TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik

CSR-26-01

Cloud-Based AI Solutions for Ensuring Data Quality in Predictive Models

Vismay Gunda · Shadi Saleh · Wolfram Hardt

Februar 2026

Chemnitzer Informatik-Berichte

Cloud-Based AI Solutions for Ensuring Data Quality in Predictive Models

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Vismay Gunda
Student ID: 756310
Date: 04.12.2025

Supervising tutor: Prof. Dr. W. Hardt
Prof. Dr. Shadi Saleh

Abstract

This thesis addresses the challenge of improving the quality of heterogeneous sensor data by following a data-centric rather than a model-centric approach. Instead of assuming standardized inputs for downstream AI models, the work focuses on making data itself reliable and analysis-ready across diverse SmartCityCloud (SCC) sources. The proposed solution combines exploratory data analysis (EDA) with a suite of data-quality measures to assess and enhance credibility across multiple variables and datasets. The methodology includes automated profiling, duplicate removal, validity checks, imputation, feature engineering, and anomaly detection, together with out-of-distribution (OOD) generalization checks using configurable splits (e.g., 70/30 and 60/40), augmentation (noise/missingness) for stress-testing, and labeling strategies (e.g., day/night separation).

The implementation delivers a user-friendly, cloud-based platform within SCC. Users can upload datasets, run EDA, visualize time series, distributions, correlations, and boxplots, and export figures and tables (e.g., PNG/PDF for plots; CSV/JSON for data and reports). The system generates a machine-readable JSON report that is then evaluated by six practical metrics: Accuracy, Completeness, Consistency, Traceability, Timeliness, and Auditability.

Results from multiple SCC datasets indicate that the pipeline improves data readiness (e.g., fewer duplicates and invalid readings, clearer trends, and more consistent labels) while providing transparent artifacts for review. The thesis contributes (i) a reusable data-centric workflow for variable sensor data, (ii) a reference implementation as an SCC compute-task template that users can adapt, and (iii) an evaluable reporting scheme that supports dependable AI development on city-scale data.

Keywords: Data-Centric AI, Data Quality, Exploratory Data Analysis (EDA), SmartCityCloud, Data Augmentation.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
AQI	Air Quality Index
API	Application Programming Interface
App	Application
CSV	Comma-Separated Values
CTE	Compute Task Executor (SmartCityCloud UI)
DAQ	Data Acquisition
DC-AI	Data-Centric Artificial Intelligence
EDA	Exploratory Data Analysis
FE	Feature Engineering
GUI	Graphical User Interface
HQ	High Quality (Processed Dataset)
IQR	Interquartile Range
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ML	Machine Learning
MLOps	Machine Learning Operations
NaN	Not a Number (missing value placeholder)
OOD	Out-of-Distribution
QR	Quick Response (UI buttons for select/clear)
SCC	Smart City Cloud
SRAD	Solar Radiation
Std	Standard Deviation
3σ	Three-Sigma Statistical Range
ATF	Air Temperature Feature (in dataset folder naming)
FFill	Forward Fill (interpolation method)
BFill	Backward Fill (interpolation method)
ME	Month-End Resampling Frequency (pandas)
Z-Score	Standard Score for Outlier Detection

1 Introduction

Modern artificial intelligence (AI) systems depend not only on sophisticated models but also on the reliability of the data used to train, validate, and deploy them [1]. In smart city environments, where data originates from heterogeneous sensors, gateways, and services, datasets are often incomplete, noisy, poorly labeled, or statistically inconsistent over time [2]. Such variability affects the accuracy, reproducibility, and interpretability of predictive models. This thesis addresses these challenges through a data-centric approach that treats data quality as the primary focus of engineering. Instead of optimizing only model architectures, the work concentrates on preparing multi-source time series data to be reliable, representative, and analysis-ready through systematic exploration, assessment, and improvement of data quality.

The research is conducted within the SmartCityCloud(SCC) environment, a cloud-based infrastructure designed for processing and visualizing sensor data in smart city applications. Within this framework, the thesis develops and implements an integrated pipeline that ingests diverse sensor streams, performs exploratory data analysis (EDA) to characterize distributions, trends, correlations, seasonality, and outliers, and applies data-quality operations such as duplicate removal, validity screening, missing-value imputation, and feature engineering. Because labeling accuracy strongly influences downstream analytics [2], the system includes label-quality verification procedures, for example, day and night differentiation to capture diurnal variations and validate labeling consistency with temporal data characteristics.

Beyond descriptive profiling, the proposed approach evaluates robustness through out-of-distribution (OOD) generalization tests using configurable train and test splits, such as 70/30 or 60/40, to identify distribution shifts across different time windows, sites, or operational contexts. To further assess the sensitivity of data-quality methods, controlled augmentation techniques introduce synthetic noise or missing values, allowing the system to examine how these perturbations affect quality metrics and data stability. The platform enables users to perform these analyses interactively, visualize intermediate results such as tables, bar charts, boxplots, and time overlays, and export outcomes in multiple formats, including PNG, PDF, and CSV. The complete data-quality summary is stored in a structured JSON report that facilitates reproducibility and interoperability within the SmartCityCloud ecosystem.

The concept of data quality is operationalized in this thesis as a measurable property rather than a qualitative assumption. To this end, six key evaluation dimensions—

accuracy, completeness, consistency, traceability, timeliness, and auditability—are used to assess and compare results across datasets and configurations. Each dimension guides both the quantitative reporting and the qualitative interpretation of the improvements achieved by data-quality operations. Emphasis is placed on transparency and explainability; every transformation step is accompanied by a corresponding visualization, statistical summary, and metadata entry that support traceable and auditable processing. Such transparency is essential for dependable AI applications in safety-critical or regulated domains.

The contributions of this thesis are threefold. First, it introduces a reusable data-centric workflow for heterogeneous smart city time-series data that integrates exploratory data analysis, label verification, out-of-distribution analysis, and augmentation-based validation. Second, it provides a user-friendly cloud-based implementation within SmartCityCloud that enables users to execute these processes interactively through a configurable compute-task interface. Third, it defines a data-quality evaluation framework aligned with the six aforementioned dimensions, demonstrating measurable improvements in data readiness for AI models without focusing on specific model architectures. Together, these contributions highlight how systematic data-centric engineering enhances the robustness, interpretability, and dependability of AI-driven systems.

The remainder of this thesis is organized as follows. Chapter 2 introduces the theoretical and technical fundamentals. Chapter 3 presents related research and the conceptual background for data-centric artificial intelligence. Chapter 4 describes the methodology, including the exploratory data analysis process, data-quality evaluation methods, labeling procedures, and out-of-distribution generalization strategies. Chapter 5 details the system implementation within SmartCityCloud and its user interface. Chapter 6 presents the results and evaluation of data-quality improvements. Chapter 7 discusses the implications, limitations, and potential extensions of the work. Finally, Chapter 8 concludes the thesis and outlines future research directions.

1.1 Background and context

In today's artificial intelligence pipelines, data has emerged as the most important asset. In operational contexts like smart cities, data comes from a variety of sources, including ambient sensors, edge devices, cloud services, and human-operated systems. These sources generate multivariate time series with varying sampling rates, missing segments, duplicated entries, and value range shifts caused by seasonality, maintenance, and deployment changes. Such fluctuation calls into question the

assumptions made while developing and evaluating downstream models. If not addressed, it diminishes model dependability, complicates replication, and obscures root-cause investigation when systems fail in production. As a result, the practical bottleneck is less about choosing a sophisticated model and more about collecting reliable, auditable data that accurately reflect the phenomena of interest for learning and inference.

Within this context, SmartCityCloud (SCC) acts as the operational framework for the current effort. SCC provides a cloud-based environment where users may upload datasets, set up processing stages, and view results using a reusable compute-task template. The template is designed to be adaptable to various sensor modalities and projects while maintaining a consistent workflow for ingestion, investigation, transformation, and reporting. Because smart-city installations change over time and between locations, the platform prioritizes repeatable workflows and exportable artifacts, allowing analyses to be re-run, compared, and approved by various stakeholders without ambiguity.

The data landscape discussed here has three repeating characteristics. First, heterogeneity exists: streams differ in units, valid ranges, and semantics, even for seemingly equivalent variables (for example, pressure or temperature from various manufacturers). Second, non-stationarity: distributions change due to weather, urban activity cycles, firmware updates, and sensor aging, invalidating static training-testing divides and making model-centric comparisons incorrect. Third, label fragility: labels based on heuristics or external schedules (e.g., day-night, event windows) may be misaligned with real sensor behavior, resulting in inconsistent supervision and misleading correlations. These characteristics encourage a data-centric approach in which data quality is actively profiled, improved, and recorded before to and throughout modeling.

As a result, the thesis employs exploratory data analysis (EDA) as a primary component for characterizing empirical distributions, trends, correlations, and outliers across variables and time periods [3]. Visual diagnostics like time overlays, histograms, and boxplots are supplemented by basic statistical summaries and rolling descriptions that highlight gaps, spikes, and regime shifts. On top of this descriptive layer, focused quality operations are used to detect duplicates, screen ranges and validity, identify gaps with imputation choices, and create features for downstream processes. Label quality is given special attention: criteria such as day/night partitioning are aligned with

time bases and tested for internal consistency to limit the spread of mislabelled segments.

Because smart-city data must function in the face of change, background analysis takes into account distribution shift robustness. The study used out-of-distribution (OOD) checks using adjustable temporal or contextual divides (for example, 70/30 or 60/40 partitions across times or places) to identify when statistics and label behaviour diverge in held-out slices. Furthermore, augmentation serves as a stress test: controlled missingness and noise enable sensitivity analysis of processes such as imputation or outlier treatment [4]. These methods do not replace modelling; rather, they provide solid, well-defined inputs and recorded assumptions, ensuring that any subsequent model evaluation reflects the realities of the data rather than artifacts.

Operational restrictions in municipal and industrial environments exacerbate the demand for traceability. Stakeholders often expect not only findings but also a chain of proof that connects each transformation to its purpose and impact. To match this expectation, the platform generates exportable figures (PNG/PDF), tabular outputs (CSV), and a structured JSON record that includes configuration, intermediate results, and final quality indicators. This approach promotes auditability across teams and throughout time, while lowering onboarding costs when datasets, persons, or objectives change.

Finally, the background for this thesis is consistent with the formal requirements for a Master's Thesis at the Professorship of Computer Engineering, TU Chemnitz. The introduction must locate the topic in its application area, describe motivation, and define the problem at a high level, all while preparing the reader for subsequent chapters that cover fundamentals, related work, methodology, implementation, findings, discussion, and conclusion. The emphasis on clear context, neutral academic tone, and structured reporting adheres to the department's chapter organization and scientific writing guidelines, ensuring that succeeding sections can be built on a coherent and suitably thorough foundation.

1.2 Motivation

Smart-city analytics are based on multivariate sensor streams that are noisy, partial, heterogeneous, and subject to drift. In such cases, model-centric development implies standardized inputs that are rarely available in practice, resulting in brittle systems and findings that are difficult to replicate or trust [5]. The basic goal for this thesis is to

transfer the focus of development from models to data: to make data more trustworthy[6], interpretable, and auditable before—and alongside—modelling. Framing motivation early and clearly is consistent with the department's recommendations for the introduction chapter, as well as the expectation that goals be articulated concretely and verifiably.

Operational requirements further support a data-centric strategy. Municipal and industrial stakeholders must track how each preprocessing step impacts downstream use, compare outcomes over time and between locations, and justify actions during audits or handovers [7]. A cloud environment, such as SmartCityCloud, provides a consistent location to upload datasets, examine them using EDA, apply quality metrics, and collect artifacts—plots, tables, and machine-readable summaries—to make the process transparent. By incorporating repeatability and proof (exports and JSON records), the platform facilitates cooperation and long-term maintenance, addressing common issues in evolving installations where sensors, firmware, and usage patterns vary.

Finally, measurable outcomes are required to steer improvements and convey value. This study uses six practical dimensions—accuracy, completeness, consistency, traceability, timeliness, and auditability—to assess how specific operations (duplicate handling, validity checks, imputation, labeling rules, out-of-distribution splits, and augmentation-based stress tests) enhance data readiness [8]. Articulating such objectives as explicit, measurable aims is consistent with the preferred motivation style (clear purpose, evaluability, and time-bound execution within the thesis timeframe) and creates a cohesive bridge from introduction to technique, implementation, and results.

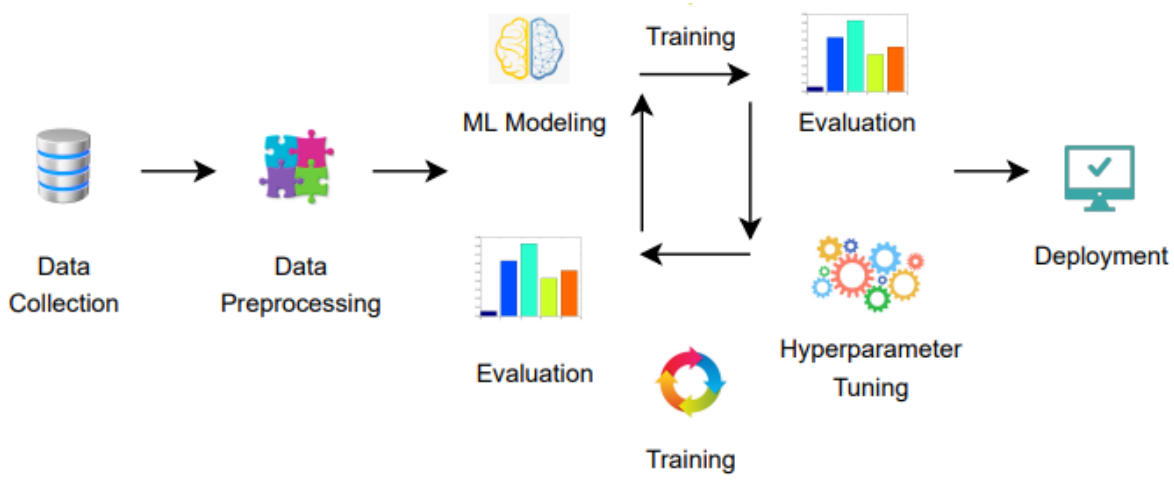


Figure 1. Model-Centric Approach [9]

Fig 1. Model-centric approach [9].

