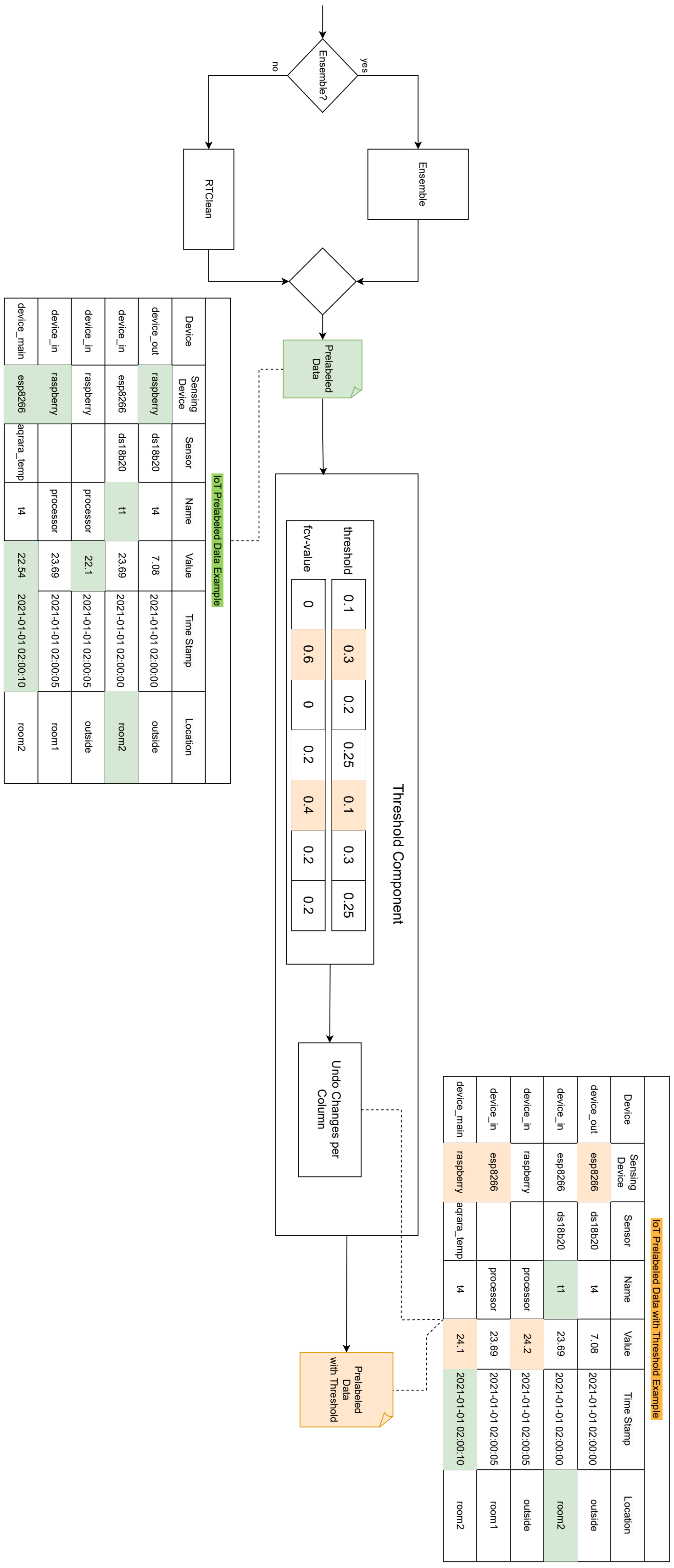
Figure 4.8 illustrates the improved workflow. Here, for each column in the prelabeled dataset, the *fcv*-value is computed and compared against the user-specified threshold. Columns marked in orange signify a breach of the threshold, implying that changes for these columns will be reversed. The final modified data is stored as the 'prelabeled with threshold file', which is subsequently utilized in Stage 2 for ML, as depicted in Figure 4.9.

This wraps up our proposal for a context-aware ML model pipeline. We've introduced a two-stage framework: Stage 1 employs RTClean to comprehend contextual knowledge, which is then used in Stage 2 to augment ML. Furthermore, we have outlined four approaches designed to improve the pipelines scalability and effectiveness.

In the following section, we will discuss how RTClean can be leveraged to automatically initiate retraining of ML models, particularly beneficial when environmental changes risk to make these models obsolete.

4.6 Automated Retraining of Machine Learning in Context-changing Environments

Machine learning models, while powerful at identifying erroneous values, often face challenges in dynamic environments where the underlying context can vary. The challenge arises from determining when to retrain these models, balancing accuracy and resource utilization.



IoT Prelabeled Data Example

Device	Sensing Device	Sensor	Name	Value	Time Stamp	Location
device_out	raspberry	ds18b20	t4	7.08	2021-01-01 02:00:00	outside
device_in	esp8266	ds18b20	t1	23.69	2021-01-01 02:00:00	room2
device_in	raspberry		processor	22.1	2021-01-01 02:00:05	outside
device_in	raspberry		processor	23.69	2021-01-01 02:00:05	room1
device_main	esp8266	aqara_temp	t4	22.54	2021-01-01 02:00:10	room2

threshold	0.1	0.3	0.2	0.25	0.1	0.3	0.25
fcv-value	0	0.6	0	0.2	0.4	0.2	0.2

IoT Prelabeled Data with Threshold Example

Device	Sensing Device	Sensor	Name	Value	Time Stamp	Location
device_out	esp8266	ds18b20	t4	7.08	2021-01-01 02:00:00	outside
device_in	esp8266	ds18b20	t1	23.69	2021-01-01 02:00:00	room2
device_in	raspberry		processor	24.2	2021-01-01 02:00:05	outside
device_in	esp8266		processor	23.69	2021-01-01 02:00:05	room1
device_main	raspberry	aqara_temp	t4	24.1	2021-01-01 02:00:10	room2

Figure 4.8: Example of User Threshold

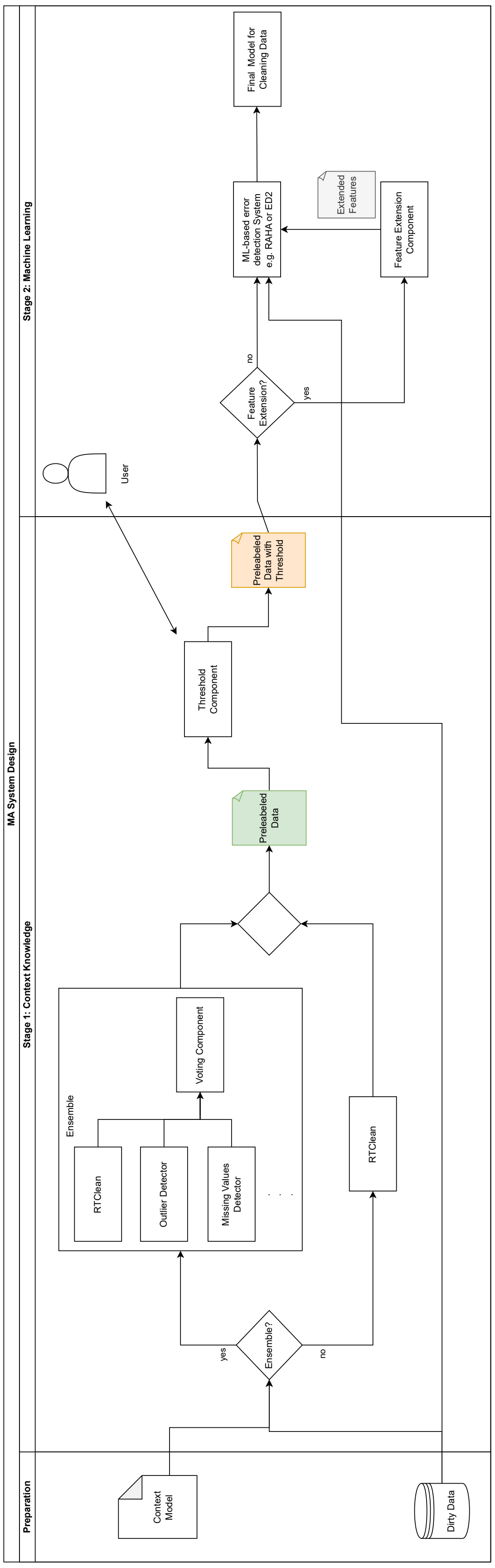


Figure 4.9: Overall Pipeline including User Threshold

Retraining ML models is a resource-intensive process. Undertaking this task too early can waste computational resources. In contrast, delaying it might risk using an outdated model for an extended period of time, cleaning large sections of data obsolete due to their misalignment with real-world representations. Striking an optimal balance is paramount.

This section introduces an approach to automate the retraining of ML models in response to context changes, leveraging RTClean. Instead of enhancing the ML models directly with RTClean, our pipeline pivots towards ensuring the models remain attuned to changing contexts.

Contextual changes can manifest in various ways, from sensor location shifts to temperature fluctuations that devices might encounter. Our two stage architecture, initially cleans errors based on context information. A crucial observation is that any significant deviation in the regular cleaning pattern of RTClean can signal a contextual change. We can use the *fcv*-value to measure this deviation.

Figures 4.10 and 4.11 show the idea of our proposal. We utilize a sliding window allowing for systematic and sequential inspection of incoming data streams.

Definition 4.6.1
Window: *A fixed-size subset of a larger dataset, designed to capture a specific portion of the data for processing by RTClean.*

Definition 4.6.2
Window size: *Denotes the count of data points within a window.*

Definition 4.6.3
Step size: *Represents the interval (in terms of time) after which the window progresses (slides).*

For illustrative purposes, consider a lengthy stream of data as shown in Figure 4.10. Instead of processing this stream in its entirety, we employ a sliding window of size n to analyze the last n data points at a time. After processing the first subset, the window slides by a pre-defined step, and the subsequent subset is then processed. This process repeats itself continously.

The sliding windows role is to detect variations in RTCleans cleaning patterns. Should a contextual shift occur, the window should identify it by observing a spike in the cleaning tasks undertaken by RTClean. Such an uptick in cleaning operations is measured through the *fcv*-value. A spike in this metric signals potential context changes, warranting the retraining of the ML model.

Our refined approach, depicted in Figure 4.11, incorporates this strategy into the earlier introduced two-stage pipeline. In Stage 1, the Window Splitter module manages the windows segmentation and advancement. Each window is subsequently processed by RTClean, and with the static context model. This results in cleaned data for each window. Additionally, for each window its *fcv*-value gets computed and stored. The pipeline then evaluates if any significant spikes in the *fcv*-values are present. Absence of such spikes implies no retraining is necessary, and the system persists in its monitoring mode. However, the detection of such spikes triggers the retraining of the ML-based error detection system.

Throughout this chapter, we have presented different context-aware ML models, based on four approaches which utilize RTClean. Beyond this, we showcased the utility of RTClean in automated retraining mechanisms for these models. In the next chapter, we highlight the implementation of these approaches into a proof of concept