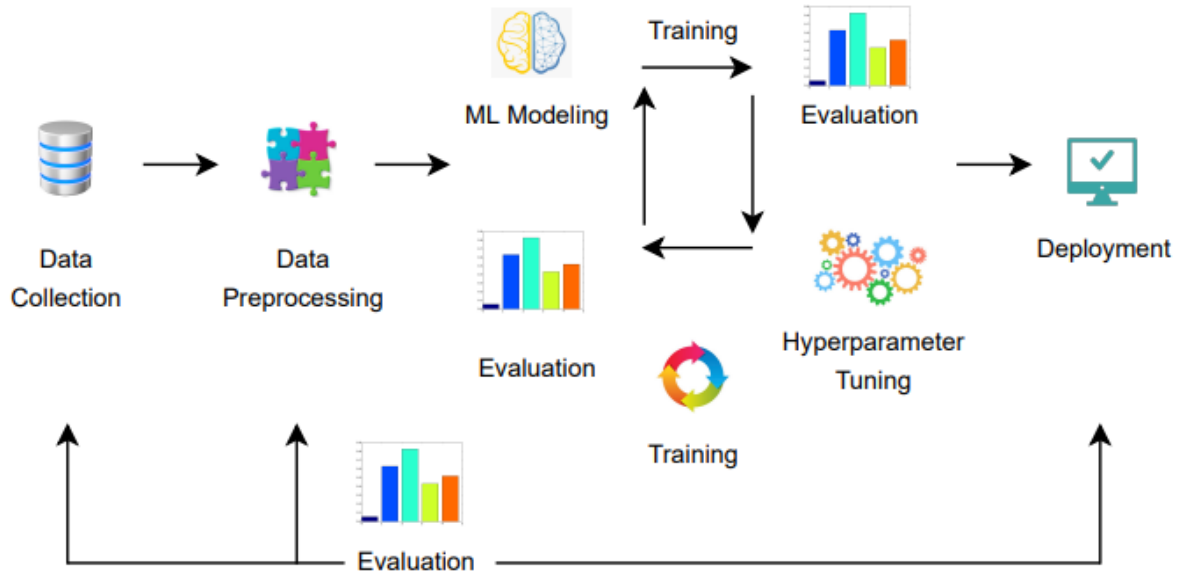


Figure 2. Data-Centric Approach [9]

In Fig 1 (model-centric AI), most iterative effort concentrates on the modelling loop: training, evaluation, and hyperparameter tuning cycle repeatedly, while data preprocessing is treated as a largely one-off step before the loop begins. This workflow assumes the dataset is already standardized and sufficiently representative; quality issues are addressed ad hoc, if at all, and evaluation focuses on comparing models rather than interrogating the data that drive them. As a result, gains typically come from architectural choices or tuning, and failure modes often trace back to silent data problems—label noise, drift, duplicates, or gaps—that the loop is not designed to surface.

However, in Fig 2 (data-centric AI), evaluation expands to include the data itself. The modelling loop remains, but a parallel feedback loop targets dataset curation: systematic EDA, labelling checks, augmentation for stress testing, and iterative remediation of errors become first-class activities. This shifts improvement leverage toward making signals clearer and assumptions explicit, yielding models that are simpler to train, easier to reproduce, and more stable under deployment shifts. In practice, the data-centric loop produces auditables—figures, tables, and machine-readable summaries—that document how changes in data quality translate into performance changes, enabling controlled, evidence-based progress.

1.3 Problem Statement

SmartCityCloud aggregates heterogeneous, multi-source time series with missing segments, duplicates, outliers, non-stationary distributions, and fragile or heuristic labels. Conventional model-centric pipelines presume standardized inputs and regard preprocessing as a one-time operation, which obscures the underlying causes of downstream brittleness and makes outcomes difficult to replicate or audit. The central problem addressed in this thesis is to make data quality the primary engineering objective: to systematically surface, measure, and improve the credibility of diverse sensor datasets before—and concurrently with—modelling, while maintaining a transparent record of how each transformation affects the data.

Existing literature and tools provide useful components—profilers, cleaning utilities, experiment trackers [10]—but they rarely provide an integrated, cloud-based workflow that unifies exploratory data analysis, targeted quality operations (e.g., duplicate handling, validity screening, gap detection and imputation, feature engineering), label-quality verification, out-of-distribution checks, and augmentation-based stress testing, all linked to machine-readable provenance. In practice, teams rely on ad hoc scripts[11] and informal notebooks, limiting comparability across time, places, and users and making audits time-consuming. This thesis addresses that gap by creating a reusable data-centric pipeline within SmartCityCloud that combines interactive visualization and exporting (PNG/PDF/CSV) with a structured JSON report containing configurations, interim findings, and outcomes mapped to pragmatic quality metrics.

The thesis is guided by the research questions listed below.

- RQ1: How to design SmartCityCloud compute task wrapper and implement a reusable data-centric workflow to improve the readiness of heterogeneous smart-city time-series data through systematic EDA, quality operations, label verification, out-of-distribution analysis, and augmentation-based stress tests, all while producing auditable provenance?
- RQ2: How much do these interventions improve data quality—as assessed by accuracy, completeness, consistency, traceability, timeliness, and auditability—and boost stability under distribution shift when compared to baseline preprocessing typical of model-centric practice?
-

1.4 Objectives and Research Goals

This thesis explores a data-centric alternative to model-first development for smart-city analytics. The overarching goal is to make heterogeneous sensor datasets analysis

ready, auditable, and resilient to distribution shifts by prioritizing data quality [2]. The goals are articulated in accordance with the standards for a clear, measurable motivation and objectives, allowing them to be evaluated within the thesis scope and schedule.

Primary objectives of this thesis include:

- First, create and implement an end-to-end workflow in SmartCityCloud that allows users to upload datasets, conduct exploratory data analysis, perform targeted data-quality operations (duplicate handling, validity checks, gap detection and imputation, feature engineering, and label verification), and visualize the results.
- Second, implement robustness checks using out-of-distribution protocols (e.g., temporal/site-based 70/30 splits) and augmentation-based stress tests (controlled missingness and noise). Third, establish complete traceability by exporting artifacts (PNG/PDF plots and CSV tables) and creating a machine-readable JSON report that includes configurations, intermediate results, and final quality indicators for each run.

Evaluation Goals of this thesis include:

- Quantify the workflow's impact on data readiness across numerous SmartCityCloud datasets using six practical dimensions: correctness, completeness, consistency, traceability, timeliness, and auditability.
- Compare the results to a baseline representing model-centric preprocessing (minimum cleaning plus direct modelling assumptions). Success will be demonstrated by systematic gains across all dimensions, clearer and more reliable descriptive statistics, and verifiable provenance that connects each alteration to its measured impact.

Secondary Objectives of this thesis include:

- Improving the platform's usability and maintainability by including an extensible compute-task template, clear user advice inside the interface, and defaults that encourage recurring analysis.
- Where possible, define the runtime and scalability of essential procedures (such as profiling and imputation) to ensure their suitability for city-scale operations.
- Expected contributions include:
    (i) A reusable data-centric workflow for heterogeneous smart-city time series

(ii) A cloud-based implementation in SmartCityCloud that generates verifiable artifacts and provenance

(iii) An evaluation scheme that links concrete quality interventions to measurable gains, thereby promoting dependable and transparent AI development.

1.5 Scope and Limitations

This thesis focuses on improving data quality for heterogeneous smart-city time-series within the SmartCityCloud environment. The scope covers dataset ingestion, exploratory data analysis, and key quality operations such as duplicate handling, validity screening, gap detection with imputation, and feature engineering. It also includes label verification (e.g., day–night alignment), robustness checks using out-of-distribution splits, and augmentation-based stress tests. All outcomes are documented through exportable artifacts (PNG, PDF, CSV) and a machine-readable JSON provenance record. Evaluation is based on six process-oriented dimensions—accuracy, completeness, consistency, traceability, timeliness, and auditability—making data readiness measurable and verifiable within the thesis scope. The structure and writing style follow the department's scientific thesis guidelines, ensuring clarity, consistency, and alignment with the overall research framework.

The work does not aim to advance model architectures, large-scale hyperparameter optimization [12], or state-of-the-art benchmark contests; any modeling references serve only to contextualize data-quality effects. Topics outside the scope include production MLOps hardening (e.g., autoscaling, CI/CD), privacy/legal compliance, ethics reviews, and real-time latency guarantee. Limitations arise from dataset availability and representativeness, potential imperfections in heuristic labels, the bounded set of quality metrics (which may not capture every domain-specific notion of "quality"), and the specific OOD and augmentation scenarios considered (primarily temporal or site-based shifts with simple missingness/noise models) [13]. Results should therefore be interpreted as evidence of process improvements and reproducible provenance within SmartCityCloud, rather than as universal guarantees across all sensor modalities or deployment contexts. These delimitations align with the thesis requirement to define a clear, feasible scope and to maintain a neutral, structured academic presentation.

1.6 Thesis Structure

The rest of this thesis is organized as follows. Chapter 2 covers the theoretical and technical foundations needed for data-centric quality engineering and SCC. Chapter 3 examines related work on exploratory data analysis, data-quality metrics, labelling techniques, out-of-distribution analysis, and augmentation, situating the contribution within the status of the field. Chapter 4 describes the methodology, which includes procedures for EDA, targeted quality activities, label verification, robustness tests, and reporting. The fifth chapter details the SmartCityCloud implementation, which includes system components, a user interface, and data flows. Chapter 6 presents the results and evaluations for the six quality dimensions. Chapter 7 explores the consequences, limitations, and parallels to typical model-centric practice. Chapter 8 summarizes the findings and outlines future research and platform extensions Fig 3.

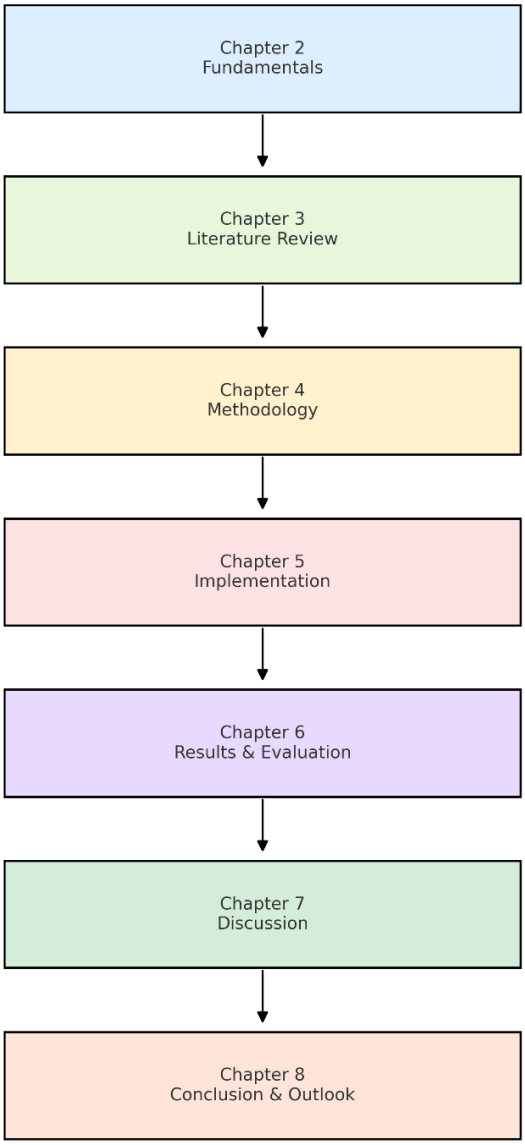


Figure 3. Thesis Structure Overview

2 Fundamentals

This chapter covers the theoretical principles that underpin the research and implementation in this thesis. It covers the fundamental concepts and principles needed to comprehend how cloud-based infrastructures, data-centric artificial intelligence, and data quality engineering interact in smart-city settings. The emphasis is on explaining the fundamental background required to understand the methodological and technical decisions detailed in subsequent chapters. The section discusses the fundamentals of cloud computing and data management, the theoretical foundations of data-centric AI, the key dimensions used to assess data quality, and the data governance and reproducibility in AI-powered systems.

2.1 SmartCityCloud and the Compute Task wrapper

SmartCityCloud (SCC) is a modular platform designed for managing heterogeneous smart-city sensor streams—traffic, forestry, air quality, parking, drones, and related domains—supporting both real-time and batch analytics across an ingestion–storage–compute pipeline. Its reference architecture separates concerns into four cooperating layers: a User Interface layer for interaction and visualization; a Compute Task layer encapsulating the core data-processing logic; a Data Streams layer that normalizes time-series, images, and tabular values into typed streams; and a Data Storage layer that handles input acquisition and output persistence. In a typical workflow, the backend ingests or reads data, a compute engine on a GPU server executes the user-defined task, and results are returned for visualization and export (schedule → download → compute → upload). Within this ecosystem, SCC offers a template and lab workflow to set up environments (local or Docker), bind inputs, configure options, and surface results through an auto-generated browser UI—providing a consistent integration point for domain-specific analytics like the data-quality engineering carried out in this thesis.

The SmartCity Compute Task Wrapper is the central execution framework within SmartCityCloud that standardizes how analytical processes are defined, executed, and visualized in a cloud environment. It manages the complete workflow—from loading and harmonizing input data streams to configuring analytical parameters, executing data-processing operations, and generating interpretable outputs—ensuring modularity, scalability, and reproducibility across diverse smart-city datasets. Abstracting low-level data handling and interface logic, it enables a seamless integration of domain-specific analytics such as exploratory data analysis, data

validation, feature extraction, label quality verification, and out-of-distribution stability checks. The wrapper automatically converts analytical outputs into structured tables, plots, and JSON reports, which can be visualized or exported directly through the SmartCityCloud interface. This design transforms the platform into a Data Quality as a Service (DQaaS) system, providing transparent, version-controlled, and auditable data processing that aligns with data-centric AI principles and supports high-quality, trustworthy predictive modeling for smart-city applications.

The SmartCityCloud Compute Task Wrapper, as shown below in Fig 4, is the key architectural workflow that allows for modular, task-based data processing within the SmartCityCloud ecosystem. It allows for smooth interaction between the data ingestion, computing, and visualization levels via a standardized pipeline. Raw sensor data—whether tabular, image-based, or time-series—is first collected in the storage layer and then accessible via an input reader, which turns the sources into standardized data streams. These streams are then routed to the job implementation module, where specialized analytical reasoning is used. The task implementation component of this thesis includes all of the code created for data quality engineering, exploratory data analysis, validity screening, feature extraction, label quality evaluation, and out-of-distribution generalization. After processing, the findings are transmitted to the output reader, which returns cleaned and processed data in the form of Python-native structures or reusable data streams. The wrapper combines visualization components for both input and output, as well as adjustable compute options available via a web interface, allowing users to enter parameters and evaluate results interactively. This design makes the framework completely extensible—any researcher or developer can incorporate their own analytical logic into the task implementation block to perform domain-specific operations, transforming the SmartCityCloud Compute Task Wrapper into a versatile and reusable foundation for scalable cloud-based data analytics.