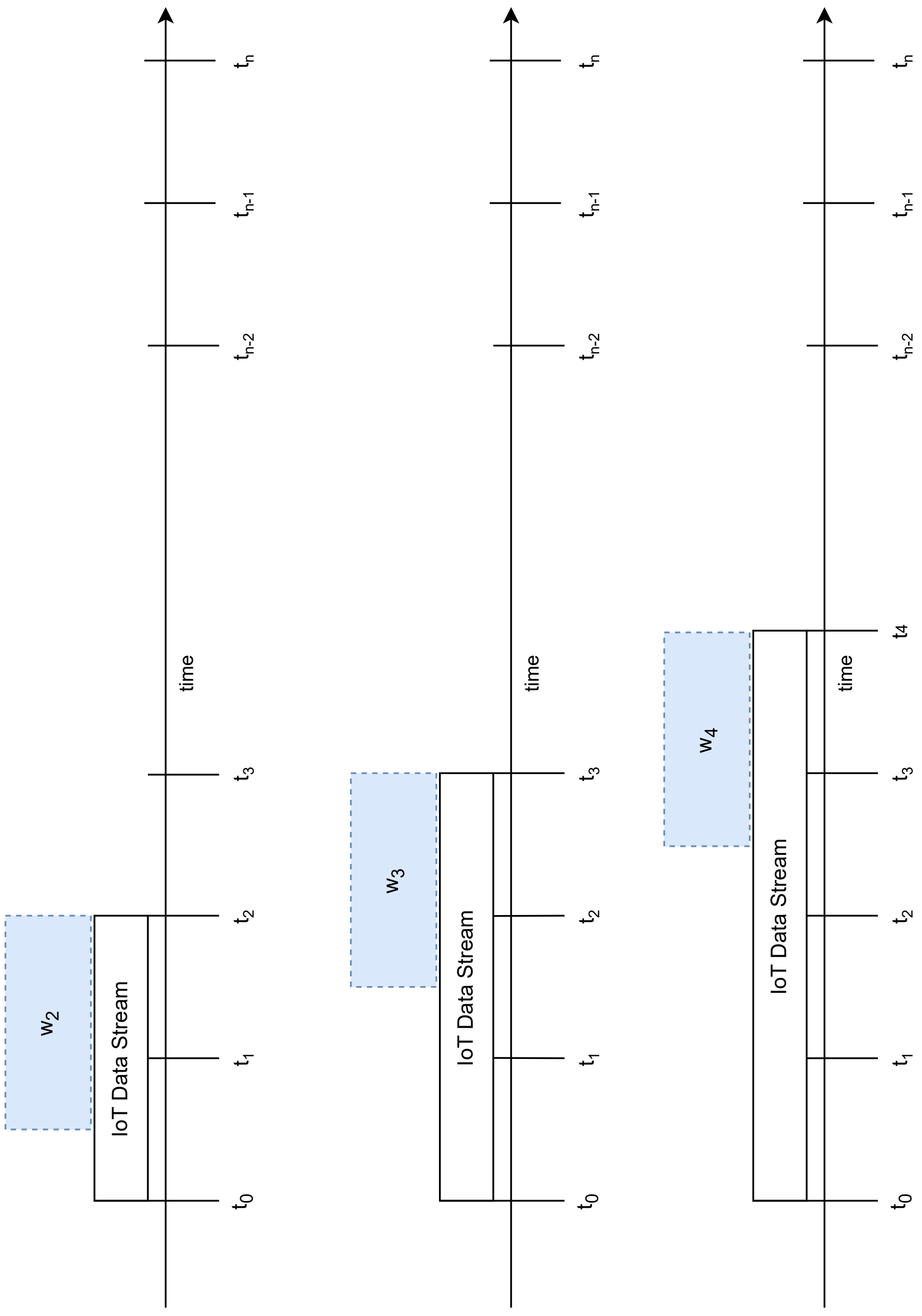


Figure 4.10: Example of Sliding Window for Automated Retraining

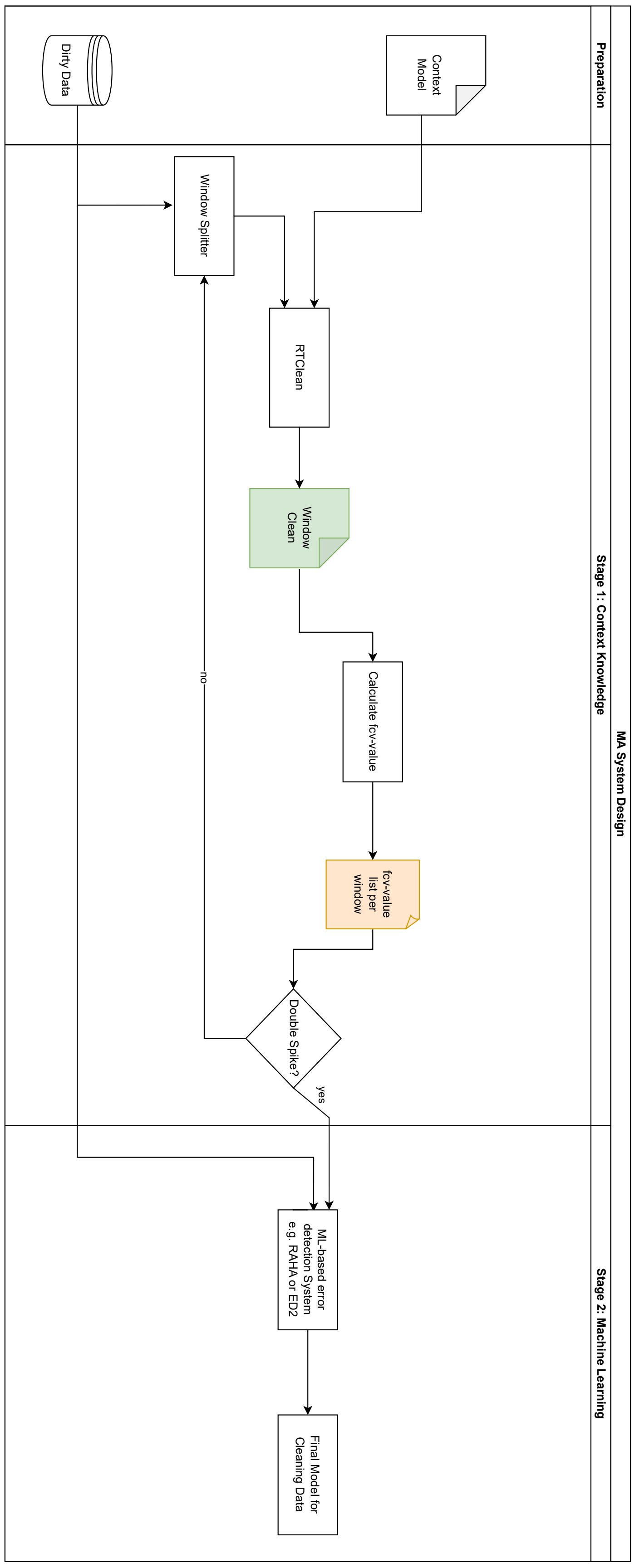

MA System Design
Preparation
Stage 1: Context Knowledge
Stage 2: Machine Learning
Dirty Data
Context Model
Window Splitter
RTClean
Window Clean
Calculate fcv-value
fcv-value list per window
Double Spike?
no
yes
ML-based error detection System e.g. RAHA or ED2
Final Model for Cleaning Data

Figure 4.11: Two-Stage Pipeline for Automated Retraining of ML models

5 Implementation

This chapter presents some details of our proof-of-concept implementation. We explain how we implemented the previously presented approaches with the corresponding tooling. We begin in Section 5.1 with an introduction to our active learning algorithm, namely ED2. Afterward, in Section 5.2, we present some highlights and concepts used to implement the first stage of our proposed pipeline. In Section 5.3, we presented our used tools and the machine we worked on. Lastly, Section 5.4 highlights the challenges we faced in intetragting research tools into our pipeline.

We implemented the previously explained approaches for our proof-of-concept implementation. In general, we followed the following structure. For the first Stage, we used the publicly available RTClean version and integrated it into our pipeline. Therefore, we created Python scripts that execute RTClean on a particular dataset with a corresponding context model. After each step in the pipelines, we store the intermediary results in CSV files. This is done to increase the interchangeable nature of the proposed pipeline. This ensures that in Future Work, when our implementation is used, one would only need to change the swap out the approach we used and store the results in the corresponding CSV files. Additionally, this ensured that possible extensions should be facilitated. In our case, we utilized in Stage 1 RTClean [DSA23] and in Stage 2 ED2 [NMA19].

5.1 Stage 2 Implementation

We utilize ED2 for the second stage because it is an active learning framework that yields promising performance in the literature described in Section 3.2. It is important to know how our implementation deviates from traditional active learning approaches. Therefore, we illustrate the functionality of ED2 in Figure 5.1. Unlike other active learning approaches, ED2 automatically generates the features it uses to describe the dataset. This means that before it starts the active learning process, it uses different mechanisms, as mentioned in [NMA19] to derive the features it finds feasible from the dataset. Then, it starts the active learning as explained in 2.3. It provides unlabeled instances from the dirty dataset through an active learning component to the oracle. The oracle then labels the instances provided. The active learning component monitors the model to decide which instances to forward to the oracle. This process is repeated until the labeling budget runs out.

In our pipeline implementation, we added adapters to ED2, which allow it to run with different configurations. These adapters allow us to run ED2 with different labeling budgets, with different data as oracle, and extend the features generated from the feature generator. For Approach 3: Feature Extension as described in Section 4.4, we needed the adapter on the feature generator. This adapter transforms data into the format required by the feature generator and appends it to the end of the feature generated. This adapter is then used to provide the prelabeling to the features file

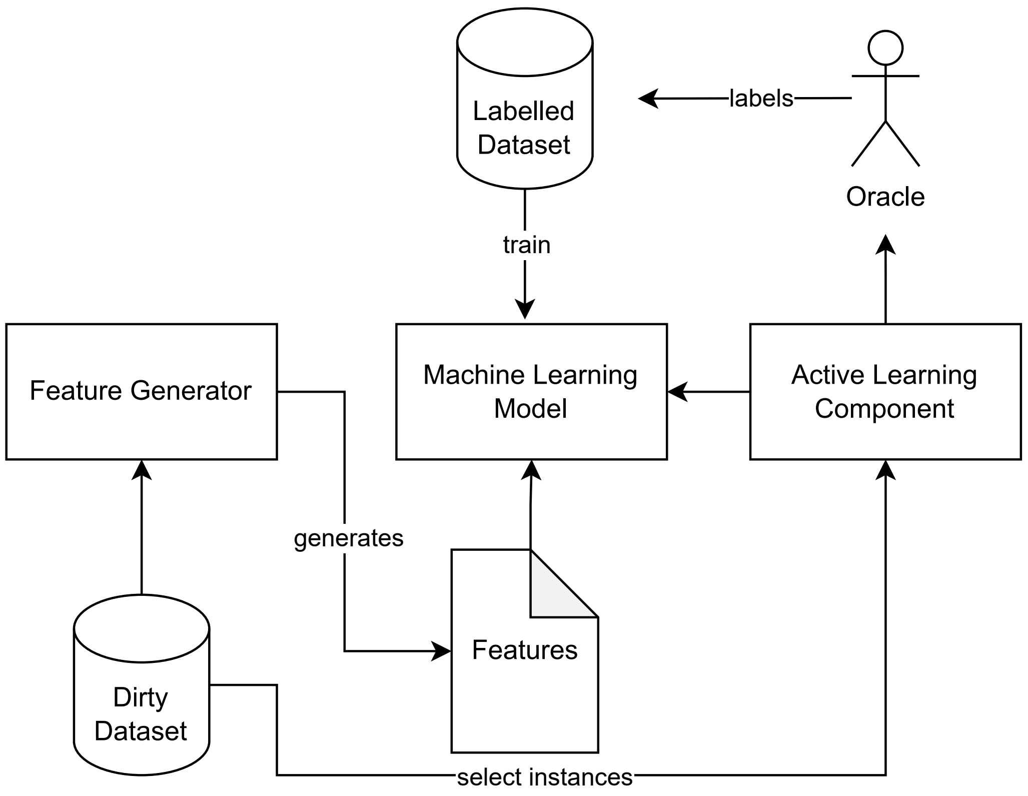


Figure 5.1: Conceptual Workflow of ED2 [NMA19]

used by ED2. For Approach 1: Prelabeling and Approach 2: Ensemble, we created an adapter that replaces the oracle with the data derived from the prelabeling. Finally, we needed an adapter that encapsulates the whole ED2 workflow to run it with different configurations of the labeling budget.

In the implementation of Approach 3, we needed to extract the features generated by ED2. Therefore, we implemented a component that interrupts the execution of ED2 after it generates the features. Before it starts training the model, we extend the features by the prelabeling of ED2.

5.2 Stage 1 Implementation

Our proof of concept implementation for the first Stage is mainly built using RTClean [DSA23] for context aware error detection and smart adapters that allow it to be used. For the implementation of Stage 1, we encapsulated RTClean in a script. This allows us to rerun RTClean on different datasets and handle different environments. RTClean stores the results in a PostgreSQL[1] database. We extract those results, store and forward them to Stage 2. For our further improvement step, we

[1]PostgreSQL: https://www.postgresql.org

calculated the *fcv*-value for each column with the help of the original dirty data. For each column where the value exceeded the user-defined threshold our implementation undos all the columns values. Afterwards, the data is forwarded to Stage 2.

For the automated retraining, we extended our implementation with capabilities to split the data into windows according to the window and step size. We implemented the sliding window and corresponding evaluation with special tooling except for pandas. Again, we followed the principle of storing everything in CSV files, allowing for a flexible and interchangeable implementation for future work.

Lastly, we highlighted that we implemented Approach 2 using RTClean, a missing values detector, and a duplicate detector. We will not go into the details of the implementation of this approach since we were unable to time constraints in this thesis to evaluate it. Still, we contributed our implementation of the approach for future research.

5.3 Used Tools and Machine

In this section, we present the tooling we used for our implementation. The instructions on how to set it up and requirements are present in the associated publicly available version control system.

We build our implementation using **Python**[2]. We used Python as it allows us to easily run different algorithms together and perform the appropriate data processing to handle data between different processes. For the processing we mainly used **Pandas**[3] and **NumPy**[4]. They provided us with the tooling to implement the processing required to realize and evaluate all our approaches.

We implemented everything and ran all the evaluations on the machine:

- MacBook Pro 2021 M1 Pro Chip
- 512 GB SSD
- 16 GB RAM
- MacOS 13.2.1

5.4 Challenges

Additionally, we wanted to highlight challenges when implementing our proposed pipeline. We have designed our pipeline and approaches to allow the implementation to interchange the approaches used easily. We found that many implementations of error detection approaches are very tightly coupled and need more effort to be used outside the proposed paper they are presented in. This means that when, for example, changing the ML-based error detection approach, our architecture and implementation allow for this very easily. However, getting another approach up and running on the system has shown to be particularly difficult. We first used RAHA for ML-based error detection.

[2]Python: https://www.python.org
[3]Pandas: https://pandas.org
[4]NumPy: https://numpy.org

Getting up and running on a dataset that the developers did not provide was challenging. RTClean also requires many configurations before usage and has rather limited flexibility for other datasets. This all consumed a lot of time when creating the proof of concept and is necessary for further development.

6 Evaluation

This chapter delves into the assessment of the proposed approaches. Beginning with the datasets description, the chapter then explains the evaluation procedure, metrics, and results, concluding with a comprehensive discussion. Section 6.1 introduces the dataset that is used for the evaluation. Section 6.2 explains the evaluation procedure. Section 6.3 contains results of RTClean and ED2 evaluation that serve as a basis of comparison for our approaches. Section 6.4 contains the evaluation of our Approach 1 implementation. Section 6.5 contains the results of the evaluation of our Approach 3. In Section 6.6, we present the results of our Approach 1 and 3 implementation including the proposed improvement. Section 6.7 presents the results of the analysis if our pipeline can detect context changes. Section 6.8 concludes the evaluation chapter with a discussion about the meaning and implications of our approaches.

6.1 IoT Dataset

The evaluation employs the dataset introduced by Schubert et al. [Sch22]. They developed a real-world dataset created from an IoT environment. Most publicly available datasets contain the data without any additional context information. And the datasets which contain context information do not include context information which can be used by RTClean. That is why we chose to use the self-recoded dataset of Schubert et al. [Sch22] from a smart home application.

Figure 6.1 presents the IoT environment leveraged by Schubert et al. [Sch22] to generate their labeled data. Four temperature sensors, distributed across three rooms, accumulated temperature readings every hour over a period of six days. The dataset structure, as illustrated in Figure 6.2, comprises 7000 datapoints across 1000 rows and seven columns. An open-source tool[1] was used to inject errors of various types, such as typos, value discrepancies, and null values, amounting to 15% of the total dataset.

6.2 Evaluation Procedure

This section describes the evaluation methodology adopted to assess the effectiveness and efficiency of the proposed pipelines.

[1]https://github.com/BigDaMa/error-generator

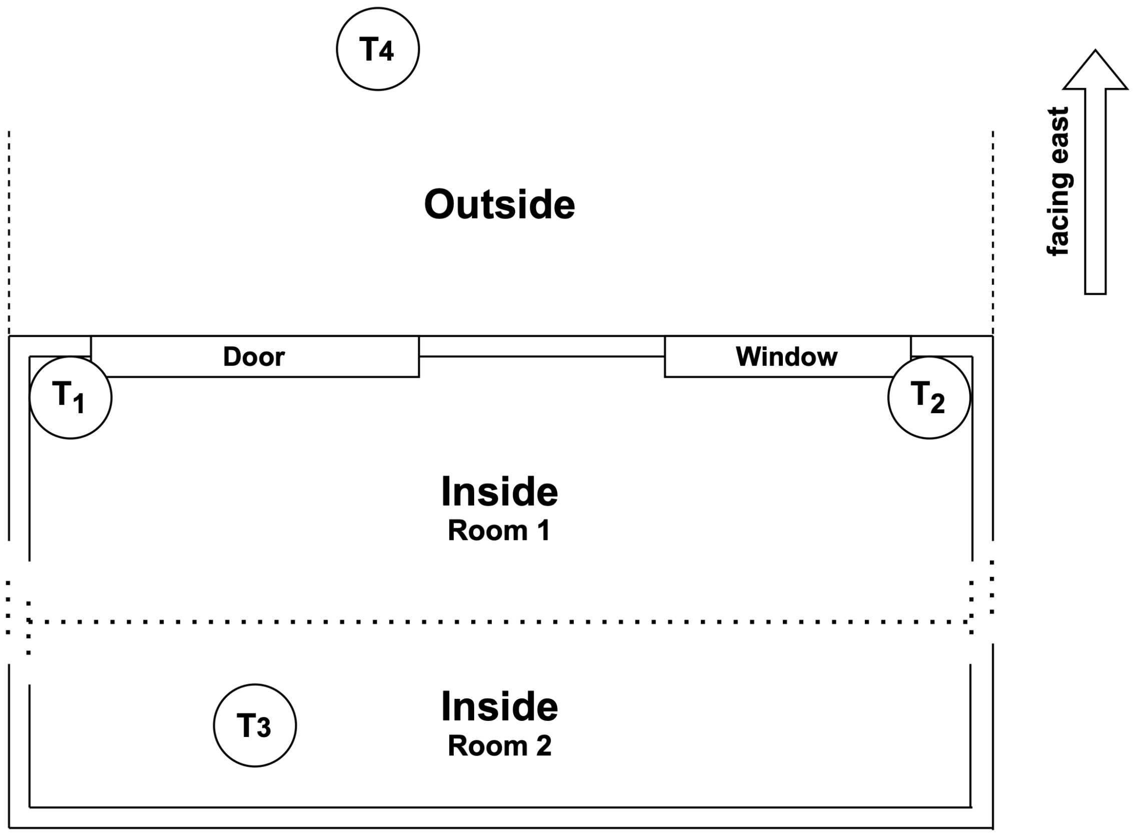


Figure 6.1: Sensor Layout of Evaluation Dataset [Sch22]

Device	**SensingDevice**	**Sensor**	**name**	**value**	**timestamp**	**location**
device_out	aqara_multisensor_2	aqara_temp_2	t4	7.08	2021-01-01 02:00:00	outside
device_in_1	esp8266_2	ds18b20_2	t2	23.69	2021-01-01 02:00:00	room1
device_in_1	esp8266_1	ds18b20_1	t1	22.69	2021-01-01 02:00:00	room1
device_in_2	aqara_multisensor_1	aqara_temp_1	t3	22.32	2021-01-01 02:00:00	room2
device_main	raspberry		processor	7.08	2021-01-01 02:00:05	outside
device_main	raspberry		processor	23.69	2021-01-01 02:00:05	room1
device_main	raspberry		processor	22.69	2021-01-01 02:00:05	room1
device_main	raspberry		processor	22.32	2021-01-01 02:00:05	room2
device_in_1	esp8266_1	ds18b20_1	t1	22.46	2021-01-01 03:00:00	room1
device_out	aqara_multisensor_2	aqara_temp_2	t4	7.25	2021-01-01 03:00:00	outside
...	...	...	...	...	...	...

Figure 6.2: An Example of the IoT Data used in the Evaluation [Sch22]

6.2.1 Objective

The primary objective of this evaluation is to find out:

1. Determine whether the entire pipeline operates faster than solely employing an ML-based error detector. Therefore, we check whether our proposed pipeline performs more or similarly accurately without human intervention. If this is the case, then our pipeline can replace the human from ML-based error detection. Similarly, we need to check if our pipeline with human intervention but with extended features performs well. If this is the case, we can reduce the number of features generated automatically and use our extended features instead of reducing the time it takes to generate them.

2. Assess if our pipeline increases the predictive accuracy compared to using the pipelines component algorithms independently. We assess this by analyzing our approaches' F1-score, precision, and recall and comparing the results to the results of RTClean and ED2 alone. If the measured accuracy does improve, it shows that using the pipeline is better than using the algorithms on their own.

6.2.2 Baseline Comparisons

For a comprehensive evaluation, our proposed pipelines are compared against the following baseline algorithms:

- The context-aware error detection baseline, namely RTClean.
- ML-based error detection baseline, namely ED2.

6.2.3 Evaluation Scenarios

Our evaluation encompasses two distinct scenarios:

1. Evaluation without human intervention for the analysis of Approach 1: Prelabeling as described in Section 4.2.

2. Evaluation incorporating human input for the analysis of Approach 3: Feature Extension as described in Section 4.4.

Given that some of our approaches either mandate or exclude user input, distinguishing between these scenarios is crucial. To authentically emulate human input during active learning, the ground truth version of the IoT dataset is utilized. Within the evaluation framework, this ground truth functions as a simulated human oracle for active learning. The extent of human annotation is regulated via the labelling budget.

6.2.4 Experimental Setup

The experimental protocol is as follows: Run each approach on the IoT dataset. This produces a model whose performance is then evaluated. The performance of this model, derived from our pipeline, is then compared against a benchmark. This benchmark is set by performing the same error detection task using only ED2 and RTClean. The purpose of this comparison is to determine whether our pipeline improves the outcomes of the baseline algorithms. It investigates whether an integrated approach, as proposed by our pipelines, actually enhances performance, or if the individual algorithms, in their original forms, are more effective.

6.2.5 Evaluation Metrics

To objectively assess the performance of our approaches, we use a combination of both established and custom metrics. These metrics cater to different aspects of error detection and data cleaning. Detailed descriptions and mathematical representations of these metrics are provided below:

Detected Error: This metric quantifies the total number of errors identified by the error detection mechanism.

Correct Cleaning: Represents the number of accurate corrections made to values that were previously identified as errors.

Total Errors: Signifies the total number of errors present in the dataset.

Total Cleaning: Counts the total number of cleaning operations performed on the tainted data.

Labeling Budget: The amount of instances which have to be labeled by the oracle in ED2s active learning process.

Fraction of Changed Values by RTClean: Represented as the proportion of records changed by RTClean.

Precision: Precision measures the proportion of identified errors that are actual errors. Mathematically,

$$precision = \frac{TP}{TP + FP}$$

where:

- TP = True Positive
- FP = False Positive

Recall: Recall calculates the fraction of actual errors that the system successfully detects. It is given by:

$$recall = \frac{TP}{TP + FN}$$

where:

- TP = True Positive

- FN = False Negative

F1-score: F1-score provides a balanced metric, calculated as the harmonic mean of precision and recall. It is expressed as:

$$F1score = 2 * \frac{precision \times recall}{precision + recall}$$

	Actual	
	Positive	Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Table 6.1: Confusion Matrix

The table above, known as the Confusion Matrix, is fundamental for understanding classification performance.

6.3 Baseline Results

This section presents the performance results of our baseline approaches: RTClean and ED2. Establishing the independent effectiveness of these algorithms is crucial to understand the potential improvements offered by our proposed methods. Therefore, we assess their individual performance first, allowing us to make a comprehensive comparison with our proposed pipelines in the subsequent discussion.

6.3.1 Experimental Setup

We ran both RTClean and ED2 on the specified testbed, detailed in 5.3. The execution times for both methods were recorded across different labeling budgets (lb) for ED2. This was done to gauge its scalability and its adaptability to varying amounts of human-annotated data.

Algorithm	Execution Time (s)
RTClean	24.33
ED2 (50 lb)	39.9
ED2 (75 lb)	45.14
ED2 (100 lb)	47.77
ED2 (200 lb)	63.02
ED2 (500 lb)	70.86

Table 6.2: Execution Times for RTClean and ED2 across varied Labeling Budgets.

The execution time for RTClean consistently averages around 25 seconds on our data. In contrast, ED2's execution time varies between 40 to 70 seconds, contingent upon the labeling budget.

6.3.2 RTClean Performance Analysis

The RTClean results, as shown in Table 6.2 indicate an F1-score of 0.36, with a precision of 0.78 and a recall of 0.23. The cleaning recall stands at 0.36, and the cleaning F1-score is 0.5. Notably, the data reveals that significantly more values are cleaned than the number of errors present. Specifically, 541 values are cleaned, despite there being only 138 actual errors in the dataset.

Metric	Score	Metric	Score	Metric	Count
Precision	0.78	Cleaning Recall	0.36	Detected Errors	88
Recall	0.23	Cleaning Precision	0.81	Correct Cleaning	32
F1-score	0.36	Cleaning F1-score	0.5	Total Errors	138
				Total Cleaning	541

Table 6.3: Performance Metrics for RTClean.