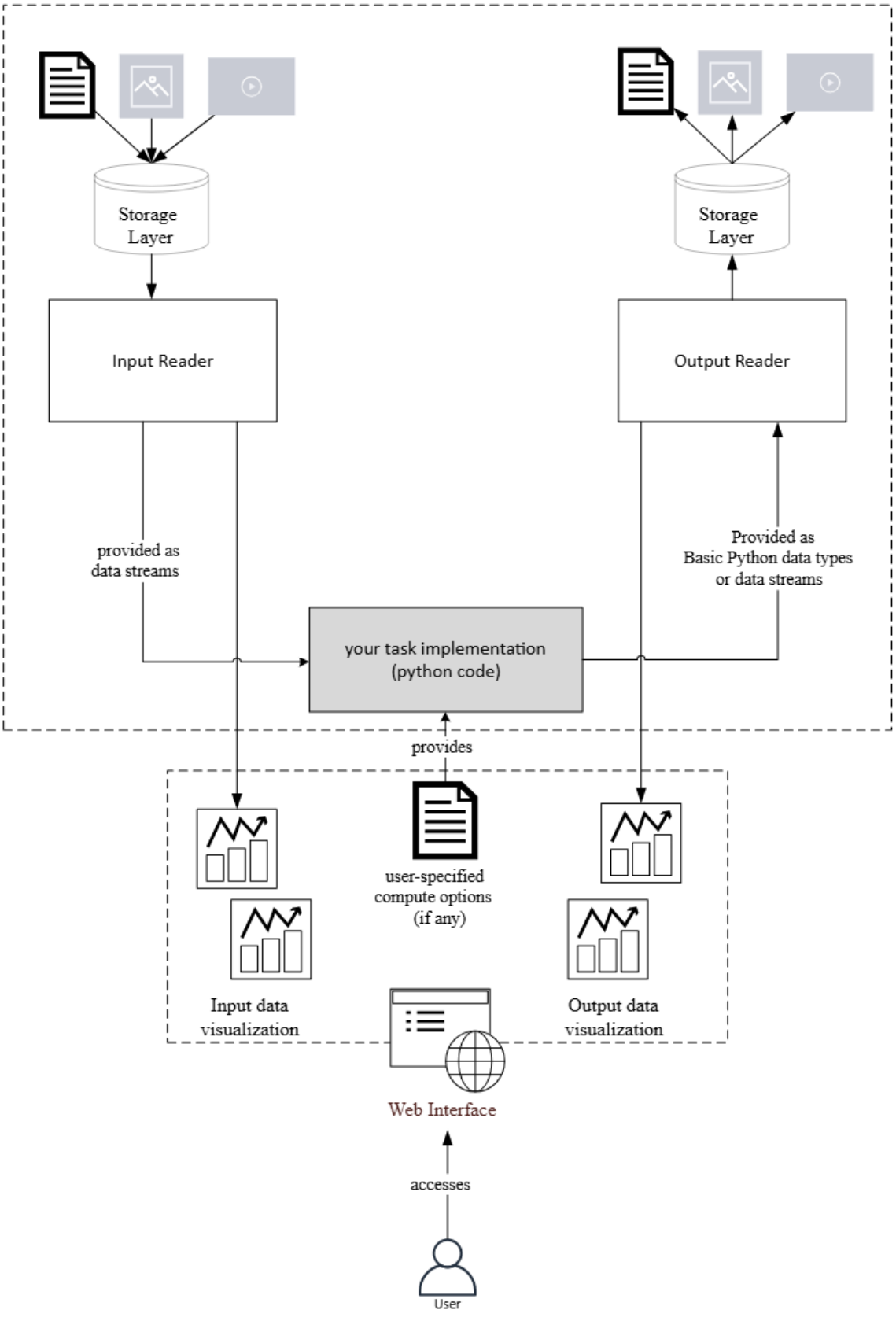


Figure 4. SmartcityCloud Compute Task Wrapper

2.2 Data Centric Artificial Intelligence

Data-centric Artificial Intelligence (AI) differs from the traditional model-centric approach by emphasizing the systematic improvement of data quality rather than solely focusing on algorithmic sophistication [2]. While model-centric AI optimizes model architectures and parameters based on fixed datasets, data-centric AI recognizes that data accuracy, completeness, and consistency are equally crucial for overall system performance. It treats data as a first-class element, requiring iterative refinement, curation, and validation to ensure models learn from high-quality, representative, and well-labeled samples. This involves creating standardized and balanced datasets, removing noise, and improving label consistency to enhance model generalization.

Within the SmartCityCloud environment, this paradigm is implemented to increase the reliability of heterogeneous sensor data used for urban intelligence and predictive analytics. Instead of relying solely on model accuracy, the system prioritizes data reliability, completeness, and consistency as prerequisites for effective AI-driven decision-making. The approach is realized through several sub-processes: data labeling and annotation, ensuring semantic accuracy and temporal alignment (e.g., day/night label quality checks); data augmentation, which enhances robustness via interpolation and noise-based synthetic generation; feature engineering, enriching time-series data with lag, rolling, and trend-based features; and out-of-distribution (OOD) generalization, managing domain shifts to maintain stability under varying data conditions. Together, these components embody the theoretical and practical foundation of this thesis, demonstrating how SmartCityCloud operationalizes data-centric AI to produce high-quality, context-aware datasets that enable reliable and reproducible predictive modeling.

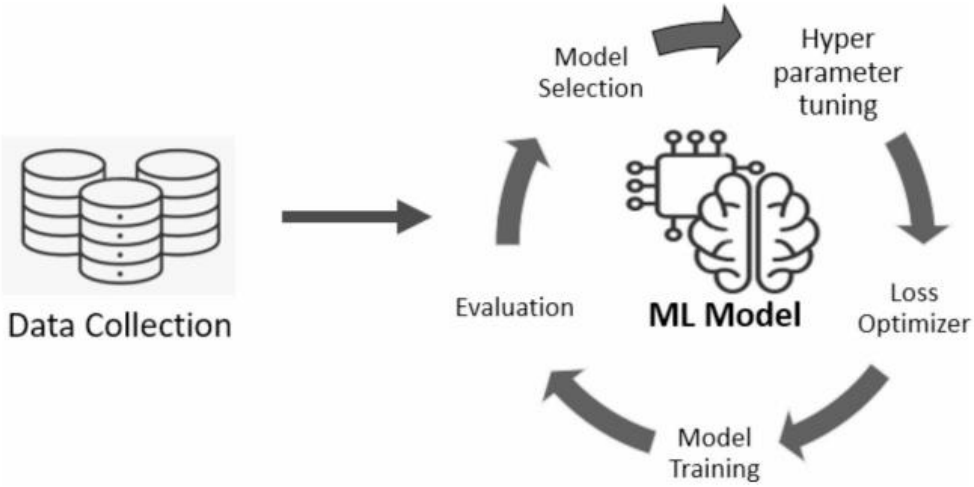


Figure 5. The steps for Model Centric Approach [14]

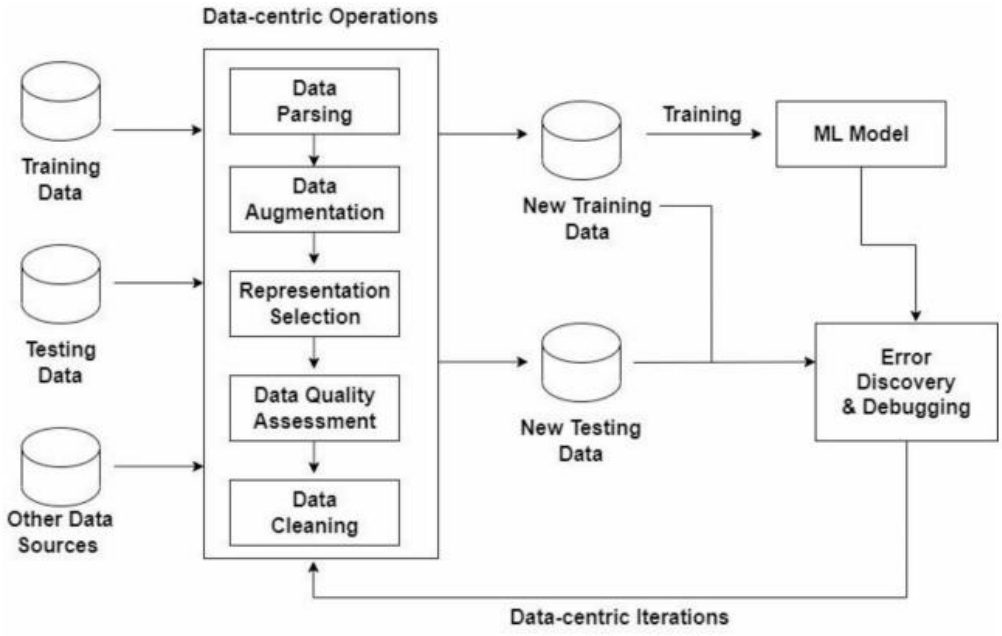


Figure 6. The Steps for a Data-Centric Approach [14]

Fig 5 outlines the traditional model-centric loop: construct or select a network architecture, tune hyperparameters, and repeat on algorithms while assuming the dataset is essentially fixed [14]. This underlines how this approach has traditionally outperformed architectures such as AlexNet, VGG, GoogLeNet, and ResNet—but also points out its susceptibility in real-world circumstances when data contains inconsistencies, bias, noise, and missing values [15]. In other words, when urban sensor feeds are defective (as smart-city streams frequently are), simply refining models cannot compensate for label noise, duplication, or gaps; data quality issues become a performance barrier. This explains the shift that inspired your thesis: to move away from a model-only approach and face data reliability head-on with SmartCityCloud.

Fig 6 depicts the core processes of a data-centric pipeline—data parsing, augmentation, representation, quality assessment, and cleaning—arranged as systematic phases to improve data before model training. The paper [14] outlines concrete tactics, including multi-stage hashing for duplicate removal, "confident learning" for noisy-label detection and correction, and controlled augmentations. These measures consistently outperform model-centric baselines (≥3% relative gains in their experiments). SmartCityCloud implements the same concepts: parsing/representation via stream readers for CSV/Excel/image inputs; quality assessment & cleaning via missing/validity checks, duplicate handling, interpolation, and flatline detection; label quality via day-night alignment; augmentation & feature engineering to enrich time-series (lags, rolling statistics, diffs); and stability/OOD checks for drift-aware

robustness, with all steps captured as exportable artifacts (HQ Thus, the figure's data-centric workflow corresponds precisely to your SCC compute-task implementation and supports your emphasis on engineering data quality as the fundamental lever for reliable smart-city prediction.

2.3 Data Quality Dimensions and Evaluation Metrics

Data quality is the degree to which data is suitable for its intended analytical purpose, as measured by criteria such as correctness, completeness, consistency, timeliness, traceability, and auditability (as defined in standards such as ISO/IEC 25012) [16]. Operationally, accuracy reflects closeness of values to true or physically plausible ranges; completeness measures the proportion of required values present; consistency captures the absence of contradictions, duplicates, or implausible flatlines across time; timeliness concerns whether timestamps are valid and appropriately aligned with the phenomena observed; traceability denotes the ability to follow data and transformations through the pipeline; and auditability requires These parameters offer a rigorous, implementation-independent perspective for determining whether sensor streams are reliable inputs to prediction models.

In this thesis implementation, each dimension is instantiated by concrete, reportable metrics within the SmartCityCloud compute task. From Table 1, Completeness is quantified via missing-value counts and percentages; accuracy is enforced through range/validity screening against realistic domain thresholds; consistency is supported by duplicate timestamp removal and flatline-run detection; timeliness is addressed through robust timestamp parsing, ordering, and resampling checks; and traceability/auditability are achieved by exporting machine-readable JSON provenance alongside high-quality (HQ) datasets, capturing configuration choices, transformation counts (e.g., invalid→NaN, imputed values), and evaluation summaries. The NiceGUI-based UI surfaces these metrics as tables, plots, and scalar cards, while the saved artifacts ensure reproducibility. Together, these measures define a practical data readiness framework: only when the dataset meets acceptable thresholds across all dimensions can downstream predictive modeling be trusted, a principle later applied in the Results chapter to interpret performance in terms of measurable data quality, not model tuning alone.

Criteria	Description / Purpose	Implementation in SmartCityCloud
Completeness	Measures how much of the required data is available and not missing.	Detect and calculate using missing-value percentages across records and time intervals.
Accuracy	Ensures that recorded values are valid and within realistic domain thresholds.	Check using range/validity screening for each numeric attribute.
Consistency	Verifies that data remains uniform and logically coherent over time.	Handle through duplicate timestamp removal and flatline run detection.
Timeliness	Checks whether timestamps are valid, sequential, and correctly aligned.	Validate using timestamp parsing, ordering, and resampling steps.
Traceability	Maintains the ability to trace data sources and transformations.	Achieved through JSON provenance files capturing dataset lineage and applied operations.
Auditability	Ensures all operations are documented and reproducible.	Export as versioned HQ datasets and JSON reports for reproducibility and review.

Table 1. Data Quality Evaluation Criteria

2.4 Data Governance and Versioning

During the implementation, data governance and version control are critical for assuring openness, reproducibility, and accountability across the SmartCityCloud-based data quality workflow. To ensure controlled code evolution and reproducible experimental states, a separate working branch was built in the SmartCityCloud Compute Task Wrapper GitLab repository. PyCharm's local Python development environment was linked directly to this branch, allowing for smooth synchronization of implementation updates with the version-controlled repository. Each alteration to the thesis-related modules, such as data ingestion, exploratory analysis, or quality processes, was routinely committed and pushed to the GitLab branch, resulting in a traceable history of all code revisions. This integration not only enforces version control but also enables collaborative reproducibility, with each modification or enhancement

noted, vetted, and retrievable. Furthermore, the developed interface allows users to download the created high-quality datasets corresponding to specific features in both CSV and Excel formats, promoting open data practices and allowing downstream verification of processed outputs.

Beyond software versioning, the system reflects the larger principles of data governance and provenance tracking that are essential to modern AI-powered infrastructures. Data governance in this context ensures that every dataset transformation, from ingestion to export, is documented with complete metadata, configuration parameters, and quality indicators. This architecture is informed by tools and principles similar to Apache Atlas [17] or Git-based metadata tracking, which ensure that datasets are auditable and traceable across processing cycles. The SmartCityCloud system puts these standards into action by automatically providing JSON provenance files and quality reports that detail dataset lineage, transformation stages, imputation counts, and validation results. The generated High-Quality (HQ) datasets are versioned and accompanied by metadata artifacts, providing an immutable audit trail for each analytical run. Collectively, these mechanisms reinforce reproducibility and reliability—core tenets of data-centric AI—and lay the groundwork for the implementation workflow that will be described in Chapter 4, which transforms theoretical principles into an operational, cloud-based data quality management system.

2.5 Summary

Chapter 2 established the theoretical and architectural foundations for this work by introducing SmartCityCloud (SCC) as a modular platform that standardizes sensor-data ingestion, processing, visualization, and output generation through its Compute Task Wrapper. It highlighted the shift from model-centric to data-centric AI, emphasizing that high-quality, well-structured data is essential for building reliable predictive systems in dynamic smart-city environments. The chapter also defined key data-quality dimensions—completeness, validity, consistency, timeliness, traceability, and auditability—and linked them to measurable indicators such as missing-value ratios, range checks, timestamp correctness, duplicate detection, and metadata provenance. Finally, it underscored the role of data governance practices, including versioning and reproducible pipelines, which ensure transparency and long-term reliability. Together, these fundamentals form the conceptual basis for the methodology and implementation developed in the subsequent chapters.

3 State of the Art

This chapter surveys the state of the art to ground the thesis in current research and to justify the design choices made later in the implementation. The subsections are organized by current research trends; in each, the review (i) synthesizes representative existing solutions and their technical approaches, (ii) identifies the gaps and limitations that arise in heterogeneous smart-city time-series (e.g., missing/invalid data, label inconsistencies, weak provenance, or distributional drift), and (iii) explains how the thesis responds within SmartCityCloud—through a modular compute-task that operationalizes data labeling quality, augmentation, feature engineering, data governance/versioning, and OOD stability. For every trend, the implications for implementation are made explicit: what must be supported in the options/UI, what checks and metrics are computed, what artifacts are exported (HQ datasets and JSON provenance), and how these choices improve downstream reliability and reproducibility. The chapter closes with a concise comparison table mapping each trend to its leading solutions, the uncovered gaps, and the thesis's concrete remedies, providing a direct bridge to the methodology and implementation that follow.

3.1 Data-Centric AI and Quality Engineering

- **Research Trend:** Data-centric artificial intelligence shifts the supervised learning optimization focus from model architecture to data engineering [18]. Rather than assuming a fixed dataset and focusing primarily on networks and hyperparameters, data-centric practice prioritizes label fidelity, coverage and balance, validity and range conformance, temporal integrity, and lineage documentation—on the assumption that model performance gains quickly saturate if underlying data issues persist. Canonical position pieces and surveys[19] describe this as a disciplined toolkit for designing datasets—not merely enlarging them, but making them more appropriate for the task via schema standards, labeling protocols, cleaning, augmentation, and governance mechanisms that render pipelines reproducible and auditable.