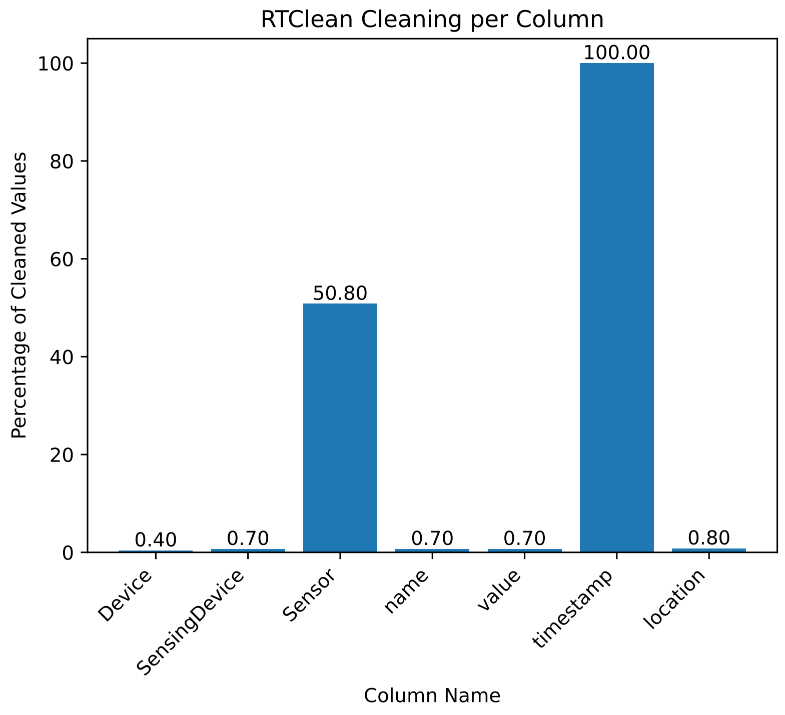


Figure 6.3: Proportion of Cleaning Operations by Column for RTClean.

Figure 6.3 indicates that the bulk of the unnecessary cleaning occurs in the Sensor and Timestamp columns.

6.3.3 ED2 Performance Analysis

To gain a deeper insight into ED2's capabilities, we evaluated it under various labeling budgets. This approach shed light on how the inclusion of a human oracle, achieved through labeling, affects the algorithm's performance.

As illustrated in Figure 6.4, the F1-score noticeably rises with an increase in the labeling budget. Specifically, with a limited budget of 50, the F1-score is 0.28. This score sees a marked improvement, reaching 0.73, when the labeling budget is expanded to 500.

It is crucial to emphasize that if the labeling budget is set too small, the algorithm's effectiveness diminishes, as demonstrated by the F1-score dropping to zero with an insufficient labeling budget.

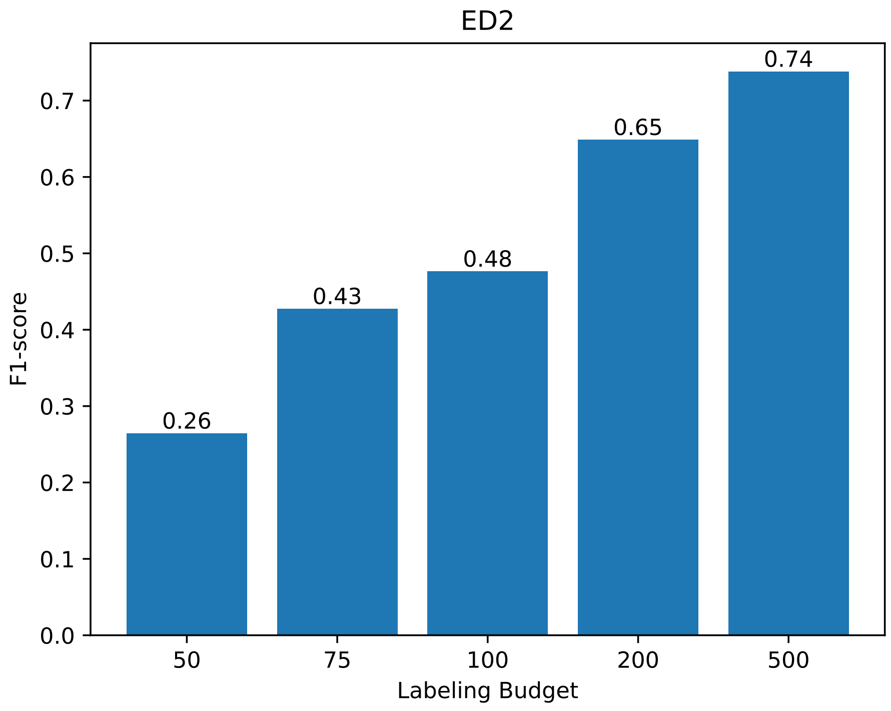


Figure 6.4: F1-score of only using ED2 under Diverse Labeling Budgets.

Both RTClean and ED2 exhibit unique performance patterns under the given conditions. This baseline analysis lays the groundwork for future evaluations, wherein these algorithms are incorporated into intricate pipelines, and their combined potential is delved into more deeply.

6.4 Approach 1: Prelabeling Results

In this section, we explore the results derived from our Approach 1: Prelabeling. Using the provided dataset, we assessed the efficacy of the resulting error detection model.

Referring to Figure 6.5, the F1-score exhibits a discernible trend in relation to the labeling budget. For budgets up to 100, the F1-score remains approximately at 0.01. However, with an increase in the budget, there's a consistent rise, culminating in an F1-score of 0.05. This pattern reveals that, despite our model's consistent suboptimal performance, it experiences improvement with an increased labeling budget.

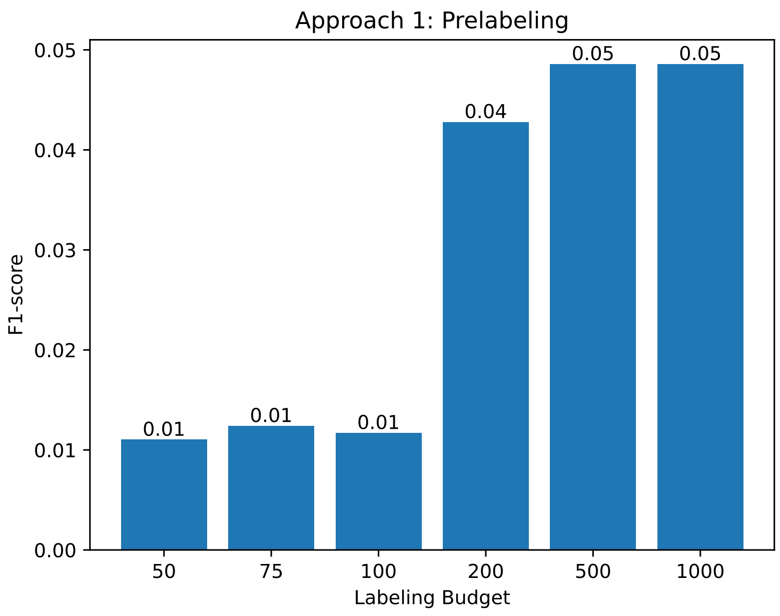


Figure 6.5: F1-score Results from Approach 1: Prelabeling using RTClean for Prelabeling and then utilizing Prelabeling as Oracle for ED2

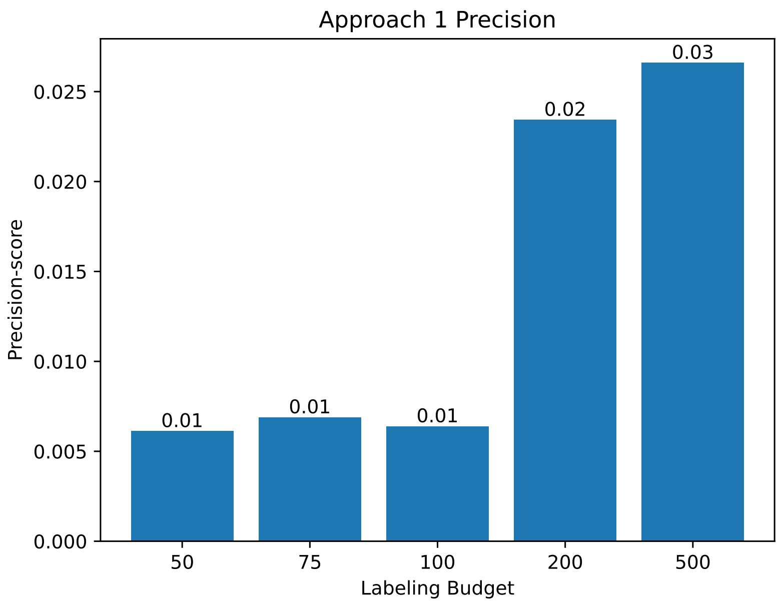


Figure 6.6: Precision Results from Approach 1: Prelabeling using RTClean for Prelabeling and then utilizing Prelabeling as Oracle for ED2

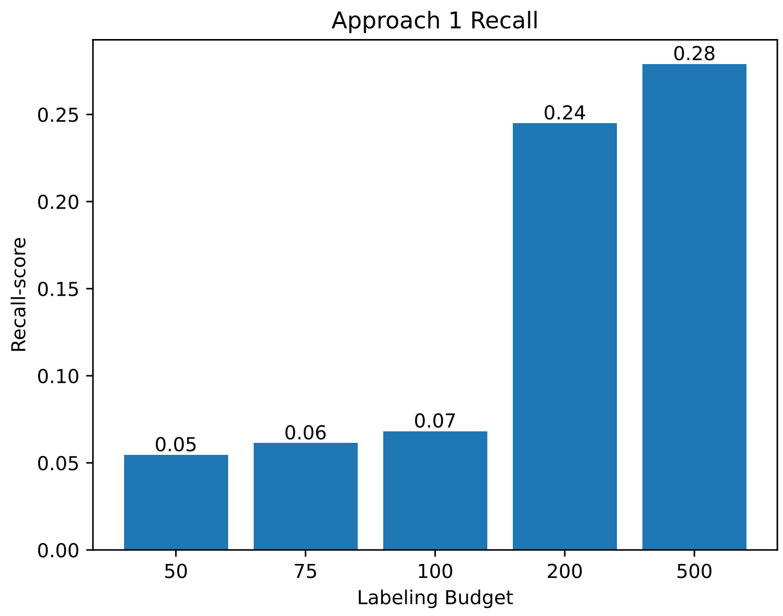


Figure 6.7: Recall Results from Approach 1: Prelabeling using RTClean for Prelabeling and then utilizing Prelabeling as Oracle for ED2

Figures 6.6 and 6.7 provide insights into the precision and recall metrics, respectively. Up to a labeling budget of 100, both metrics register modest figures. However, beyond the 200-point threshold in the labeling budget, a significant divergence is observed. For budgets of 200 and 500, precision presents values of approximately 0.02 and 0.03, respectively. In contrast, recall displays a more marked increase, reaching 0.24 and 0.28 for the same budgets. Understanding the relationship between these metrics is pivotal. The elevated recall suggests our model excels at detecting true positives. Yet, the relatively lower precision indicates a surge in false positives with the expansion of the labeling budget. This interrelation directly affects the F1-score, which aims to strike a harmonious balance between precision and recall. Furthermore, the F1-score's consistency up to a budget of 100, followed by its noticeable rise post the 200-point, presents an intriguing avenue for deeper investigation. Through Approach 1: Prelabeling, our findings emphasize the significance of an ample labeling budget. While precision and recall furnish a detailed perspective on model performance, the ascending F1-score with an increasing budget offers hope regarding the model's error detection capabilities, despite the challenge posed by the diminished precision.

6.5 Approach 3: Feature Extension Results

In this section, we analyze the outcomes derived from Approach 3: Feature Extension. Within this approach, we reintegrated the user into the active learning process and assessed performance across various labeling budgets.

Directing attention to Figure 6.8, we note an evident pattern in F1-scores in regards to the labeling budget. At all of the budgets, the F1-score settles at a modest 0.01-0.05 range. In contrast to Approach 1, the F1-score decreased with an increase in labeling budget.