  Fig 7 illustrates the increasing global research interest in data-centric AI by showing the sharp growth of publications containing the keyword "data-centric AI" on Google Scholar over recent years. The trend indicates that, although still an emerging discipline, the focus on improving data quality, labeling, and curation is rapidly gaining momentum across AI communities [20]. The figure highlights that most progress so far has concentrated on training-data development—cleaning, annotation, and augmentation—while comparatively

little attention has been given to data maintenance and inference-data design, especially within scientific and engineering domains. The author notes that this surge in research activity has not yet been mirrored strongly in domain-specific fields such as Earth and space sciences, implying a major opportunity for applied research to adopt and operationalize data-centric practices. In the context of this thesis, Figure 2 reinforces the motivation for implementing a unified SmartCityCloud framework that embodies these evolving global efforts: transforming theoretical advances in data-centric AI into a practical, reproducible system for data-quality engineering and management in heterogeneous smart-city datasets.

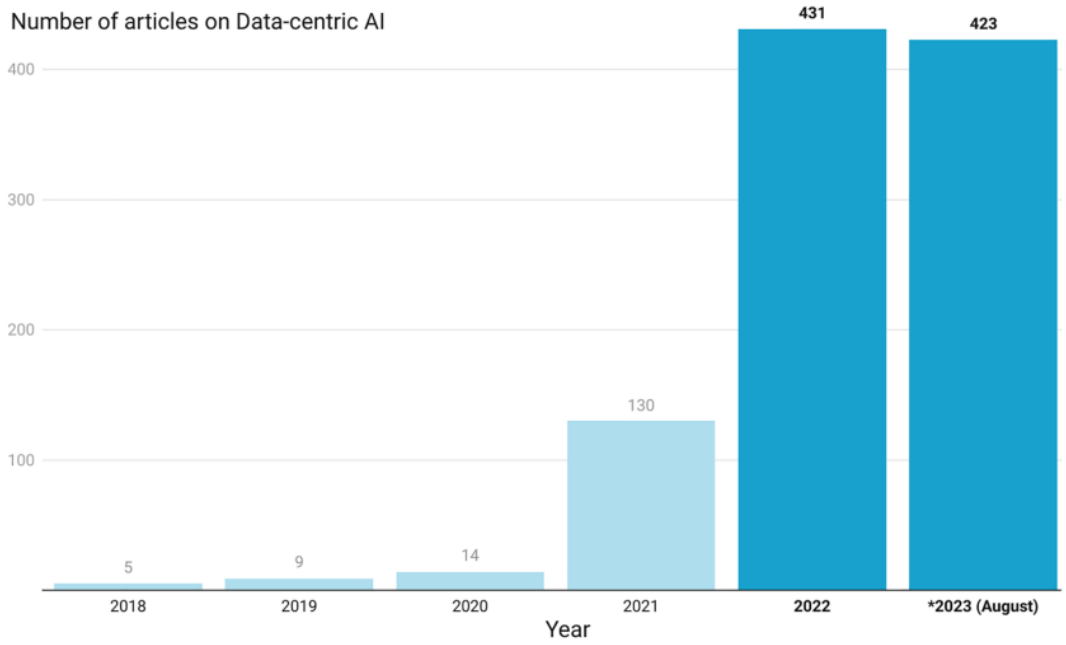


Figure 7. Recent articles published with the keyword "data-centric AI" [20]

Current research highlights several complementary strategies contributing to Quality Engineering. Label quality assurance has become a focal point, with methods like Confident Learning (CL) designed to identify mislabeled data and quantify label noise, significantly enhancing model accuracy [21]. Data augmentation techniques—such as interpolation, warping, and noise injection—have proven effective in increasing dataset diversity and robustness against overfitting[22]. Furthermore, feature engineering and data validation frameworks like Deequ by Amazon enable scalable quality checks and rule-based data profiling [23]. Collectively, these methods represent a growing global effort to embed data quality improvement directly into the AI development lifecycle rather than treating it as a preprocessing step [24].

- **Existing Solutions:** The implementation of data-centric AI principles has gained momentum across multiple domains, leading to the development of frameworks, tools, and algorithms specifically aimed at improving data quality and reliability. One of the most widely recognized approaches is Confident Learning (CL), which identifies and corrects mislabeled samples by estimating label confidence and uncertainty [21]. This method enhances dataset integrity and ensures that training samples accurately represent their classes, a critical factor for improving the robustness and interpretability of AI models. Similarly, the Data-Centric AI Initiative emphasizes systematic data curation, validation, and documentation over endless model fine-tuning [25]. This initiative inspired global competitions, encouraging practitioners to clean and balance datasets for better generalization rather than modifying neural architectures.

  In parallel, significant progress has been made in automated data validation and profiling frameworks. Recently introduced Deequ, a library developed by Amazon that performs declarative data validation using constraint-based quality checks on large-scale datasets [23]. This tool enables data engineers to automatically detect anomalies, validate numerical ranges, and measure data completeness and uniqueness—principles that are now fundamental to modern DCAI workflows. Additionally, data augmentation techniques such as interpolation, extrapolation, and synthetic sampling can be used to enrich datasets, thereby increasing model generalization while reducing overfitting [22]. These augmentation approaches are particularly relevant for dynamic, time-dependent data, like that in smart-city environments, where data collection is continuous and heterogeneous.

  Collectively, these existing solutions represent a significant step forward toward operationalizing data-centric principles. They establish a foundation for systematic data quality engineering, integrating labeling verification, automated validation, augmentation, and provenance management. However, most of these tools address isolated data-quality aspects and are not yet unified into a cohesive, reproducible cloud-based framework. This fragmentation underscores the need for integrated solutions—such as the SmartCityCloud Compute Task Wrapper developed in this thesis—that combine these diverse DCAI techniques into a single, automated, and scalable environment for ensuring high-quality data in smart-city AI applications.

- **Gaps Identified from Current Research:** Despite rapid progress, three integration gaps remain critical for heterogeneous smart-city time-series.

    - Many pipelines apply single data-centric techniques in isolation—e.g., label cleaning with Confident Learning or Cleanlab—without coupling them to temporal validity checks (range/physical plausibility), duplicate/flatline screening, or feature enrichment, making it hard to attribute gains and to certify cross-dimensional "data readiness." Evidence from label-error studies [21] shows meaningful accuracy gains from data cleaning, but most implementations stop short of tying label quality to broader validation and augmentation regimes.
    - Even when validation frameworks are used, many deployments lack embedded, machine-readable provenance that captures options, thresholds, and transformations, limiting auditability and reproducibility across teams and runs; this emphasizes declarative data-quality "guardrails" (e.g., Deequ, TFDV) and automated constraint generation, underscoring how often such guardrails are ad-hoc in practice[23].
    - Cleaning and augmentation are common, but stability is rarely quantified under diurnal/seasonal patterns, sensor drift, or site changes in streaming contexts; recent surveys highlight that OOD generalization for time series remains under-systematized and needs explicit evaluation protocols in operational pipelines. Out-of-Distribution Generalization in Time Series [26]. Together, these gaps mean that—even where individual procedures exist—end-to-end, verified fitness-for-use is not consistently achieved in real deployments.

- **How does This Thesis address the Gaps? :** This thesis consolidates data-centric practices into a single, auditable SmartCityCloud compute task that operationalizes quality engineering for urban sensor streams. Integrated, not isolated. The pipeline combines label verification (day–night alignment, inspired by label-error detection's focus on fidelity), data validation (range/validity thresholds; duplicate-timestamp removal; flatline-run detection; missing-value accounting), and time-series enrichment (interpolation regimes; noise-based perturbations; lag/rolling/differencing features), aligning with evidence that curated labels and augmented, validated data yield larger gains than further model tinkering. Embedded provenance for auditability. Each run emits structured artifacts—tables, plots, and a JSON provenance record of configuration, thresholds, and transformation counts (e.g., invalid→NaN,

imputations)—operationalizing the “guardrails” advocated by modern DQ tooling and enabling reproducibility and change tracking practices. Explicit robustness checks include stability/OOD diagnostics (e.g., monthly drift summaries over selected attributes), so data readiness is measured not only at a snapshot but also under realistic distributional variation, reflecting current calls to make OOD evaluation a first-class component in time-series pipelines. By turning label quality, validation, augmentation, provenance, and OOD stability into first-class, configurable steps within one cloud task, the thesis translates data-centric AI from principle to a cohesive service for dependable smart-city prediction.

3.2 Cloud-Native, Governed Data-Quality Pipelines with OOD Monitoring

- **Research Trend:** Data Analysis and Validation converging on end-to-end, cloud-native data-quality pipelines that automate validation and profiling as first-class stages in ML/AI workflows, rather than ad-hoc preprocessing. Production frameworks such as TensorFlow Data Validation (TFDV) and TFX demonstrate scalable, declarative checks for schema drift, anomalies, and distribution changes embedded directly in continuous pipelines, signaling a move toward “data unit tests” at scale [27]. In parallel, Deequ and Great Expectations operationalize constraint-based quality verification and human-readable quality reports, enabling teams to specify expectations (completeness, uniqueness, ranges) and to materialize versioned validation artifacts that can live alongside code in CI/CD [23]. On the governance side, organizations increasingly adopt metadata lineage and cataloging (e.g., Apache Atlas) so that datasets, transformations, and quality checks are discoverable and auditable across platforms—an essential prerequisite for regulated or safety-critical AI. Complementing these quality and governance layers, MLOps tooling (e.g., MLflow) standardizes experiment tracking and model/data versioning to support reproducibility across teams and time[28].

  A second, tightly related strand addresses robustness under distribution shift for streaming and time-series data. Foundational surveys on concept drift emphasize that in live environments, the relationship between features and targets changes, requiring continuous monitoring, adaptive evaluation windows, and drift-aware retraining triggers [29]. Also, out-of-distribution (OOD) generalizes for time series, highlighting protocols and metrics for assessing

stability when operational data departs from training regimes (seasonality changes, sensor aging, deployment to new sites) [26]. Finally, to make these pipelines accountable, research communities reference ISO/IEC 25012 and contemporary DQ surveys to formalize evaluation dimensions (accuracy, completeness, consistency, timeliness, traceability, auditability) and to tie automated checks to measurable “data readiness” scores that can be reported and reviewed [30].

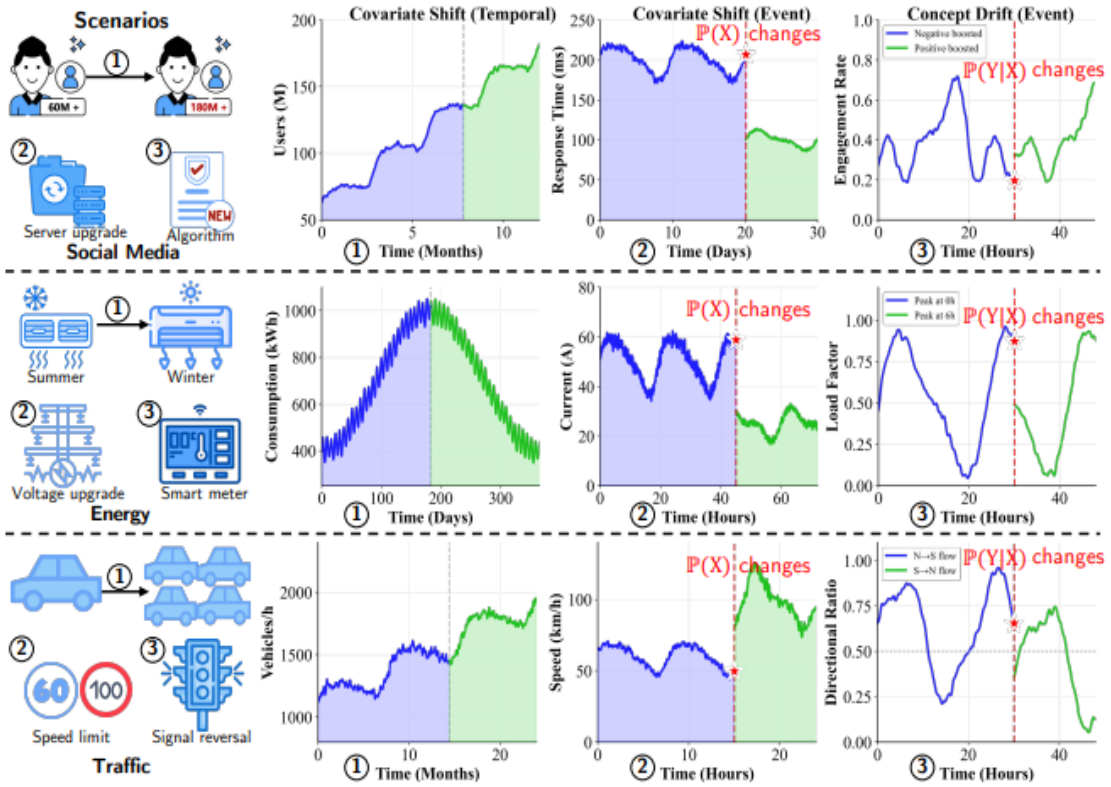


Figure 8. Covariate Shift and Drift with In-Distribution vs. OOD Periods [26]

The Fig 8 illustrates the central challenge of Out-of-Distribution (OOD) generalization in time-series data by showing how real-world sensor and social data evolve through covariate shift—changes in the input distribution $P(X)$—and concept drift—changes in the relationship between inputs and outputs $P(Y \mid X)$. It depicts these shifts across domains such as social media, energy, and traffic, where variations arise from natural evolution, seasonal patterns, or abrupt external events like server upgrades or policy changes. The blue regions represent periods used for training (in-distribution), while the green regions mark future periods where data distributions differ, causing model degradation if unaccounted for. This visualization underscores that non-stationarity is an inherent property of real-world time-series and that AI pipelines must incorporate continuous drift detection, adaptation, and monitoring to sustain

predictive reliability—a principle directly relevant to SmartCityCloud's goal of ensuring robust and auditable data quality across evolving urban data streams.

- **Existing Solutions:** Recent advances in cloud-native data-quality frameworks have enabled automated and scalable data validation within AI pipelines. Tools such as TensorFlow Data Validation (TFDV) and Amazon Deequ perform large-scale anomaly detection, schema validation, and constraint-based data profiling directly in production environments [31]. These frameworks operationalize data-quality checks as "data unit tests," ensuring that completeness, consistency, and range compliance are continuously verified. However, while they automate many aspects of quality assurance, most remain domain-agnostic and do not integrate downstream provenance tracking or contextual drift analysis—critical requirements for real-time smart-city data streams.

  Complementing these validation frameworks, data governance and versioning tools such as Apache Atlas and MLflow provide metadata management, lineage tracking, and experiment logging to maintain reproducibility across evolving datasets and models. However, these platforms often function in isolation and lack deep integration with quality metrics or time-series drift analysis, making it difficult to trace how data changes impact model behaviour. Research in Out-of-Distribution (OOD) generalization and drift detection further extends this landscape, proposing statistical and adaptive methods to identify data distribution changes over time [32]. Frameworks such as Evidently AI visualize drift trends and monitor model degradation, yet most remain limited to static or model-centric monitoring and fail to incorporate contextual temporal drifts typical in urban sensor data streams.

  At the governance and accountability level, initiatives like ISO/IEC 25012 and open-source platforms such as Great Expectations aim to formalize measurable quality dimensions—accuracy, completeness, consistency, timeliness, traceability, and auditability. These frameworks highlight the need for transparency and standardization but often lack automation or unified provenance tracking. In contrast, this thesis integrates these fragmented developments into the SmartCityCloud compute-task environment, combining automated validation, JSON-based provenance recording, version-controlled dataset exports, and OOD drift monitoring within a single, cloud-native system. This holistic approach transforms disconnected research advances into an

operational, reproducible, and auditable framework tailored for data-quality engineering in smart-city time-series analytics.

- **Gaps Identified from Current Research:** Despite significant progress in establishing cloud-based frameworks for data quality validation, governance, and drift detection, the majority of existing techniques are fragmented and domain-agnostic. Tools like TensorFlow Data Validation (TFDV) and Amazon Deequ provide robust mechanisms for anomaly detection and schema validation, but they are primarily designed for general-purpose machine learning pipelines and lack domain-specific adaptability for heterogeneous time-series data, such as those found in smart-city environments [31]. These frameworks primarily evaluate data at a single moment in time, with no consideration for temporal dependencies or multi-modal data interactions among sensors, environmental variables, and event-driven dynamics. Furthermore, while MLflow and Apache Atlas help with governance and version control, they work as separate tools, thus metadata lineage, validation metrics, and quality findings are frequently decoupled, making end-to-end traceability problematic. As a result, the integration of data validation, governance, and drift monitoring remains poor, prohibiting firms from developing pipelines that are both auditable and continually adaptable to changing data patterns.

  Another critical gap lies in the limited treatment of Out-of-Distribution (OOD) generalization and drift evaluation within current data-quality pipelines. Highlights from the importance of identifying distributional and conceptual shifts, yet these techniques are seldom embedded within automated quality frameworks. Most tools focus on syntactic or statistical validation, overlooking semantic drifts such as those arising from seasonal variations, sensor recalibration, or environmental changes that influence real-world predictive performance. Additionally, while frameworks like Evidently AI provide visualization dashboards for monitoring drift, they often lack automated provenance generation and version-linked reporting, which are essential for reproducibility and accountability in AI-driven decision systems [33]. Finally, standardized data-quality frameworks such as ISO/IEC 25012 define the theoretical dimensions of data quality—accuracy, completeness, consistency, timeliness, traceability, and auditability—but do not provide the computational mechanisms to measure or enforce them in streaming environments. Consequently, existing research does not yet deliver a unified, domain-aware, and drift-resilient solution that ensures data readiness over time, leaving a

crucial implementation gap that this thesis addresses within the SmartCityCloud ecosystem.

- **How This Thesis Addresses the Gaps:** This thesis bridges the identified gaps by developing a unified SmartCityCloud compute-task framework that operationalizes cloud-native data-quality engineering with integrated governance, provenance, and drift evaluation. Unlike existing fragmented systems, the proposed solution automates the entire data-quality lifecycle—from validation and profiling to versioning and auditability—within a single, reproducible environment. The compute-task pipeline consolidates missing-value detection, range and validity checks, duplication and flatline screening, and timestamp validation while automatically generating JSON-based provenance records that capture configurations, thresholds, and applied transformations. Each processing run produces High-Quality (HQ) datasets in CSV or Excel formats, ensuring transparency and reproducibility. To address OOD and drift challenges, the system incorporates temporal drift detection and stability monitoring, quantifying feature and distributional shifts over time to maintain model robustness under real-world non-stationarity. By embedding these mechanisms into a modular, scalable cloud architecture, the SmartCityCloud platform transforms disparate research advances in validation, governance, and OOD evaluation into a coherent, auditable, and domain-specific data-quality pipeline for heterogeneous smart-city time-series analytics.

3.3 Summary

Research Trend	Existing Solutions	Gaps Identified	Approach of this Thesis
Data-Centric AI & Quality Engineering [24]	Systematic improvement of data over model tinkering; practices include dataset curation, cleaning, balancing, and governance embedded into the ML lifecycle.	Often treated as ad-hoc preprocessing; limited unification across labeling, validation, augmentation, and governance; weak audit trails in practice.	Implements an integrated SCC compute-task: unified validation (missing/validity/duplicates/flatlines/timeliness), label checks, augmentation, feature engineering, plus JSON provenance and HQ dataset export.
Label Quality & Noise-Aware Learning [21]	Confident Learning/Cleanlab to detect/correct mislabelled samples; noise-aware loss functions and relabelling protocols.	Label fixes are rarely linked to time-series validity (e.g., day–night semantics) or to downstream governance; impact attribution across dimensions is unclear.	Adds day–night label alignment and label-quality reporting inside SCC; records decisions to link label edits to quality/robustness outcomes.

Data Augmentation for Time Series [22]	Jitter/noise, time warping, window slicing, mixup, interpolation; used to increase diversity and reduce overfitting.	Generic augmentations may distort physics/semantics; rarely coupled with drift/stability evaluation or with provenance of synthetic data.	Provides controlled interpolation and noise regimes tied to sensor meaning; exports augmentation configs in JSON and evaluates stability on monthly OOD windows.
Feature Engineering for Analytics [34]	Lag features, rolling mean/std, differences, calendar/diurnal encodings; widely used in classical and hybrid ML pipelines.	Feature creation is often decoupled from validation & governance; limited reporting of feature provenance and effect on stability.	Implements lag/rolling/diff features with automatic logging (config → JSON), and links feature sets to stability metrics and HQ exports.
Cloud-Native Data-Quality Pipelines & Automation [23] [31]	TFDV/TFX for schema/anomaly checks; Deequ/Great Expectations for constraint-based validation; CI/CD-style data unit tests.	Frameworks are domain-agnostic and fragmented; limited temporal semantics for heterogeneous smart-city streams; integration burden is high.	Bundles validation, visualization, and export in one SCC compute-task; domain-aware checks (timestamps, flatlines), and ready-to-use UI for non-experts.
Data Governance, Provenance & Version Control [16]	MLflow for experiment tracking; Apache Atlas/catalogs for lineage; Git-based versioning for code/models/datasets.	Metadata, validation metrics, and datasets often live in separate tools, with er4t565weak end-to-end traceability across runs and teams.	Links PyCharm to a dedicated GitLab branch; emits run-level JSON provenance (options, thresholds, transformations) and versioned HQ datasets (CSV/XLSX).
OOD Generalization & Drift Detection [32]	Statistical drift metrics (KL/JS/Wasserstein), windowed tests; surveys on concept drift and time-series OOD.	Embedded OOD checks are rare in DQ pipelines; little separation of ID vs OOD windows; minimal linkage to governance/audit.	Adds monthly drift summaries and stability diagnostics; separates ID/OOD periods in evaluation and logs outcomes in provenance.
Explainable & Auditable Data-Readiness Metrics [16]	ISO/IEC 25012 dimensions (accuracy, completeness, consistency, timeliness, traceability, auditability); WhyLogs/GE scorecards.	Standards specify *what* to measure but not *how* to automate in streaming; few systems bind metrics to artifacts for audit.	Computes metrics in-task and exports tables/plots + JSON; ties scores to concrete artifacts (HQ datasets, figures) for auditability.

Table 2. Summary

4 Methodology

This chapter provides the methodological foundation for the thesis by giving the systematic strategy, methodology, and experimental design used to address the identified research problem. It transforms the conceptual insights and research gaps identified in the literature review into a practical, operational framework for execution. The section describes the system architecture, data acquisition processes, preprocessing pipeline, analytical models, and evaluation strategies that allow for the engineering and assessment of data quality in diverse smart-city time series. By formally defining each processing stage—from data ingestion, cleaning, and transformation to outlier detection, imputation, augmentation, and quality evaluation—the chapter demonstrates how the proposed workflow improves on existing methods through automation, reproducibility, and explainability. Every design decision is supported by methodological explanations, ensuring that it meets the objectives of accuracy, completeness, consistency, traceability, timeliness, and auditability. This chapter serves as a bridge between the state-of-the-art analysis and the practical realization presented in Chapter 5, ensuring that the subsequent implementation and evaluation are based on a well-defined, transparent, and scientifically verifiable framework.

4.1 Overview and Design Rationale

This section describes the overall methodological philosophy and reasoning that led to the development of the proposed data quality framework. It teaches the fundamental concept of using a systematic, data-centric approach to creating, analysing, and enhancing the quality of heterogeneous smart-city time-series data. The presentation focuses on how the literature review findings influenced methodological selections, ensuring that the chosen methodologies addressed the specific issues of temporal heterogeneity, missingness, and outlier behaviour in sensor environments. This section also discusses the motivations for essential design principles such as modularity, openness, and repeatability, which ensure that each stage of data processing contributes demonstrably to the stated quality parameters. This section defines the methodological framework for the following architecture, algorithms, and experimental settings by explaining the intellectual underpinnings and reasons for the chosen tactics.

4.1.1 Purpose, Scope, and Quality Objectives of the Methodology

The goal of this methodology is to create a systematic, data-centric framework for designing, evaluating, and validating the quality of diverse time-series datasets derived from smart-city sensor settings. The framework describes a unified process that combines data ingestion, preprocessing, exploratory analysis, quality evaluation, and artifact generation into a single, reproducible pipeline. Within this setting, the methodology serves as the research's operational core, transforming conceptual findings from previous studies into a measured, automated, and transparent workflow that enables a quantitative assessment of data preparation.

This methodology encompasses the entire process, from raw data collection to the creation of high-quality (HQ) datasets and evaluation dashboards. It includes a variety of operations, such as timestamp alignment, duplication removal, validity screening, missing-value imputation, outlier detection using z-score and rolling statistics, feature derivation, and out-of-distribution (OOD) stability checks. Each of these processes is parametrically customizable, allowing the system to adapt to a variety of sensor properties and operational scenarios. This adaptability ensures that the same methodology may be used for numerous qualities and sensor types while preserving consistency and traceability in quality evaluation.

The technique directly addresses ten important data quality dimensions, as described in Table 3: accuracy, completeness, consistency, traceability, timeliness, and auditability. These dimensions collectively establish the standards for evaluating the fitness of smart-city time-series data in relation to the research objectives. Timeliness is verified by resampling and temporal aggregation mechanisms that track data delays or irregular intervals; accuracy is strengthened by detecting and eliminating erroneous or out-of-range values; completeness is ensured by identifying and imputing missing records; consistency is maintained by enforcing duplicate-free and range-constrained datasets; traceability is achieved through detailed JSON-based provenance reporting and versioned output artifacts; and auditability is realized by integrating transparent, UI-driven workflows that enable the reproduction and export. By defining these methodological purposes and objectives, this subsection situates the research framework as a bridge between theoretical understanding and practical validation. It establishes how the system operationalizes abstract data quality principles into measurable, interpretable, and automatable components, forming the foundation upon which the subsequent architecture, algorithms, and evaluation processes are built.

Research Objective	Methodological Component	Expected Outcome
Improve data accuracy	Outlier detection (z-score, range screening)	Reliable and error-free sensor values
Enhance completeness	Missing-value detection and imputation	Continuous time-series without gaps
Ensure consistency	Duplicate removal and temporal resampling	Uniform and stable data representation
Strengthen traceability	Provenance logging (JSON metadata)	Transparent and reproducible workflow
Maintain timeliness	Timestamp alignment and delay monitoring	Regular and up-to-date data streams
Support auditability	Exportable HQ reports and UI configuration tracking	Verifiable and reviewable analytical outputs
Assure overall data quality	Integrated quality metrics and six-dimensional evaluation	Quantified assessment of dataset readiness
Enable OOD generalization	Stability and drift screening across time segments	Robust performance under data distribution shifts
Improve feature representation	Rolling statistics and derived temporal features	Enhanced interpretability for downstream analysis
Simulate real-world variability	Data augmentation through noise and missingness injection	Improved model robustness and pipeline validation

Table 3. Mapping of research objectives to methodological components

4.1.2 Architectural Philosophy and Methodological Justification

The proposed methodology is based on a data-centric and pipeline-oriented architectural philosophy, with a focus on systematic data quality management as the foundation for analytical reliability. Unlike typical model-centric techniques, which promote prediction accuracy or algorithmic optimization, the accepted framework prioritizes data integrity, completeness, and consistency before beginning any modeling or evaluation process. This design perspective is consistent with current trends in data-driven system engineering, in which the quality of input data directly influences the validity of analytical results. The methodology uses a pipeline-first structure to ensure that each data transformation—from ingestion to high-quality (HQ) dataset generation—is clear, modular, and reproducible.

The architecture is organized into well-defined stages that include data ingestion, validation, cleaning, feature derivation, exploratory analysis, and quality evaluation, all of which are controlled by adjustable parameters. Each stage operates independently but cooperatively within the unified processing workflow. This modularity allows for scalability across several smart-city sensor streams, such as temperature, humidity, barometric pressure, and solar radiation, while preserving common data handling standards. The pipeline is designed in a tiered structure that separates stream management, option setup, and output creation, resulting in scalability and maintainability. This design choice enables the system to detect timestamps, handle duplication, rectify invalid or missing entries, and perform statistical quality checks without requiring operator interaction, resulting in a completely traceable and self-documenting process.

From a methodological standpoint, this data-centric design is supported by current literature, which emphasizes that the bulk of analytical inconsistencies in Internet-of-Things (IoT) and smart-city applications are caused by data-level shortcomings rather than model restrictions [20]. Previous research has shown that flaws, including missing values, outliers, and uneven sample intervals, can spread and amplify through downstream analysis, resulting in incorrect insights and poor model generalization [24]. As a result, the emphasis on preprocessing, imputation, and quality assessment before model training directly solves the cited limitations. The proposed methodology incorporates rolling statistics, z-score-based outlier detection, resampling, and out-of-distribution (OOD) drift analysis, which are empirically verified best practices for managing temporal and statistical variability in sensor environments.

Furthermore, the formalization of quality metrics—which encompass characteristics such as correctness, completeness, and timeliness—expands previous frameworks by including traceability and auditability as additional dimensions required for explainable AI and transparent data governance. This methodological emphasis on end-to-end data quality is consistent with TU Chemnitz's academic criteria for applied computer engineering research, which emphasize methodological rigor, reproducibility, and measurable progress over previous art. Thus, the selected architectural philosophy not only expands on existing research but also turns it into a realistic, automated, and extendable solution for real-world smart-city data pipelines.

4.1.3 Improvements and Formalization Plan

The proposed methodology significantly improves on existing data-quality assessment methodologies for smart-city time series by combining automation, repeatability, and provenance tracking inside a unified, user-driven framework. Traditional data-quality solutions frequently rely on static scripts or offline analysis, which necessitate human interaction and domain-specific configuration. In contrast, the proposed system automates the entire processing sequence—from data ingestion and validation to high-quality dataset production and evaluation—via a user-configurable task pipeline. This automation lowers human bias, operational errors, and assures consistent execution across different datasets and sensor properties.

A significant development is the inclusion of a graphical user interface (GUI) that enables dynamic configuration of preprocessing and quality-analysis tasks. Users can interact with numerical attributes, resampling intervals, z-score thresholds, and imputation algorithms without changing the underlying code. These UI-driven choices improve accessibility and transparency, allowing non-technical stakeholders to carry out reproducible experiments while retaining scientific rigor. The addition of provenance metadata via JSON-based reporting sets the system apart from previous approaches. Each execution contains critical parameters, statistical summaries, and data-quality measurements, ensuring total traceability and accountability—essential needs for explainable and auditable smart-city analytics.