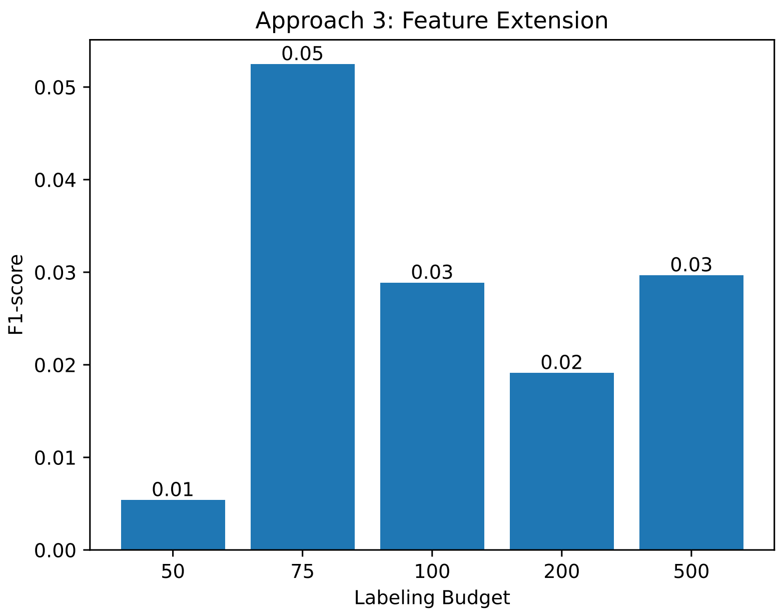


Figure 6.8: F1-score Results from Approach 3: Feature Extension using RTClean for Prelabeling and extending the Features of ED2 with Prelabeling

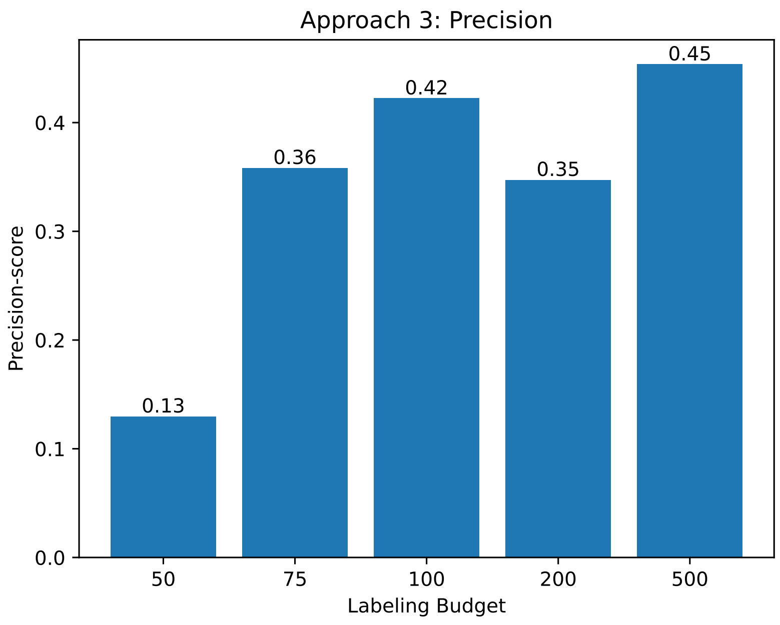


Figure 6.9: Precision Results from Approach 3: Feature Extension using RTClean for Prelabeling and Extending the Features of ED2 with Prelabeling

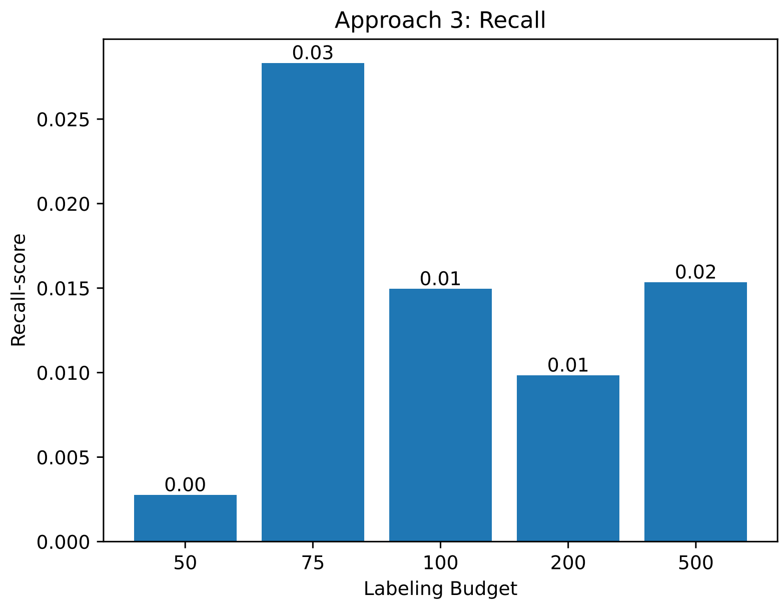


Figure 6.10: Recall Results from Approach 3: Feature Extension using RTClean for Prelabeling and extending the Features of ED2 with Prelabeling

As depicted in Figure 6.9, precision demonstrates a steady ascent with increasing labeling budgets. Starting from 0.13 for a budget of 50, the precision climbs to 0.36, 0.42, 0.35, and ultimately peaks at 0.45 for labeling budgets of 75, 100, 200, and 500, respectively. This continual rise underscores that, with an augmented set of features and an expanded labeling budget, the model effectively diminishes the occurrence of false positives.

An examination of Figure 6.10 yields notable observations. While the recall score for a budget of 50 is virtually non-existent, it sees a significant jump to 0.03 with a budget of 75. The subsequent budgets of 100, 200, and 500 show recall values of 0.01, 0.01, and 0.02, respectively. The noticeable peak at a budget of 75, followed by a decline, indicates specific feature interactions at this budgeting threshold that merit further analysis. In contrast to Approach 1, where recall increased with the labeling budget, the recall in this approach remains relatively low.

Approach 3: Feature Extension emphasizes the advantages of feature extension with respect to precision. However, the F1-score remains comparatively low due to the subdued recall.

6.6 Results for Enhancements Using User-defined Thresholds

In this segment, we present the results of our suggested enhancement using user-defined thresholds. Accordingly, we evaluated both Approach 1 and 3 with the said improvement integrated.

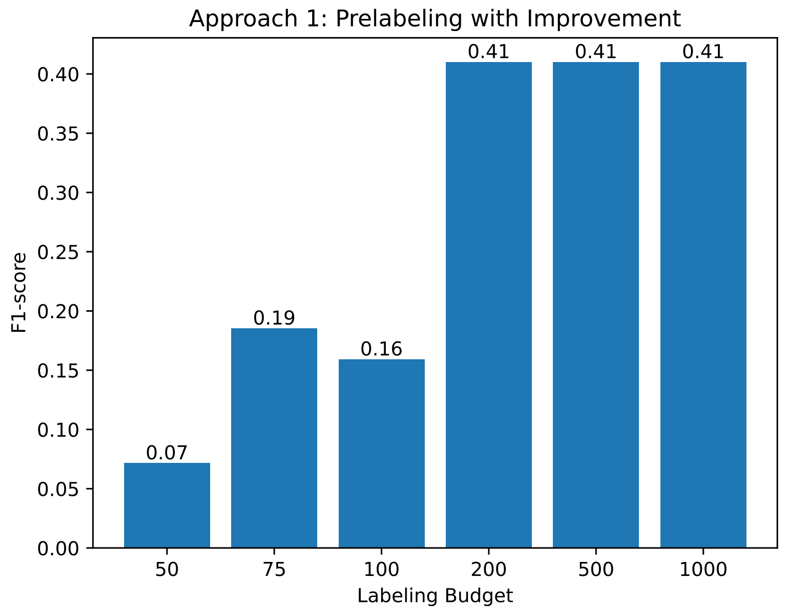


Figure 6.11: Results Approach 1: Prelabeling with User Input

6.6.1 Enhanced Approach 1: Prelabeling

Here, we detail the performance metrics of Approach 1 when incorporating the threshold improvement across various labeling budgets.

Starting with a labeling budget of 50, Figure 6.11 indicates an initial F1-score of 0.07. As we progressively increase the labeling budgets, there's a noticeable trend in the F1-score: it rises to 0.19 at 75, then slightly dips to 0.16 at 100, before stabilizing at a commendable 0.41 for budgets of 200, 500, and 1000.

Starting with a labeling budget of 50, the precision score was recorded at 0.29, indicating a fair accuracy of the model's positive predictions. Impressively, by increasing the labeling budget to 75, the precision shot up to a flawless score of 1.00. This impeccable precision was maintained for the following labeling budgets of 200 and 500. However, it observed a dip, settling at 0.51, when the labeling budget was set to 100.

For a labeling budget of 50, the recall score was a mere 0.04, indicating the model's struggles with pinpointing all true positive cases. As we raised the labeling budgets to 75 and 100, there was a modest increase in recall to 0.10, suggesting that a vast majority (90%) of the true positives were still undetected. When we allocated larger labeling budgets of 200 and 500, the recall further improved to 0.22. However, this still highlighted the model's persistent challenges in recognizing a significant number of true positives.

By integrating improvements into Approach 1, we observed a noticeable enhancement in precision across most labeling budgets. Yet, recall, even with its incremental growth, was significantly outpaced by precision. This reflects the model's emphasis on making accurate predictions at the possible expense of missing a large proportion of actual positives. The resulting F1-score

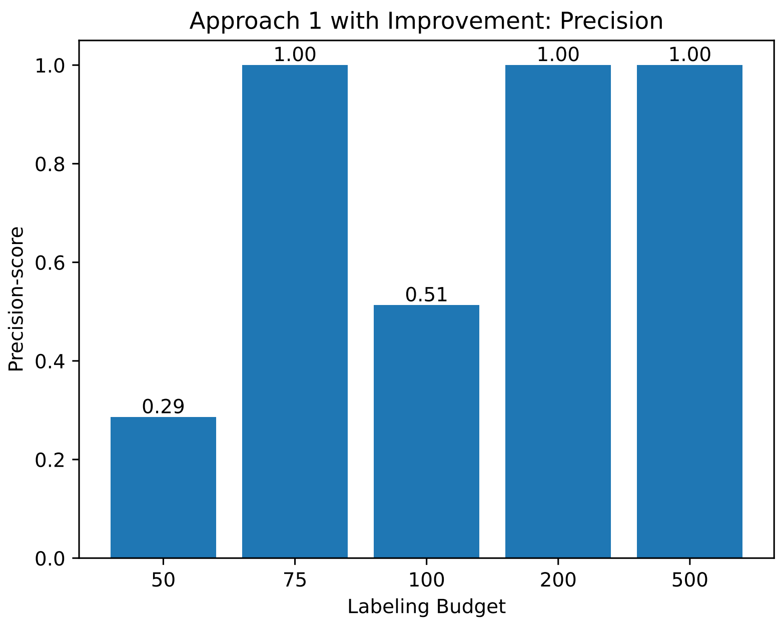


Figure 6.12: Results in terms of Precision from Approach 1: Prelabeling with User Threshold Improvement

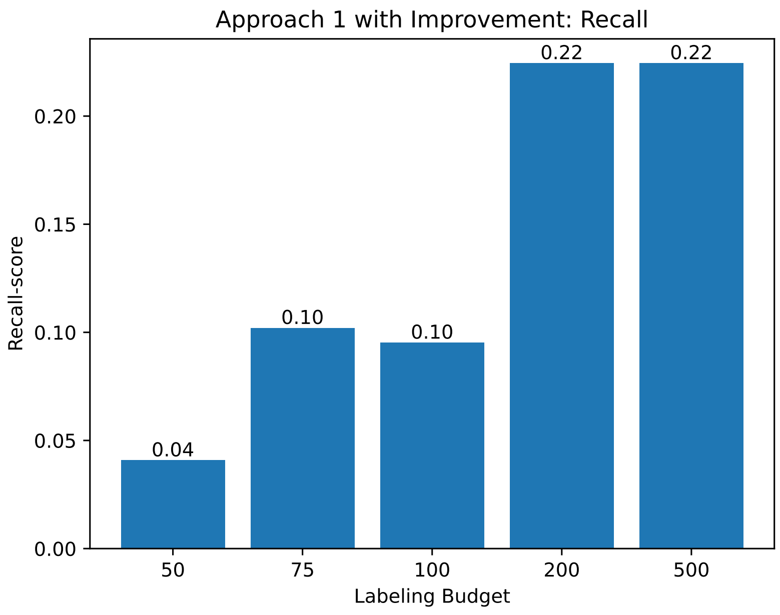


Figure 6.13: Results in terms of Recall from Approach 1: Prelabeling with User Threshold Improvement

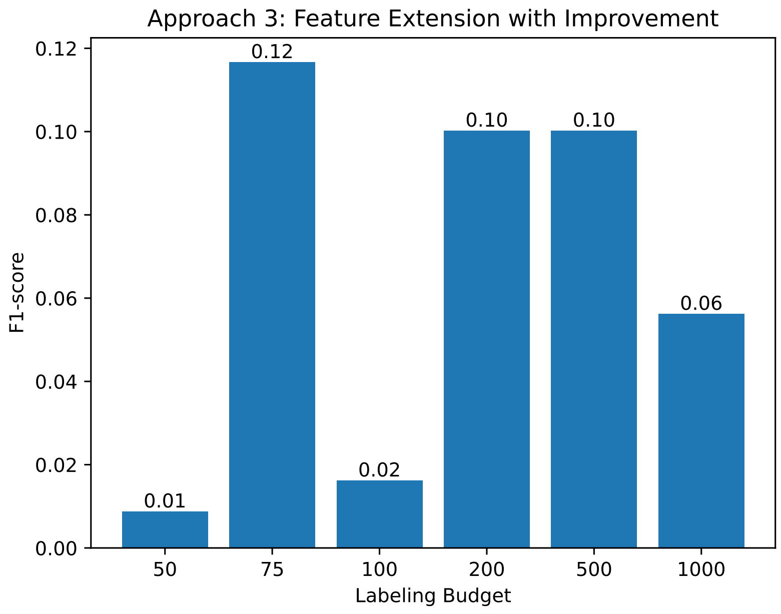


Figure 6.14: Results Approach 3: Feature Extension with User Input

underscores this balance, demonstrating the model's enhanced capability with increased labeling budgets. Its stabilization at the higher budgets suggests potential diminishing returns, implying that additional budget increases might not substantially boost the model's effectiveness.