From a scientific perspective, the methodology formalizes the entire data-quality workflow into measurable and mathematically expressible components. Subsequent sections of this chapter present the formal definitions for major processes, including range-based validity screening, z-score outlier detection, rolling statistical computation, missing-value imputation, and out-of-distribution (OOD) stability evaluation. Each operation is defined as a function $f_i(D, \theta_i)$ that maps an input dataset $D$and configuration parameters $\theta_i$to a transformed dataset and quality score. This mathematical formalization enables consistent comparison, parameter optimization, and quantitative evaluation of the quality dimensions—accuracy, completeness, consistency, traceability, timeliness, and auditability—defined earlier in this chapter.

Collectively, these innovations shift the approach from a human and fragmented process to a standardized, automated, and verifiable pipeline, offering both conceptual and operational advantages over previous literature. The chapter thus moves from establishing these theoretical components to their organized implementation and

assessment, serving as an analytical bridge between the research framework and the practical system realization detailed in Chapter 5.

4.2 System Architecture & Data Ingestion

This section describes the architectural design and operational flow of the proposed SmartCity data-quality framework, including how data is ingested, formatted, and prepared for analytical analysis. It explains the system's functional organization, including the interaction of core components such as the task runner, option parser, processing modules, and output writers, which together allow for automatic and reproducible data-quality review. The section also describes the input data streams, which are predominantly time-series sensor datasets in CSV and Excel formats, as well as the techniques for timestamp detection, attribute selection, and schema validation, all of which ensure structural consistency before analysis. It also provides variable experimental elements such as resampling frequency, z-score thresholds, imputation procedures, and output formats, all of which define the methodology's flexibility and adaptability. This section provides a complete overview of the system's architecture and data-handling pipeline, laying the groundwork for transforming raw sensor data into high-quality, verified, and analysis-ready datasets. As a result, it acts as the methodology's operational backbone, connecting the previously stated conceptual framework to the documented data-processing operations that follow.

4.2.1 System Context and Components

The proposed SmartCity data-quality framework works in a distributed computing environment, connecting the SmartCityCloud data repository to the CE GPU Server, where analytical activities are performed. The system is built on a modular and service-oriented architecture, which separates data storage, computation, and visualization components. The framework's central feature is a task-execution mechanism that can be configured to automate the whole data-quality evaluation process. The main internal components of this system are listed below:

- **Task Runner –** Serves as the central execution engine that orchestrates the complete pipeline once a user initiates a task. It loads the selected dataset, triggers the data-processing modules sequentially (validation, cleaning, analysis, and reporting), and manages the flow of intermediate and final results.
- **Option Parser –** Interprets the configuration parameters received from the web interface—such as attribute selection, resampling frequency, z-score threshold, imputation strategy, or export format—and converts them into executable

settings. This ensures reproducibility and parameter traceability for every experimental run.

- **Processors –** Represent the functional units that perform the actual data transformations, including timestamp normalization, duplicate elimination, range and validity screening, missing-value imputation, rolling-statistic computation, z-score-based outlier detection, and out-of-distribution stability analysis. Each processor outputs both transformed data and quantitative quality metrics.
- **Writers / Exporters –** Handle the generation and storage of final artifacts such as high-quality (HQ) datasets, JSON reports, and evaluation tables. They apply standardized file naming and versioning to maintain provenance and ensure that data are saved only when explicitly requested by the user.
- **UI Triggers –** Constitute the interactive bridge between the user and the computational backend. Actions such as Run Task, Save Results, or View Plots activate corresponding Python functions in the task runner, enabling real-time visualization and control through the graphical interface.

Together, these components enable an end-to-end automated process that transforms raw, heterogeneous sensor inputs into validated, high-quality datasets ready for analysis or visualization.

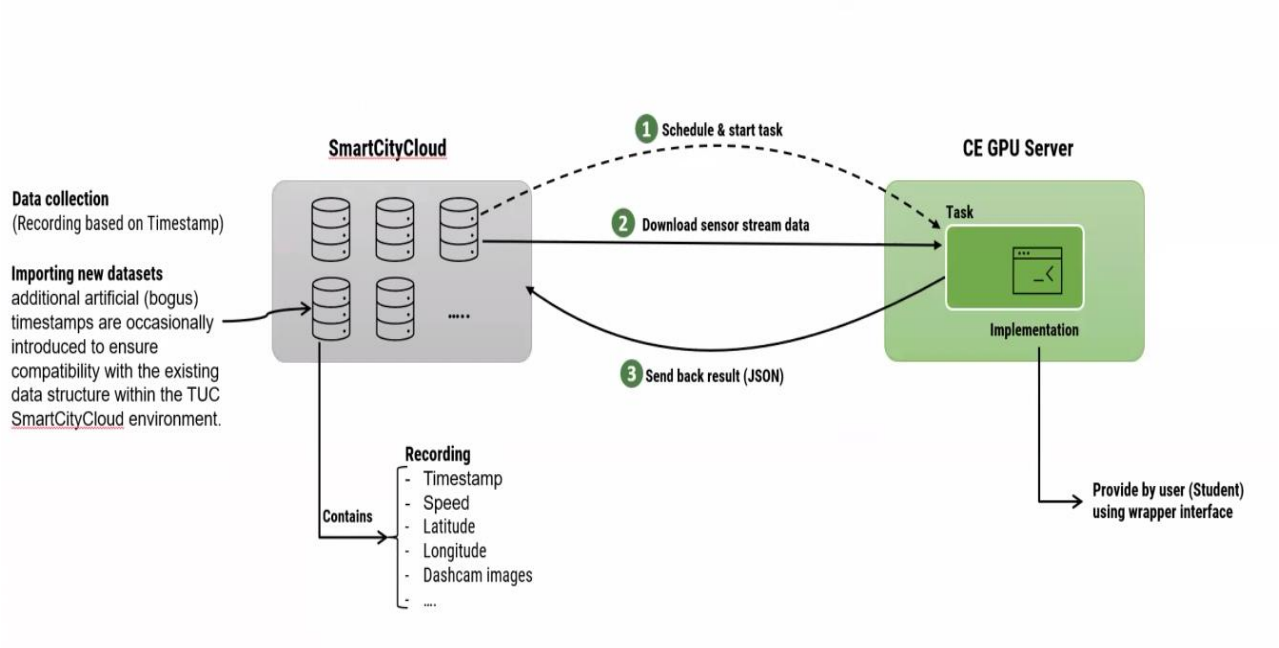


Figure 9. Architecture Overview

Fig 9 depicts the broader system environment, as well as the SmartCity Compute framework's overall Overview Architecture. The design has two core environments: the

SmartCityCloud and the CE GPU Server. SmartCityCloud acts as the persistent data repository where sensor streams are recorded based on timestamps and may include various attributes such as speed, latitude, longitude, or environmental parameters. When new datasets are added, additional artificial timestamps can be used to preserve temporal compatibility with the existing storage format, ensuring synchronization and consistent sampling for downstream processing. On the computational side, the CE GPU Server hosts the task logic created for this thesis. Users initiate a job through the web interface, which schedules and begins execution on the compute server. The workflow consists of three stages:

- Scheduling and configuring the task.
- Downloading the relevant sensor streams from SmartCityCloud
- Returning the processed results—typically JSON summaries or HQ datasets—to the platform for viewing or further evaluation.

This interaction establishes a clear line of responsibility: SmartCityCloud maintains data gathering and persistence, while the CE GPU Server handles computation, validation, and quality analytics. The cyclical interchange between these two levels serves as the operational backbone of the proposed technique, ensuring that data retrieval, processing, and reporting are seamlessly integrated, automated, and traceable inside the TU Chemnitz SmartCity computing ecosystem.

4.2.2 Data Sources, Stream Types, and Timestamp Normalization

The datasets employed in this study are derived from several air-quality and environmental sensor streams, constituting a broad and complex data ecosystem within a smart-city infrastructure. These datasets include observations of air temperature, barometric pressure, relative humidity, wind speed, and particulate matter concentrations, all of which are timestamped and collected from scattered monitoring sites. Multiple datasets are used to assess the framework's resilience and flexibility, which is consistent with the proposed methodology's data-centric mindset. This multi-dataset testing assures that the system is not specific to a single data source or device type, but rather generalizes across different sensor conditions, sampling rates, and data quality attributes, hence proving the pipeline's universality.

All sensor streams are collected in CSV or Excel formats, which are standard formats used in environmental and IoT data collecting systems. Each stream typically contains a timestamp column and one or more numeric attributes that indicate sensor readings. In keeping with the data-centric goal, the framework does not rely on a fixed schema and instead interprets each dataset dynamically upon intake. This allows for easy

integration of additional data sources with different attribute names and formats, allowing the methodology to function independently of the original data-collecting environment. The architecture allows for simultaneous management of many qualities, with the user picking specific numeric variables via the web interface for extensive quality analysis. This flexibility ensures that both univariate and multivariate datasets can be examined under identical methodological settings, further emphasizing reproducibility and comparability across experiments.

Timestamp discovery and normalization are crucial steps in the intake process, as they provide temporal consistency before performing analytical processes. As time alignment is the foundation of all subsequent processes—from resampling and imputation to outlier detection and OOD analysis—the system uses a hierarchical detection method to automatically locate the timestamp column. During dataset loading, the framework first looks for columns with names that match popular time-related identifiers (such as "timestamp," "time," "date," "datetime," and "recorded_at"). If numerous possibilities are identified, the algorithm selects the column with the highest proportion of successfully parsed date-time values. When explicit timestamp columns are missing or inconsistently constructed, the system automatically creates a temporal index based on row order or sampling intervals derived from metadata. This multi-layered detection approach guarantees robustness against schema variations, which is a common drawback in previous cutting-edge data-quality frameworks that commonly presume consistent timestamp fields.

Once found, timestamps are normalized to a consistent format (ISO 8601 standard) and transformed to a pandas DateTimeIndex for more efficient temporal operations. The normalization procedure also accounts for uneven intervals, missing time steps, and overlapping entries, which are prevalent in sensor-based recordings due to transmission delays or system resets. These corrected and ordered timestamps serve as the temporal foundation for subsequent quality assessment tasks such as missing-value analysis, resampling at predetermined granularities (hourly, daily, or monthly), rolling statistical computation, and time-dependent outlier detection. The result is a consistent time-series representation that retains both temporal precision and analytical integrity across datasets.

In summary, this part lays the groundwork for the technique by describing how disparate smart-city air-quality datasets are systematically digested, temporally aligned, and standardized into a single analytical structure. By automating timestamp discovery and supporting several dataset formats, the suggested technique addresses

interoperability issues that exist in existing systems. It thereby operationalizes the basic principles of data-centric research by emphasizing data readiness, flexibility, and cross-source generalization, all of which are critical to maintaining the validity and scalability of the SmartCity data-quality evaluation system.

4.2.3 End-to-End Workflow

The integrity of the data-quality analysis process is dependent on rigorous input validation and schema conformance before execution. The suggested system is meant to accommodate diverse datasets while maintaining a uniform logical schema to ensure correct interpretation and replication. Every dataset that is put into the system is first checked for minimum structural criteria, such as the availability of a timestamp column and at least one numeric characteristic suitable for statistical analysis. The ingestion layer automatically checks for data completeness, ensuring that timestamps are unique, chronologically ordered, and parseable into a single DateTimeIndex. Non-numeric or categorical columns are disregarded during analysis but remain in the metadata for traceability. To ensure analytical consistency, type casting is used to transform numerical fields to floating-point representations, and columns with incompatible datatypes are omitted from downstream computation. This pre-validation stage prevents schema mistakes, enforces uniform data types, and ensures that the following quality-assessment algorithms perform reliably.

Beyond structural validation, the system allows for a large range of experimental configuration parameters, known as the Option Space. These options establish the methodology's operational flexibility, allowing the same pipeline to be evaluated with a variety of parameter settings and data conditions. Each customizable piece is an experimental factor that can be controlled to alter data quality interpretation. The most crucial options are:

- **Resampling Frequency:** Determines the temporal granularity at which the data are aggregated—Hourly (H), Daily (D), or Monthly (M). This enables the analysis of short-term fluctuations versus long-term trends in sensor behaviour.
- **Z-Score Threshold:** Sets the statistical cutoff for outlier detection. Common values include ±2.5 or ±3, allowing control over sensitivity to extreme deviations.
- **Duplicate Handling:** Removes repeated timestamps or sensor readings, preserving the first valid occurrence to maintain consistency.
- **Imputation Method:** Specifies how missing values are replaced, with available strategies including forward fill (ffill), backward fill (bfill), or linear interpolation based on temporal index continuity.

- **Augmentation Preview:** Applies simulated perturbations (noise or artificial missingness) to assess pipeline robustness.

These options transform the system into a parameterized experimentation framework, supporting controlled comparisons across datasets and configurations. Each run is automatically logged with its specific parameter set, ensuring that experiments can be reproduced and independently verified. Once the inputs have been validated and the experimental parameters have been determined, the framework's end-to-end workflow is carried out in a well-defined series of steps. The process begins when the user picks an input dataset and starts the execution via the user interface. The task runner gets the desired file, applies the specified settings, and starts the data-processing pipeline. This pipeline includes the following important stages:

- **Input Acquisition and Validation:** The dataset is parsed, schema conformity is checked, and timestamps are standardized.
- **Preprocessing and Cleaning:** Duplicate entries are removed, invalid values are replaced with NaN, and missing data are handled through the configured imputation method.
- **Feature Derivation:** Rolling statistics, resampled means, and lag features are computed to reveal temporal patterns and anomalies.
- **Quality Evaluation:** Z-score and range-based outlier detection, completeness calculation, and OOD stability checks are performed.
- **Result Compilation and Export:** The final high-quality (HQ) dataset and its associated JSON report are generated, summarizing all statistics, parameter settings, and quality metrics.

This method produces tabular datasets, graphic plots (such as time series, boxplots, and trend analyses), and machine-readable JSON reports. Each result product contains complete provenance metadata, including processing timestamps, parameter configurations, and evaluation summaries, allowing for traceable and auditable documentation of each run. This automated report production also facilitates later performance benchmarking or sensitivity analysis by maintaining a consistent format across studies.

The tight integration of the user interface and backend processing is an important design aspect of this workflow. The user interface triggers are directly related to backend functions: "Run Task" starts data processing, "Show Results" shows plots and statistical summaries, and "Save Results" saves the HQ dataset and reports to the output folder. This combination of automation and user-driven setup improves

productivity and transparency by reducing the need for human data handling while still retaining total control over analytical parameters.

In essence, the combination of schema validation, parameter configurability, and automated end-to-end execution transforms the suggested technique into an adaptable and reproducible framework for assessing the quality of data in smart cities. The design not only supports numerous datasets and sensor types, but it also allows for systematic testing in a variety of scenarios, which is essential for data-driven research. The system's design combines a conceptual approach with operational pragmatism, ensuring that each stage—from input ingestion to artifact generation—is both computationally robust and scientifically clear.

4.3 Formal Processing Pipeline

This section provides a formal description of the whole data-processing pipeline that implements the methods provided in this thesis. It describes each computational stage, including its mathematical formulation and functional significance in transforming raw, diverse sensor data into analytically valid, high-quality datasets. The formalization ensures that the underlying procedures—from parsing and ordering to outlier detection and feature engineering—are not only implemented programmatically, but also described in a reproducible, verifiable manner that adheres to scientific norms. The pipeline is portrayed as a linear yet modular process that starts with timestamp alignment and schema validation, then moves on to range screening, missing-value treatment, and statistical outlier analysis.

4.3.1 Data Parsing and Validity Formalization

The formal processing of sensor data begins with defining the notation, data structures, and rules that establish the mathematical foundation of the proposed data-quality framework. Let each dataset D represent a multivariate time series consisting of n temporal observations. Each observation is indexed by a timestamp $\mathrm{t_i} \in \mathrm{T}$, where $\mathrm{T} = \{\mathrm{t_1}, \mathrm{t_2}, \ldots, \mathrm{t_n}\}$ denotes the ordered set of sampling times. The corresponding sensor measurement at time $\mathrm{t_i}$ is represented as $\mathrm{x_{t_i}} \in \mathbb{R}$, forming a sequence $\mathrm{X} = \{\mathrm{x_{t_1}}, \mathrm{x_{t_2}}, \ldots, \mathrm{x_{t_n}}\}$. Missing or invalid readings are treated as special cases within the dataset and represented as

$$x_{ti} = \begin{cases} NaN, & if\ the\ value\ is\ missing\ or\ invalid, \\ v_{ti}, & if\ the\ value\ is\ valid\ and\ measurable. \end{cases}$$

Accordingly, three disjoint sets are defined:

$$V = \{ x_{ti} \mid x_{ti}\ is\ valid \}, \qquad M = \{ x_{ti} \mid x_{ti} = NaN \}, \qquad I = \{ x_{ti} \mid x_{ti}\ is\ invalid \}$$

The first stage in the processing pipeline involves parsing, ordering, and duplicate handling**,** which ensures the dataset's temporal consistency. Parsing converts timestamp strings into numerical or datetime representations suitable for ordered computation. Let the mapping p:string → datetime define this conversion such that

$$ti' = p(ti), \qquad \forall ti \in T.$$

The timestamps are then sorted in ascending order $t_1' < t_2' < \ldots < t_n'$, establishing the chronological sequence required for temporal analytics. Duplicate timestamps often arise due to sensor synchronization issues or redundant data transmission. A duplicate predicate $\delta(t_i)$is defined as:

$$\delta(ti) = \begin{cases} 1, & if\ \exists\ tj \neq ti\ such\ that\ tj = ti \\ 0, & Otherwise \end{cases}$$

All instances where $\delta(t_i) = 1$ are flagged as duplicates, and the system retains only the first occurrence, following a keep-first policy. This decision preserves temporal continuity while preventing inflation of sample counts. The number of duplicates removed is logged as $N_{\text{dup}} = sum\ of\ \delta(t_i)$, forming part of the provenance record. This stage enhances consistency—one of the six quality dimensions—by ensuring that each timestamp uniquely identifies a single, valid observation. Once the dataset is temporally ordered and duplicates removed, the next phase applies validity screening, which ensures that all recorded sensor values fall within realistic operational limits. Each numeric attribute is associated with a predefined range function:

$$b(name) = (L, U)$$

where L and U denote the lower and upper acceptable bounds, respectively. For instance, for the air temperature attribute, $\mathrm{b(temperature)} = (-40{,}60)$; for barometric pressure, $\mathrm{b(pressure)} = (850{,}1100)$; and for relative humidity, $\mathrm{b(humidity)} = (0{,}100)$. A validity operator $\phi(\mathrm{x_{t_i}})$ determines whether a measurement falls within the defined range:

$$\phi(xti) = \begin{cases} 1, & if\ L \leq xti \leq U. \\ 0, & otherwise. \end{cases}$$

All values failing this criterion ($\phi(x_{t_i}) = 0$) are replaced with NaN, effectively marking them as missing for subsequent imputation. The total number of invalid entries is stored as $N_{inv} = \sum_i (1 - \phi(x_{t_i}))$, which contributes to the accuracy metric during data-quality evaluation. By enforcing these attribute-specific validity constraints, the framework eliminates physically implausible sensor readings and ensures the accuracy and reliability of downstream analyses.

Collectively, the parsing, ordering, and validity screening procedures establish the first layer of data quality assurance in the SmartCity pipeline. Parsing and ordering guarantee temporal integrity, while duplicate handling enforces consistency. Range-based screening ensures attribute-level validity and corrects systematic anomalies such as out-of-range spikes or negative physical quantities. These foundational transformations not only reduce noise and inconsistencies but also prepare the dataset for higher-level processes such as missing-value imputation, outlier analysis, and rolling-feature computation. Through their mathematical formalization, these operations transform raw sensor inputs into structured, interpretable, and quality-verified time series, thereby forming the essential basis for the subsequent stages of the data-quality framework.

4.3.2 Missing-Data Treatment, Outlier Detection, and Feature Engineering

An essential part of the formal data-quality pipeline is the treatment of missing values, identification of statistical outliers, and derivation of temporal features that capture trends, variability, and stability in sensor data. These processes transform irregular, incomplete, and noisy raw observations into structured and analytically robust sequences suitable for further evaluation.

Sensor-based time-series data are inherently prone to missing readings due to transmission errors, hardware malfunctions, or latency in data recording. In the proposed methodology, missing entries are denoted as NaN and are handled through a configurable imputation function. Let the observed series be $x_{t_1}, x_{t_2}, \ldots, x_{t_n}$, where some x_{t_i} are missing. The general imputation operator $\Im(x_{t_i})$ is defined as:

$$\hat{x}_{t_i} = \Im(x_{t_i}) = \begin{cases} x_{ti-1}, & for\ forward\ fill \\ x_{ti+1}, & for\ backward\ fill \\ x_{tk} + \left(\frac{(x_{tk+1} - x_{tk})}{(t_{k+1} - t_k)}\right) \times (t_i - t_k), & for\ linear\ interpolation \end{cases}$$

Here, t_k and t_{k+1} represent the timestamps immediately before and after the missing value. The method fills gaps using the chosen interpolation mode, ensuring temporal continuity while avoiding the introduction of unrealistic fluctuations. The number of imputed values, N_{imp}, is reported for each dataset as:

$$N\_imp = \Sigma\,[x_i = NaN \rightarrow replaced]$$

After imputation, the dataset is subjected to statistical outlier detection to remove or flag anomalous values. The framework applies the Z-score method, which standardizes each observation relative to the mean and standard deviation of the series. For every time point t_i, the standardized score is computed as:

$$z_i = (x_i - \mu_x) / \sigma_x$$

where μ_x is the sample mean and σ_x is the sample standard deviation. Any observation satisfying the condition $|z_i| > \tau$ is flagged as an outlier, where τ is a user-defined threshold (typically 2.5 or 3). These flagged points are marked as invalid and treated as missing (NaN) for subsequent imputation or exclusion. To complement Z-score detection, an Interquartile Range (IQR) method is applied to describe the overall distributional shape. The IQR is calculated as:

$$IQR = Q_3 - Q_1$$

Values outside the range $[Q_1 - 1.5 \cdot IQR,\ Q_3 + 1.5 \cdot IQR]$ are considered extreme and may indicate data drift. This dual outlier framework—combining Z-score and IQR screening—improves accuracy and robustness, ensuring that retained values represent physically plausible and statistically consistent measurements. Once the dataset is cleaned of missing and extreme values, the system computes derived temporal features that quantify evolving patterns and enhance interpretability. The feature-engineering stage generates higher-level metrics such as rolling mean, rolling standard deviation, first differences, and lag features. The rolling mean $\bar{x}_t$ and rolling standard deviation s_t over a moving window of size w are defined as:

$$\bar{x}_t = (1/w) \times \sum_{k-0}^{w-1} x_{t\_k}$$

$$s_t = (1/w) \times \sum_{k-0}^{w-1} (x_{t-k} - \bar{x}_t)^2$$

These features capture local trends and volatility, providing insight into the short-term stability of sensor readings. The temporal change between consecutive readings is computed as:

$$\Delta x_t = x_t - x_{t-1}$$

This highlights sudden deviations, often indicative of potential anomalies or system events. To detect prolonged sensor inactivity, the framework checks for consecutive identical values across Ktime steps. A flatline indicator F_t is defined as:

$$F_t = \begin{cases} 1, & if\ x_t = x_{t-1} = \cdots = x_{t-k}\ \ for\ K \geq threshold \\ 0, & otherwise. \end{cases}$$

Flatline detection helps identify faulty sensors that report constant values over extended durations—an important aspect of assessing data reliability. Together, the missing-data treatment, outlier detection, and feature-engineering stages form the core refinement layer of the SmartCity data-quality pipeline. These operations transform raw, noisy sensor readings into a structured, continuous, and statistically consistent time series, ready for subsequent resampling and high-quality dataset construction. By combining mathematical rigor with configurable parameters, the methodology ensures that the resulting data are accurate, complete, consistent, and interpretable, satisfying the key quality dimensions required for reliable smart-city analytics.

4.3.3 Resampling and High-Quality Dataset Construction

The concluding stage of the formal data-processing pipeline involves temporal resampling of the cleaned sensor streams and the construction of a High-Quality (HQ) dataset that satisfies all established data-quality dimensions. This stage ensures that all previously validated and imputed observations are aggregated at consistent temporal intervals and exported in a standardized structure suitable for analytical evaluation and visualization. The outcome is a dataset that is duplicate-free, temporally aligned, range-validated, statistically consistent, and complete—fulfilling the fundamental criteria of accuracy, completeness, and consistency in data-quality assessment.

Sensor data collected in smart-city environments are often recorded at irregular or device-specific intervals, which can hinder statistical comparison and temporal modelling. Resampling converts such irregular sequences into uniform time grids by aggregating values into fixed time buckets (e.g., hourly, daily, or monthly). Let the cleaned time-series after imputation be $X = \{x_{t_1}, x_{t_2}, \ldots, x_{t_n}\}$with corresponding timestamps t_i. For a chosen resampling frequency $f \in \{H, D, M\}$ (Hourly, Daily, or Monthly), each period P_j is defined as a set of timestamps belonging to that interval.

$$\bar{x}\_P\square = (1 / |P\square|) \times \Sigma\_\{t_i \in P\square\} x_{t_i}$$

Where $|P_j|$ is the number of valid points in the interval P_j. If $|P_j| = 0$, the resampled value is set to NaN, preserving transparency of missing intervals. The resampling operator transforms the original series into a new time-indexed series $X_f = \{\bar{x}_{P_1}, \bar{x}_{P_2}, \ldots, \bar{x}_{P_m}\}$, where m depends on the total observation window and the chosen frequency f. However, resampling introduces certain caveats: if a large proportion of data within a window is missing, the mean may not accurately represent the period's true conditions. Therefore, resampling is performed only after imputation and outlier correction to prevent distortion of underlying statistics.

Following resampling, the cleaned and temporally harmonized data are assembled into the final High-Quality (HQ) dataset. The HQ dataset represents the definitive product of the data-quality pipeline and serves as the foundation for evaluation and visualization. It adheres to a strict post-processing contract ensuring that all integrity and consistency conditions are satisfied. The formal definition of the HQ dataset D_{HQ} is given as:

$$D\_HQ = \{ x\square_i \in X \mid \delta(t_i) = 0, \varphi(x\square_i) = 1, x\square_i \neq NaN \}$$

where:

- $\delta(t_i) = 0$, ensures that no duplicate timestamps exist,
- $\varphi(x_{t_i}) = 1$, confirms that the value has passed validity screening,
- $x_{t_i} \neq NaN$, guarantees the absence of unhandled missing values.

Thus, every observation in D_{HQ} represents a verified, validated, and temporally aligned data point. To maintain reproducibility, the system records metadata such as:

- Number of duplicates removed (N_{dup}),
- Invalid values replaced (N_{inv}),

- Imputed values (N_{imp}),
- Outliers flagged (N_{out}), and
- Resampling frequency f applied.

This metadata is stored alongside the HQ dataset in a JSON report, ensuring transparency and auditability of every computational step. The HQ dataset construction phase also supports strict and flexible configurations. In strict mode, only records with complete values are retained, ensuring absolute completeness. By ensuring consistent temporal resolution, attribute validity, and completeness, the process transforms disparate raw sensor streams into comparable analytical units. This not only enhances data interpretability but also ensures that subsequent evaluation metrics—such as accuracy, completeness, and stability—are grounded in reliable data structures. Moreover, the creation of HQ datasets for each attribute forms the basis for comparative benchmarking across time, sensors, or configurations, a critical aspect of data-centric experimentation emphasized in this thesis.

4.4 Experimental Setup and Procedure

This section outlines the experimental setup and execution procedures used to test the proposed SmartCity data-quality framework. It defines the computing environment, datasets, parameter grid, and workflow for systematically testing and benchmarking the methodology's performance. The experimental setup is intended to ensure that all processes—from data ingestion to quality evaluation—are carried out under controlled, reproducible conditions, allowing for objective comparison across datasets and parameter settings. The technical context of experimentation is established by providing details on the hardware and software environment, the properties of the datasets under study, and the changeable experimental parameters. The section also explains the entire procedural flow that occurs through the system's user interface, emphasizing how user-defined options influence data processing and quality assessment. Finally, repeatability techniques including provenance logging, fixed random seeds, and consistent file-naming conventions are explored to ensure that all results are transparent and verifiable. This structured experimental design empirically demonstrates the suggested methodology's dependability, scalability, and generalizability.

4.4.1 Experimental Environment and Parameter Configuration

The experimental evaluation of the proposed SmartCity data-quality framework was conducted within a controlled computing environment designed to ensure consistent

performance, reproducibility, and scalability. All experiments were executed on a Smart city Compute Task wrapper. This configuration provided adequate computational resources for handling large-scale sensor datasets and performing time-series analyses efficiently. The software environment was implemented using Python 3.10, with core dependencies including pandas (v2.1.1) for data manipulation, NumPy (v1.26.0) for numerical operations, Matplotlib (v3.8.0) for plotting and visualization, and SciPy (v1.11.3) for statistical computation. The interactive user interface (UI) was developed using the NiceGUI framework, which enables seamless interaction between the Python backend and visualization frontend. All code modules, including data ingestion (csv_streams.py, excel_streams.py), processing (task_impl.py, options.py, output.py), and visualization (streamvis.py), were deployed within this environment and orchestrated by the SmartCity Task Wrapper for automated task scheduling and execution.

The framework was validated using multiple air-quality and environmental datasets collected from simulated smart-city sensor networks and open-source repositories. These datasets represent diverse sensor attributes such as air temperature (°C), barometric pressure (hPa), relative humidity (%), wind speed (m/s), and solar radiation (W/m²). Each dataset was structured as a time-series stream containing a timestamp column and multiple numeric attributes, stored in CSV or Excel (.xls/.xlsx) formats. The datasets span different time intervals, sampling rates, and data densities to comprehensively test the adaptability of the data-centric methodology. On average, each dataset contained approximately 28,000–30,000 rows and covered a temporal range of several months, with varying degrees of missingness, duplicates, and outliers. This heterogeneity was intentionally preserved to assess the robustness of the framework across multiple data-quality conditions and to confirm its generalization capability to unseen sensor streams.

4.4.2 Concept Solution Description

The suggested experimental setup employs a data-centric solution flow, with each dataset going through a predetermined series of quality-oriented activities before being approved as a high-quality (HQ) air-quality dataset. The Air Quality Index (AQI) dataset (Input) serves as the starting point. In practice, this dataset may originate from several sensors or external sources, and hence may not always be in the exact structure required by the SmartCityCloud environment. As a result, the first stage is import preparation, which includes adding or normalizing a timestamp column to ensure that each record is identified in time. This is critical since all subsequent operations—

resampling, imputation, outlier detection, label checks, and OOD generalization—are time-series operations that necessitate a proper temporal index.