6.6.2 Enhanced Approach 3: Feature Extension

The evaluation of Approach 3, post-enhancement, is particularly intriguing, given its emphasis on feature extension. By segmenting the analysis based on varying labeling budgets, we are better positioned to discern the intricate connections between allocated resources and resultant performance metrics. This perspective not only aids in understanding the efficacy of the approach itself but also offers insights into the strategic allocation of budgets to optimize outcomes.

In Figure 6.14, the progression of the F1-score across various labeling budgets provides valuable insights into the model's performance dynamics: 1. Starting with a modest labeling budget of 50, the F1-score is at a mere 0.01. This suggests that, initially, there's minimal harmony between precision and recall, reflecting the model's struggles in striking a balance between the two metrics. 2. A significant improvement is observed when the labeling budget is increased to 75, with the F1-score surging to 0.12. This uptick underscores a substantial betterment in the model's ability to concurrently optimize precision and recall. 3. However, this momentum is momentarily halted as the F1-score dips to 0.02 at a labeling budget of 100, indicating a potential inconsistency in the model's performance or certain challenges specific to this budget threshold. 4. The subsequent budgets of 200 and 500 see the F1-score ascend to 0.1, suggesting a resurgence in the model's harmonized performance. This consistency across these two budgets might be indicative of the model stabilizing its performance with an expanded set of data. 5. Intriguingly, at a presumably generous labeling budget, the F1-score experiences a downturn to 0.6. This decrease, despite the model having access to more labeled data, points towards potential overfitting or diminishing returns

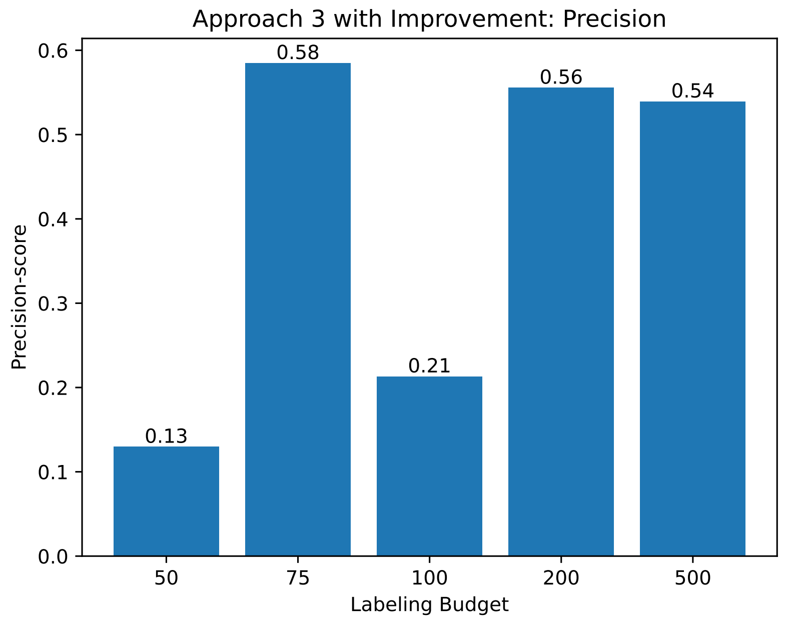


Figure 6.15: Results in terms of Precision from Approach 3: Feature Extension with User Threshold Improvement

where additional labeled data fails to contribute to significant performance gains. The journey of the F1-score across these labeling budgets underscores the intricacies and challenges of optimizing a model's performance. It becomes evident that merely increasing the labeling budget does not guarantee linear improvements, and strategic considerations are essential for maximizing returns on labeled data investments.

Precision sheds light on the models ability to correctly identify positive instances. As shown in Figure 6.9, at labeling budgets of 50, 200, and 500, the precision achieved its peak at around 0.56. However, for labeling budgets of 50 and 100, the scores were lower, registering at 0.13 and 0.21, respectively.

The recall analysis revealed the highest values at the 75, 200, and 500 labeling budgets, mirroring the trends seen in the precision and F1-score graphs. Nevertheless, throughout the recall results, the values remained comparatively low, ranging from 0 to 0.06.

The enhanced version of Approach 3 demonstrated varied performance across different metrics. While precision achieved relatively high results, both recall and F1-score remained comparatively low.

6.7 Automated Retraining Results

For the evaluation of the automated retraining approach, we sought to determine the optimal timing for retraining machine learning models in changing environments. To do this, we examined whether RTClean could detect changes in the context. Specifically, we analyzed RTClean's cleaning behavior in various scenarios where the sensor locations were altered.

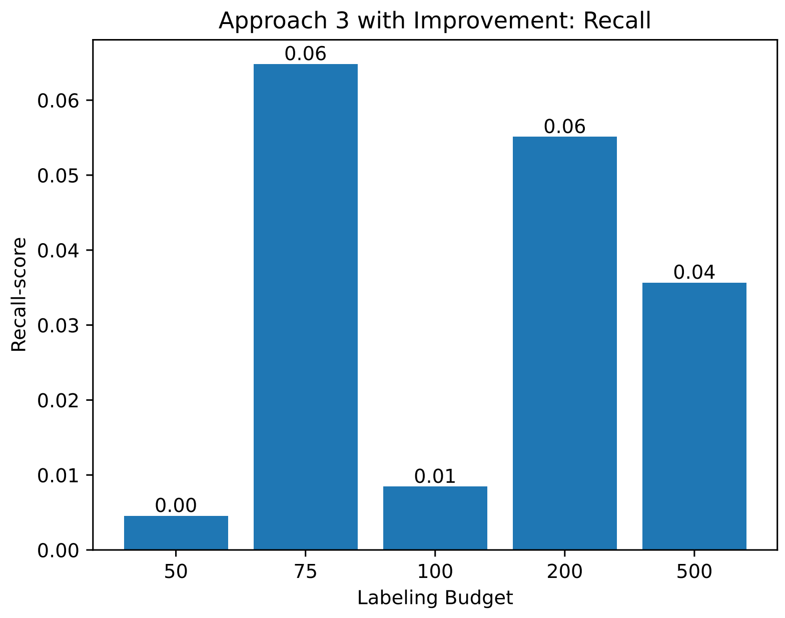


Figure 6.16: Results in terms of Recall from Approach 3: Feature Extension with User Threshold Improvement

We created three versions of the dataset presented in Section 6.1.

- Baseline: A baseline scenario where no location is changed.
- Spike: A scenario where a sensor shifts its location for a short period of time.
- Durable: A scenario where a sensor shifts its loction for the remainder of the time.

These scenarios are designed to help us determine whether RTClean recognizes context changes. For our procedure, we utilized the dataset with 15% errors and simulated an environment change by relocating the sensor from the outside to Room 1, adjusting its temperature readings accordingly. We subsequently assessed various configurations of our sliding window approach, considering different window and step sizes. For the evaluation, we selected a window size of 100 and a step size of 15, as these configurations produced the best outcomes. To ascertain whether RTClean detects the altered environments, we employed the *fcv*-value. This helped determine if RTClean, through extended cleaning, recognizes a shift in the environment. Our primary focus was on the cleaning occurring in the value column, as this is the intended point of identification.

In Figure 6.17, we display the results from executing RTClean in the baseline scenario. Our intent is to ascertain the standard cleaning behavior of RTClean when there are no contextual shifts. The x-axis represents the windows, while the y-axis shows the fraction of changed values for the corresponding window. As depicted in the figure, between 0% - 2% of the values are cleaned per window, signaling typical cleaning behavior.

In Figure 6.18, we present results analogous to our previous observations, but this time utilizing the spike version of the dataset. In this experiment, we temporarily relocated a sensor indoors and examined RTClean's cleaning behavior. The subsequent analysis of the fraction of cleaned values

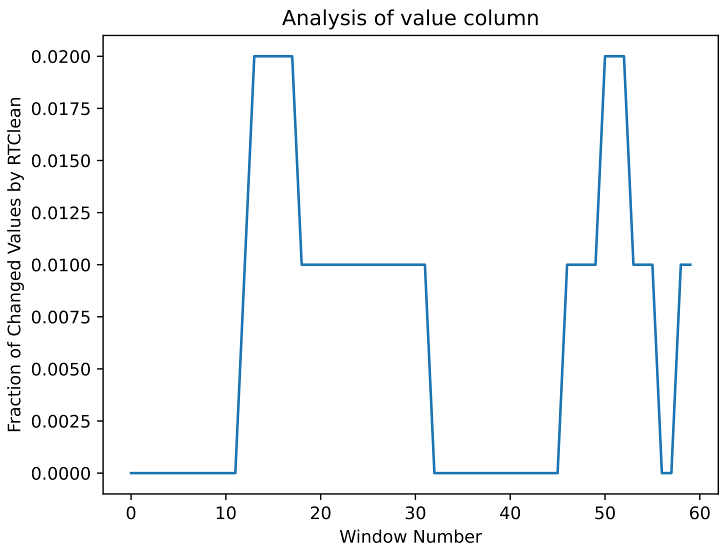


Figure 6.17: Analysis of the Value Column on the Baseline Dataset

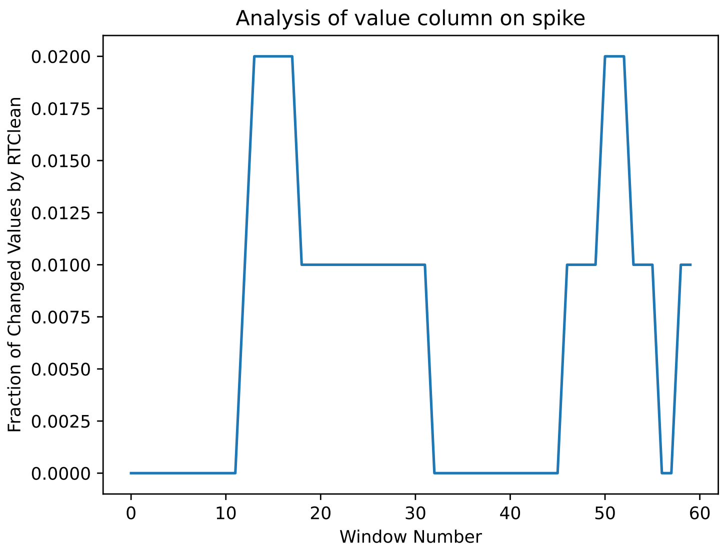


Figure 6.18: Analysis of the Value Column on the Spike Dataset

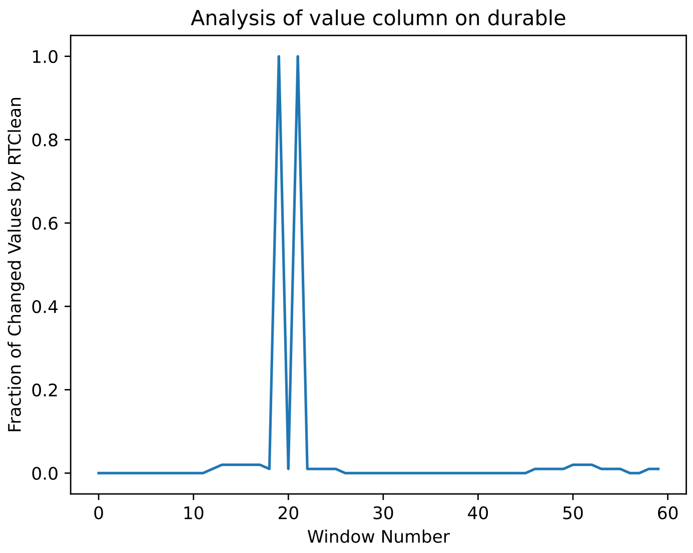


Figure 6.19: Analysis of the Value Column on the Durable Dataset

reveals a graph shape consistent with the baseline dataset. This suggests that an equivalent amount of cleaning occurred in both versions, indicating that RTClean did not clean the altered temperature values.

In Figure 6.19, we observe a transformed result graph when RTClean is applied to the 'durable' version of the dataset. For most windows, the value hovers close to zero, similar to the baseline. However, between windows 18 and 22, two pronounced spikes are evident: cleaning surges to 100%, drops back to zero, and then spikes to 100% again. These anomalies align with the period where significant durable changes begin to manifest. A comprehensive explanation for this occurrence is provided in the discussion section. For the time being, the observation that this double spike might signify context changes is noteworthy. Naturally, this is merely an indicator, and a thorough investigation of this phenomenon is essential to determine its potential for identifying context shifts.

6.8 Discussion

In the previous sections, we detailed the results stemming from our various methodologies. Now, we aim to discuss these findings, drawing potential insights and implications. Initially, we timed the runs of ED2 and RTClean on our hardware to gauge their execution time. Our primary objective was to demonstrate that the automated pipeline we advocate for offers competitive execution durations, indicating its practical viability. Essentially, the integration of these methodologies results in swift execution times when comparing against the duration-intensive task of manual instance labeling for active learning. This underscores the notable time benefits an automated tactic, like pre-labeling, offers by diminishing human intervention in the active learning process.

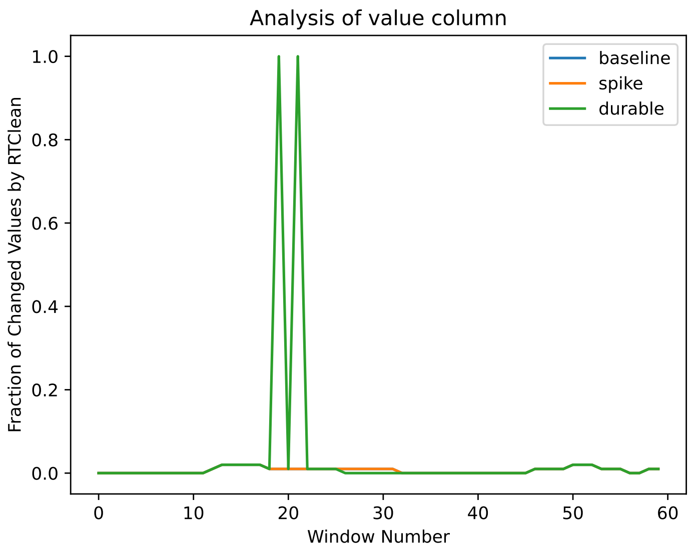


Figure 6.20: Analysis of *fcv*-value for the Column Value over all Versions of Dataset

To critically evaluate the effectiveness of our Approach 1: Prelabeling and its improved counterpart, we compared the outcomes against those procured using ED2 assisted by a human oracle. Data reveals that a human-supported ED2 surpasses both renditions of Approach 1 when the labeling budget exceeds 75. Such a revelation suggests that, strictly in performance terms, our proposition might not represent the optimal choice. However, contextualizing this, one must recognize that a 75-labeling budget is considerably substantial in real-world scenarios. Abdelaal et al.'s research [AHS23] posits that a pragmatic labeling budget for active learning typically oscillates between 20 and 50, contingent on the specific use-case. Given this perspective, our Approach 1: Prelabeling with improvement notably trumps the untouched ED2 variant, registering a 0.2 uptick in F1-score. Comparing the outputs of RTClean with our improved Approach 1 underscores the former's outperformance, thereby magnifying ED2's performance capabilities.

However, in the absence of our proposed augmentations, Approach 1 exhibits lackluster performance. This deficiency can be traced back to insights from Figure 6.3, which reveals RTClean's aggressive cleaning strategy, purging 50% and 100% of data across two columns. When such rigorously cleaned instances form the learning bedrock for ED2's active learning selector, the ensuing model internalizes flawed data facets. This predicament underscores the imperative for refining RTClean to curtail such overzealous cleaning. Should this challenge be neutralized, the additional enhancements we propose might become redundant. Our precision score analysis amplifies this sentiment; with our proposed refinement, we achieve flawless precision. This performance surge hints at a potentially stellar model in the offing if over-cleaning is reined in. However, the existing model's recall remains low. This suggests that while the model confidently identifies errors, its absence of error flagging does not necessarily imply error-free data. Given this revelation, our Approach 2: Ensemble can be further finessed. An intriguing research direction could be probing the synergy between RTClean and high-recall methodologies. By supplementing RTClean with high-recall tools, we might counterbalance its inherent low recall vulnerability, positioning it to more comprehensively flag errors.

Additionally, we wanted to discuss the results from Approach 3: Feature Extension. Surprisingly, both versions, with and without improvement, did not perform well even though we reintroduced the human element into the pipeline. This subpar performance suggests that the extended features might have 'confused' the machine learning model. This was evident in the significant dip in recall when using the feature extension approach. With a value close to 0.1, the recall undermined the significance of this approach. Nonetheless, we remain optimistic about future research on this approach. We believe the lackluster performance in the recall of RTClean was the primary hindrance to the success of Approach 3. Comparing Approach 3 with improved prelabeling would be intriguing to discern how this impacts the capabilities of feature extension combined with prelabeling.

For all the approaches, we observed that after a labeling budget of 200, the performance escalated substantially. For future research, we propose using our devised pipeline to determine the optimal labeling budget. As our Approach 1: Prelabeling necessitated no human intervention, this pipeline could be beneficial even for those looking to integrate active learning with human oversight. Our results indicated that a 200-labeling budget is potentially optimal. In future scenarios where an active learning approach is adopted without knowledge of the ideal labeling budget, establishing such a pipeline without human intervention and evaluating different budgets could assist in pinpointing the optimal budget to allocate to a human oracle. We do not assert that our findings provide definitive scientific evidence that this is the optimal range, but it does offer a fruitful avenue for future research, especially when access to a human oracle is limited.

In our discussion about automated retraining, we aim to explain why the optimal timing for retraining is observed during the double spike phenomenon. We postulate that the double spike emerges due to shifts in context. The hypothesis is that if a significant amount of data shifts due to the context change, the sliding windows near this shift will encompass data from both contexts. The first spike arises when the window contains a smaller fraction of the new context and a larger portion from the old context. In such a scenario, RTClean may get 'confused', initiating extensive cleaning, leading to the first spike. As the windows continue to move, we encounter a more balanced mix of old and new context data, causing RTClean to perceive the changes as reasonable, resulting in negligible cleaning and a subsequent drop in *fcv*-value. Yet, as the windows progress further, the influx of new context data and the presence of uncleaned old data again confound RTClean, leading to the second spike. As depicted in the combined version for all datasets (referenced as 6.20), this phenomenon was exclusive to the durable change, further suggesting that this does not manifest if the change is minimal. This is our rationale for associating the observed double spike phenomenon with data context changes. Future endeavors should delve deeper into this phenomenon to verify its validity as an indicator of context changes.

Lastly, we wish to evaluate our idea of automated retraining. Employing RTClean to discern the optimal timing for retraining emerged later in our research. We aimed to ascertain if RTClean could be a reliable detector for context shifts. Our preliminary findings are promising regarding determining the optimal retraining timing. However, a more comprehensive analysis is imperative to furnish scientific substantiation for this method, especially using a sliding window approach. Moreover, studying automated retraining should encompass an evaluation of vital data for retraining and the consistent presence of the double spike during significant context changes. Our research provides a robust foundation for future explorations into leveraging RTClean for automated retraining, yet we refrain from asserting that our outcomes serve as the conclusive scientific validation for its consistent feasibility.

This wraps up our evaluation chapter. We have provided a comprehensive review of our results, highlighting the potential of Approach 1: Prelabeling with improvements, and delved into implications and future research directions. In the subsequent and final chapter, we will conclude our thesis and forecast potential avenues for future endeavors.

7 Conclusion and Outlook

Throughout the course of this thesis, we focused on enhancing ML-based error detection methods through the use of RTClean. Recognizing the inherent challenges posed by the 'no free lunch theorem'—which postulates that there is not a universally superior model—we took an empirical approach to assessing our proposed classification methods, in line with established practices [BD87; Mur12]. First, we motivated the need for ML-based error detection algorithms to boost data quality in IoT-centric environments. Afterwards, we presented the necessary background information regarding data quality, context-aware computing, and ML with special focus on active learning. We continued with presenting related work about context-aware and learning based error detection. The central part of our work came after we completed the placement in the overall scientific context. Here, we first explained three approaches to using RTClean and Active Learning to develop an error detection model. These were based on a two-stage pipeline. We also presented how to extend the first stage to allow user intervention. After that, we presented an idea of how to use our pipeline to retrain automatic retraining of ML-based algorithms. Afterward, we showed the highlights of our implementation. We then evaluated our presented approaches, discussed their significance limits, and explained possible conclusions for the future.

Our primary contribution lies in the design of a two-stage pipeline. In the initial stage, RTClean is employed to pre-label data, identifying potential positive error instances. This not only fine-tunes the subsequent ML-based error detection process in the second stage but also serves the pragmatic purpose of conserving labeling budget. We further proposed three distinctive approaches that leverage this two-stage architecture. Each of these approaches can be further enhanced by an improvement step which uses user-defined thresholds.

Notably, the pre-labeling method, complemented by a user-set threshold improvement, showcased the most promising results in our evaluations. However, the practical implementation of this pipeline was not without challenges. These challenges, and our solutions to address them, form a critical part of our findings, underscoring the complexities of transitioning from theory to real-world application.

The analysis of our results showed that our Approach 1: Prelabeling with optimization, managed to outperform our baseline by 10 to 40 percent. However, this was only achieved with labeling budgets below 100 for the baseline. Furthermore, our analysis showed that the other approaches all remained below the baseline. In particular, Approach 3: Feature extension significantly underperformed. This shows how sensitive complex ML approaches, like ED2 in our case, are to adaptations. However, our evaluation showed two promising follow-up studies regarding automated retraining for ML models and optimal labeling budget for active learning approaches.

The methodologies and enhancements introduced in this thesis hold importance for the data preparation phase in the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework [MS17]. By addressing data errors more effectively, our techniques can impact the quality and trustworthiness of data as it is prepared for mining tasks. As organizations progressively turn to data-driven insights to guide their operations and strategies, our contributions offer a potent toolkit to elevate data quality through an automated error detection process, thus ensuring more reliable data mining outcomes.

Outlook

In our outlook, we aim to highlight potential areas for further research. To enhance our methodologies and ensure their effective application in real-world scenarios, future research must focus on refining RTClean's performance, particularly to improve the precision of preliminary labeling.

Moreover, the current version of RTClean requires enhancements in its flexibility, as it demands context information in a specific format. Investigating methods to automate the generation of context information, perhaps by leveraging advanced language models like ChatGPT or Google Bard, is an avenue worth exploring.

Our research has also brought to light two novel domains of RTCleans application: Its potential use for the automated retraining of ML-based error detectors in changing contexts. Its capability in determining the optimal labeling budget for active learning methodologies. While we've laid the groundwork in these areas, comprehensive scientific analysis is essential to validate the effectiveness and efficiency of these applications.

In relation to our approach which uses an ensemble it would be beneficial to evaluate the performance of the two-stage architecture when integrated with an ensemble of error cleaning methods alongside RTClean. A particular emphasis should be placed on methods with a high recall, addressing RTCleans challenge of low recall.

In closing, with the ascendance of data science and machine learning, we believe that the quest for robust error detection, and the development of effective models for the same, will present enduring challenges for the foreseeable future.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to Prof. Dr. rer. nat. habil. Holger Schwarz for affording me the invaluable opportunity to undertake my master's thesis at the Institute for Parallel and Distributed Systems.

I owe a significant debt of gratitude to Daniel Del Gaudio, whose pleasant supervision, motivating words, and unwavering support were instrumental throughout the course of this thesis. His keen eye, meticulous feedback during the proofreading process, and invaluable suggestions have shaped this work to its best form.

I also extend my thanks to Mohamed Abdelaal from Software AG. His input, drawing from the latest tools and concepts, was pivotal in the evaluation of my concept, and his commitment to excellence has been both instructive and inspiring.

I am incredibly blessed to have the unwavering love and support of my parents, Walid Salaheddine and Diana El-Kebbe. Their faith, encouragement, and lessons have been a bedrock of stability throughout this journey. My sister, Roufayda Salaheddine, deserves special mention for being a beacon of inspiration and support every step of the way. A heartfelt thank you goes to my close friends - Ruben Verma, Lukas Nabakowski, Barkha Sharma, and Eva Zimmermann. Your camaraderie, encouragement, and timely pep talks have been a source of comfort and strength during challenging times.

Bibliography

[ABW18] B. Alabdullah, N. Beloff, M. White. “Rise of Big Data – Issues and Challenges”. In: *2018 21st Saudi Computer Society National Computer Conference (NCC)*. 2018, pp. 1–6. DOI: 10.1109/NCG.2018.8593166 (cit. on p. 11).

[AHS23] M. Abdelaal, C. Hammacher, H. Schoening. “REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines”. In: *Proceedings of the VLDB Endowment (PVLDB)* (2023) (cit. on pp. 12, 25, 39, 69).

[AKG+14] C. Aggarwal, X. Kong, Q. Gu, J. Han, P. Yu. “Active learning: A survey”. English (US). In: *Data Classification*. Publisher Copyright: © 2015 by Taylor & Francis Group, LLC. CRC Press, Jan. 2014, pp. 571–605. ISBN: 9781466586741. DOI: 10.1201/b17320 (cit. on p. 12).

[AKG21] B. B. andNandhini Abirami R, S. Kadry, A. H. Gandomi. *Big Data: Concepts, Technology, and Architecture*. Wiley, 2021 (cit. on p. 11).

[ASW05] N. Allen, C. Shaffer, L. Watson. “Building modeling tools that support verification, validation, and testing for the domain expert”. In: *Proceedings of the Winter Simulation Conference, 2005.* 2005, 8 pp.-. DOI: 10.1109/WSC.2005.1574277 (cit. on p. 12).

[BCC20] M. Bansal, I. Chana, S. Clarke. “A Survey on IoT Big Data: Current Status, 13 V’s Challenges, and Future Directions”. In: *ACM Comput. Surv.* 53.6 (Dec. 2020). ISSN: 0360-0300. DOI: 10.1145/3419634. URL: https://doi.org/10.1145/3419634 (cit. on pp. 11, 18).

[BD87] G. Box, N. Draper. *Empirical Model-Building and Response Surfaces*. Wiley Series in Probability and Statistics. Wiley, 1987. ISBN: 9780471810339. URL: https://books.google.de/books?id=QO2dDRufJEAC (cit. on p. 73).

[BKC+17] S. Baskaran, A. Keller, F. Chiang, L. Golab, J. Szlichta. “Efficient Discovery of Ontology Functional Dependencies”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM ’17. Singapore, Singapore: Association for Computing Machinery, 2017, pp. 1847–1856. ISBN: 9781450349185. DOI: 10.1145/3132847.3132879. URL: https://doi.org/10.1145/3132847.3132879 (cit. on p. 12).

[CIKW16] X. Chu, I. F. Ilyas, S. Krishnan, J. Wang. “Data Cleaning: Overview and Emerging Challenges”. In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 2201–2206. ISBN: 9781450335317. DOI: 10.1145/2882903.2912574. URL: https://doi.org/10.1145/2882903.2912574 (cit. on p. 25).

[CMI+15] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, Y. Ye. "KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD '15. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 1247–1261. ISBN: 9781450327589. DOI: 10.1145/2723372.2749431. URL: https://doi.org/10.1145/2723372.2749431 (cit. on p. 25).

[CNC+18] T. Chakraborty, A. U. Nambi, R. Chandra, R. Sharma, M. Swaminathan, Z. Kapetanovic, J. Appavoo. "Fall-Curve: A Novel Primitive for IoT Fault Detection and Isolation". In: *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. SenSys '18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 95–107. ISBN: 9781450359528. DOI: 10.1145/3274783.3274853. URL: https://doi.org/10.1145/3274783.3274853 (cit. on p. 18).

[CPIR14] J. R. Cardoso, L. M. Pereira, M. D. Iversen, A. L. Ramos. "What is gold standard and what is ground truth?" In: *Dental press journal of orthodontics* 19 (2014), pp. 27–30 (cit. on p. 17).

[DEE+13] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, N. Tang. "NADEEF: A Commodity Data Cleaning System". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD '13. New York, New York, USA: Association for Computing Machinery, 2013, pp. 541–552. ISBN: 9781450320375. DOI: 10.1145/2463676.2465327. URL: https://doi.org/10.1145/2463676.2465327 (cit. on p. 25).

[DSA23] D. Del Gaudio, T. Schubert, M. Abdelaal. "RTClean: Context-aware Tabular Data Cleaning using Real-time OFDs". In: *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2023, pp. 546–551. DOI: 10.1109/PerComWorkshops56833.2023.10150361 (cit. on pp. 12, 19–21, 47, 48).

[GABS22] D. D. Gaudio, B. Ariguib, A. Bartenbach, G. Solakis. "A live context model for semantic reasoning in IoT applications". In: *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2022, pp. 322–327. DOI: 10.1109/PerComWorkshops53856.2022.9767267 (cit. on pp. 12, 20, 21).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 12).

[GFS+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C.-a. Saita. "Declarative Data Cleaning: Language, Model, and Algorithms". In: *VLDB* (July 2001) (cit. on p. 17).

[GSDH17] Z. Ge, Z. Song, S. X. Ding, B. Huang. "Data Mining and Analytics in the Process Industry: The Role of Machine Learning". In: *IEEE Access* 5 (2017), pp. 20590–20616. DOI: 10.1109/ACCESS.2017.2756872 (cit. on p. 11).

[Ham13] K. Ham. "OpenRefine (version 2.5). http://openrefine. org. Free, open-source tool for cleaning and transforming data". In: *Journal of the Medical Library Association: JMLA* 101.3 (2013), p. 233 (cit. on p. 26).

[KS22] P. Kumar, M. Sharma. “Data, Machine Learning, and Human Domain Experts: None Is Better than Their Collaboration”. In: *International Journal of Human–Computer Interaction* 38.14 (2022), pp. 1307–1320. DOI: 10.1080/10447318.2021.2002040. URL: https://doi.org/10.1080/10447318.2021.2002040 (cit. on p. 12).

[LGEC17] A. L’Heureux, K. Grolinger, H. F. Elyamany, M. A. M. Capretz. “Machine Learning With Big Data: Challenges and Approaches”. In: *IEEE Access* 5 (2017), pp. 7776–7797. DOI: 10.1109/ACCESS.2017.2696365 (cit. on p. 11).

[LRB+19] P. Li, S. X. Rao, J. Blase, Y. Zhang, X. Chu, C. Zhang. “CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]”. In: *ArXiv* abs/1904.09483 (2019). URL: https://api.semanticscholar.org/CorpusID:128316992 (cit. on p. 26).

[LTZ12] F. T. Liu, K. Ting, Z.-H. Zhou. “Isolation-Based Anomaly Detection”. In: *ACM Transactions on Knowledge Discovery From Data - TKDD* 6 (Mar. 2012), pp. 1–39. DOI: 10.1145/2133360.2133363 (cit. on p. 26).

[LZR21] Z. Liu, Z. Zhou, T. Rekatsinas. “Picket: Guarding against Corrupted Data in Tabular Data during Learning and Inference”. In: *The VLDB Journal* 31.5 (Oct. 2021), pp. 927–955. ISSN: 1066-8888. DOI: 10.1007/s00778-021-00699-w. URL: https://doi.org/10.1007/s00778-021-00699-w (cit. on p. 27).

[MAC+19] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, N. Tang. “Raha: A Configuration-Free Error Detection System”. In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD ’19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 865–882. ISBN: 9781450356435. DOI: 10.1145/3299869.3324956. URL: https://doi.org/10.1145/3299869.3324956 (cit. on p. 26).

[MCSG21] S. Moon, S. Chatterjee, P. H. Seeberger, K. Gilmore. “Predicting glycosylation stereoselectivity using machine learning”. In: *Chem. Sci.* 12 (8 2021), pp. 2931–2939. DOI: 10.1039/D0SC06222G. URL: http://dx.doi.org/10.1039/D0SC06222G (cit. on p. 12).

[MS17] K. McCormick, J. Salcedo. *IBM SPSS Modeler essentials: Effective techniques for building powerful data mining and predictive analytics solutions*. Packt Publishing Ltd, 2017 (cit. on p. 73).

[Mur12] K. P. Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012 (cit. on p. 73).

[NKR20] M. Nourani, J. King, E. Ragan. “The role of domain expertise in user trust and the impact of first impressions with intelligent systems”. In: *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*. Vol. 8. 1. 2020, pp. 112–121 (cit. on p. 12).

[NMA19] F. Neutatz, M. Mahdavi, Z. Abedjan. “ED2: A Case for Active Learning in Error Detection”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM ’19. Beijing, China: Association for Computing Machinery, 2019, pp. 2249–2252. ISBN: 9781450369763. DOI: 10.1145/3357384.3358129. URL: https://doi.org/10.1145/3357384.3358129 (cit. on pp. 12, 27, 39, 47, 48).

[NRP+18] A. Nikoukar, S. Raza, A. Poole, M. Günes, B. Dezfouli. "Low-Power Wireless for the Internet of Things: Standards and Applications". In: *IEEE Access* PP (Nov. 2018), pp. 1–1. DOI: 10.1109/ACCESS.2018.2879189 (cit. on p. 11).

[ORH05] P. Oliveira, F. Rodrigues, P. R. Henriques. "A formal definition of data quality problems." In: *ICIQ*. 2005 (cit. on p. 17).

[PK16] K. S. Priya, Y. Kalpana. "A review on context modeling techniques in context aware computing". In: *Int. J. Eng. Technol* 8.1 (2016), pp. 429–433 (cit. on p. 19).

[PMHM16] C. Pit–Claudel, Z. Mariet, R. Harding, S. Madden. "Outlier Detection in Heterogeneous Datasets using Automatic Tuple Expansion". In: (Feb. 2016) (cit. on p. 26).

[RCIR17] T. Rekatsinas, X. Chu, I. F. Ilyas, C. Ré. "HoloClean: Holistic Data Repairs with Probabilistic Inference". In: *Proc. VLDB Endow.* 10.11 (Aug. 2017), pp. 1190–1201. ISSN: 2150-8097. DOI: 10.14778/3137628.3137631. URL: https://doi.org/10.14778/3137628.3137631 (cit. on p. 25).

[SAW94] B. Schilit, N. Adams, R. Want. "Context-Aware Computing Applications". In: *1994 First Workshop on Mobile Computing Systems and Applications*. 1994, pp. 85–90. DOI: 10.1109/WMCSA.1994.16 (cit. on pp. 12, 19).

[Sch22] T. Schubert. "Context-aware data validation for machine learning pipelines". In: (2022) (cit. on pp. 11, 18, 21, 51, 52).

[Set22] B. Settles. *Active Learning*. Springer, 2022 (cit. on pp. 12, 22–24).

[SS13] S. Sagiroglu, D. Sinanc. "Big data: A review". In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*. 2013, pp. 42–47. DOI: 10.1109/CTS.2013.6567202 (cit. on p. 11).

[SSA+12] F. Sidi, P. H. Shariat Panahy, L. S. Affendey, M. A. Jabar, H. Ibrahim, A. Mustapha. "Data quality: A survey of data quality dimensions". In: *2012 International Conference on Information Retrieval and Knowledge Management*. 2012, pp. 300–304. DOI: 10.1109/InfRKM.2012.6204995 (cit. on p. 17).

[WCS+20] R. Wu, S. Chaba, S. Sawlani, X. Chu, S. Thirumuruganathan. "ZeroER: Entity Resolution Using Zero Labeled Examples". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD '20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 1149–1164. ISBN: 9781450367356. DOI: 10.1145/3318464.3389743. URL: https://doi.org/10.1145/3318464.3389743 (cit. on p. 26).

[Zha13] J. Zhang. "Advancements of Outlier Detection: A Survey". In: *EAI Endorsed Transactions on Scalable Information Systems* 1.1 (Feb. 2013). DOI: 10.4108/trans.sis.2013.01-03.e2 (cit. on p. 26).

[Zho22] Z.-H. Zhou. "Open-environment machine learning". In: *National Science Review* 9.8 (July 2022), nwac123. ISSN: 2095-5138. DOI: 10.1093/nsr/nwac123. eprint: https://academic.oup.com/nsr/article-pdf/9/8/nwac123/45957092/nwac123.pdf. URL: https://doi.org/10.1093/nsr/nwac123 (cit. on p. 12).

All links were last followed on October, 2023.

Declaration