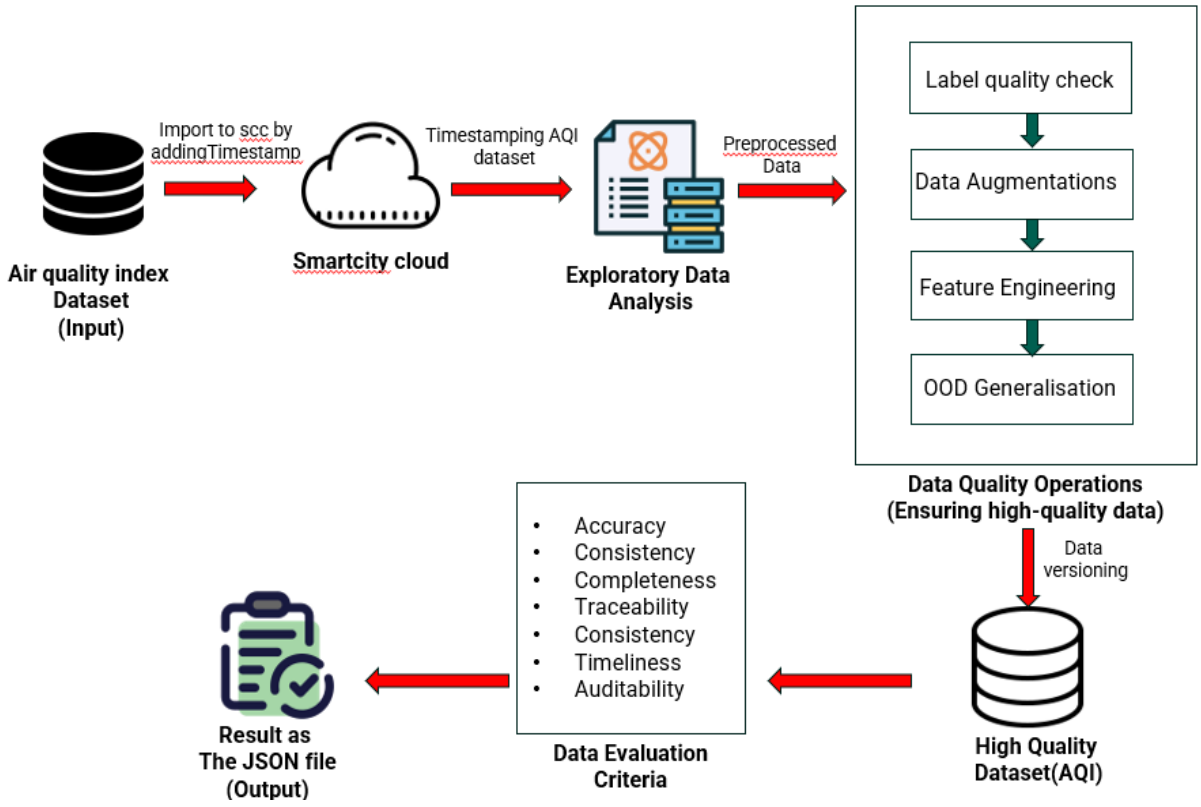


Figure 10. Concept Solution Diagram

Following this preparation, the dataset is uploaded or linked to the SmartCityCloud (SCC), which serves as the primary data repository. SCC is responsible for storing various sensor streams in a consistent, time-indexed fashion. At this point, the AQI dataset is integrated into the same context as other smart-city data, allowing for comparison, resampling, and running the same task wrapper on other attributes. The Fig 10 also mentions that SCC may "timestamp the AQI dataset," — which means it can enhance or regularize the time column if the original source has irregular or missing timestamps, assuring system interoperability.

The subsequent key block is Exploratory Data Analysis (EDA) Fig 11. During this stage, the implementation performs the descriptive and structural checks you previously created: counting missing values, detecting duplicate timestamps, displaying basic statistics, visualizing trends and boxplots, and identifying potentially invalid ranges for attributes such as temperature, pressure, or humidity. EDA is used not just for human inspection, but also to generate preprocessed data, which is then

input into quality procedures. This is where you use the options set earlier in the process (resample frequency, z-threshold, imputation method). This stage produces a cleaned, temporarily ordered AQI series that is ready for more rigorous quality enforcement.

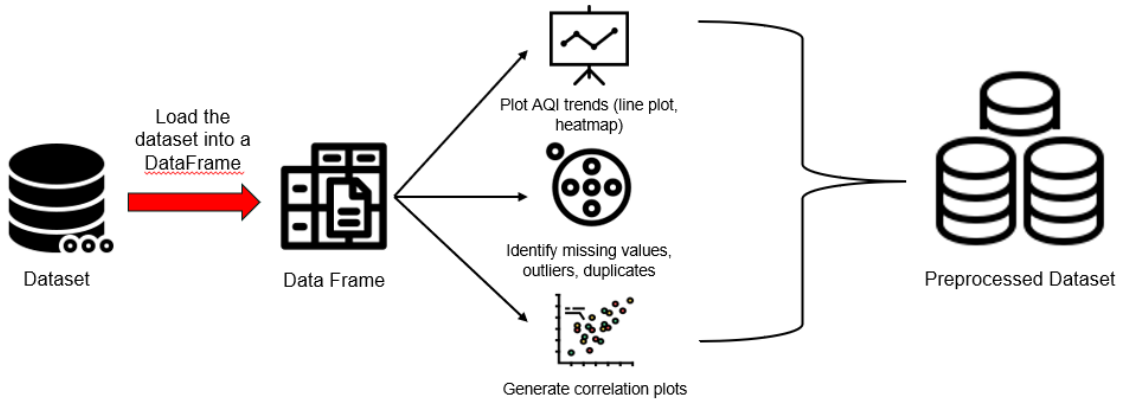


Figure 11. Exploratory Data Analysis for AQI Data

The subsequent step post the EDA is “Data Quality Operations (Ensuring high-quality data)” Fig 12. This is the core of your thesis contribution and consists of four logical stages:

- Label Quality Check: Ensures correctness of labels such as sensor-based conditions (e.g., SRAD > 0), preventing analytical bias from mislabeled records.
- Data Augmentation: Introduces controlled noise or artificial missingness to test the robustness and generalizability of the pipeline across diverse datasets.
- Feature Engineering: Generates temporal descriptors such as rolling mean, rolling standard deviation, and first differences to capture trends in AQI data.
- OOD Generalisation: Splits the series into base and future segments and compares their statistical profiles to detect distribution drift in sensor data.

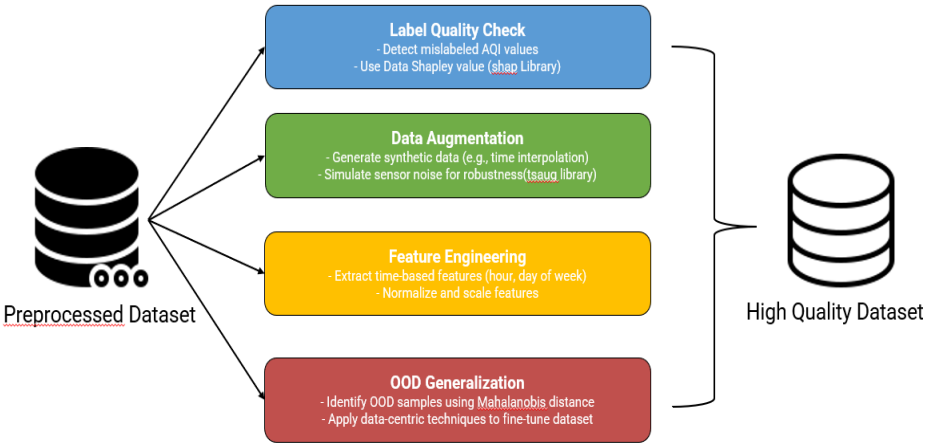


Figure 12. Data Quality Operations

Once these operations are complete, the resulting dataset is written as a High-Quality Dataset (AQI). The data quality operations diagram, which is important: every time the task is run with a different option set, a separate HQ version is stored (usually with a timestamped filename). This enables reproducibility and rollback — a key requirement . In parallel, the pipeline evaluates the dataset against the Data Evaluation Criteria you defined earlier (accuracy, consistency, completeness, traceability, timeliness, auditability). These criteria are computed from the counts collected during processing (missing replaced, outliers flagged, duplicates removed, OOD flags, etc.). Finally, the system produces the JSON file (Output). This file is the machine-readable report containing: the options used in the run, the quality metrics for the six dimensions, any warnings (e.g. “high missingness in February”), and references to the HQ dataset that was saved. This JSON is what the UI can display in your evaluation dashboard and what you can later include in Chapter 6 for results. In this way, the concept solution diagram shows a closed loop: raw AQI → SCC → analysis → quality enforcement → HQ dataset → JSON report — fully aligned with the data-centric, reproducible methodology defined in Chapter 4.

4.5 Assumptions, Limitations, and Summary

This section describes the underlying assumptions, methodological restrictions, and validation metrics used in the design and evaluation of the proposed data-quality framework. Every data-centric methodology is based on some simplifying assumptions about sensor behavior, data distribution, and temporal stability, which are required for formalization but may affect generalizability. The identified constraints emphasize potential sources of uncertainty, such as heuristic parameter sets, dataset reliance, and imputation bias, which can all have an impact on results interpretation. Furthermore, mitigation measures and sensitivity assessments are described to guarantee that these constraints are addressed consistently and their consequences are minimized. The section concludes with a summary that links the methodological framework presented in this chapter to the practical realization described in Chapter 5 and the empirical evaluation presented in Chapter 6, thereby completing the bridge between conceptual design, system implementation, and quantitative assessment.

4.5.1 Assumptions

The proposed methodology is developed under a set of foundational assumptions that ensure the stability and interpretability of the data-quality evaluation process. It is assumed that the sensor metadata provided by the SmartCityCloud environment,

including timestamps, units of measurement, and attribute labels, is accurate and reliable, allowing for consistent parsing and identification of variables. Furthermore, the validity bounds defined for each attribute (such as temperature, pressure, or humidity) are considered to be approximate yet representative of realistic environmental conditions. These bounds are derived from empirical studies and literature but may not perfectly capture local or seasonal variations. Finally, the approach presumes stationarity within the baseline window used for out-of-distribution (OOD) stability checks—that is, the statistical properties of the reference data segment (mean and variance) remain relatively constant over the observed period. This assumption enables meaningful comparison between baseline and future windows, forming the basis for detecting drift or instability in long-term sensor performance.

4.5.2 Limitations and Chapter Summary

While the proposed methodology provides a structured and automated framework for smart-city data-quality assessment, several limitations and validity threats must be acknowledged. The first limitation arises from the use of heuristic validity bounds, which, although empirically grounded, may not universally represent all sensor operating environments. This introduces potential bias when attributes deviate from expected physical ranges due to local calibration differences or extreme environmental conditions. Similarly, the Z-score-based outlier detection approach exhibits sensitivity to the underlying data distribution; highly skewed or non-Gaussian variables may yield false outlier flags or overlook subtle anomalies. The imputation methods (forward fill, backward fill, and linear interpolation) also introduce bias when missing values span large gaps or when the signal exhibits nonlinear dynamics. Moreover, the evaluation outcomes depend partly on dataset specificity—that is, the heterogeneity and volume of the sensor streams used for testing—which may affect the generalizability of the reported results to other cities or sensor infrastructures.

To address these challenges, several mitigation and sensitivity measures are incorporated into the experimental design. Parameter option sweeps are conducted to evaluate the influence of varying thresholds, resampling frequencies, and imputation strategies, ensuring that conclusions are not dependent on a single configuration. Alternative statistical thresholds and adaptive methods are compared to assess the stability of quality metrics across different parameter settings. In addition, manual spot audits—involving direct inspection of selected datasets and their visual summaries—are performed to verify the correctness of automated decisions, particularly in outlier and imputation validation. Together, these measures strengthen the robustness and

reliability of the findings. This section also concludes the methodology chapter by establishing continuity with the subsequent parts of the thesis: Chapter 5 (Implementation) details how the defined processes and algorithms are realized in software. In contrast, Chapter 6 (Results and Evaluation) presents the empirical performance of the framework across diverse datasets. Collectively, these chapters transform the conceptual and formal models introduced here into practical outcomes, completing the transition from theoretical design to experimental validation.

4.6 Summary

Overall, Chapter 4 presented the methodological framework that operationalizes the thesis's data-centric quality engineering approach by defining a systematic, reproducible, and transparent workflow for preparing heterogeneous smart-city time-series data. The chapter outlined the conceptual rationale for the methodology, explaining how design choices were informed by challenges identified in the literature—namely, temporal heterogeneity, missingness, outlier behavior, and label fragility. It then detailed the end-to-end system architecture, covering data ingestion, stream typing, timestamp normalization, and the overall workflow required to transform raw sensor datasets into analysis-ready inputs. The formal processing pipeline was introduced, defining each operation—validity screening, missing-data treatment, Z-score outlier detection, rolling statistics, feature engineering, resampling, and HQ dataset construction—as structured, parametrizable procedures that ensure consistent and explainable transformations across datasets. The chapter further described the experimental setup, including parameter configurations, concept-solution logic, and the generation of artifacts such as HQ datasets and JSON provenance reports that support auditability and reproducibility. Finally, the assumptions and limitations underlying the methodology were acknowledged, along with mitigation measures such as parameter sweeps, sensitivity analyses, and manual spot audits, thereby positioning the methodology as a scientifically grounded bridge between the foundational concepts and the implementation and evaluation presented in Chapters 5 and 6

5 Implementation

This chapter describes the practical application of the concepts, data pipelines, and design principles introduced in earlier chapters. Building on the methodological foundation presented in Chapter 4, the implementation transforms the suggested data-centric pipeline into a fully functioning system within the SmartCityCloud (SCC) environment. While previous sections highlighted the limitations of existing model-centric approaches and the importance of high-quality, traceable sensor data, this chapter shows how those theoretical foundations are realized through code, modular architecture, and automated data-quality evaluation mechanisms. The implementation details include SmartCity Compute Task Wrapper configuration, environment setup, integration of local development with SCC's cloud infrastructure, and the implementation of each functional component—from data ingestion and exploratory analysis to validation and high-quality (HQ) dataset generation. This section bridges methodological design and execution, providing a comprehensive view of how the proposed system addresses the missingness, inconsistency, and traceability challenges identified in the state-of-the-art review, thereby establishing a reproducible foundation for the Results and Evaluation chapter.

5.1 SmartCityCloud Context and Data Sources

This section describes the technology and data foundations for the proposed implementation. It describes the SmartCityCloud (SCC) platform, which serves as the underlying cloud infrastructure for large-scale management, processing, and analysis of various sensor data streams. The debate focuses on how environmental and urban sensors generate data, notably air-quality information, which is then stored and transmitted in common forms such as CSV. This section also describes the structure and semantics of the Air Quality Index (AQI) dataset used in this thesis, including its properties, temporal characteristics, and data-generation workflow inside the SCC ecosystem. By creating this backdrop, the section gives the necessary understanding of the platform architecture and sensor data flow, on which the future compute-task implementation is based.

5.1.1 SmartCityCloud Platform and Sensor Data Generation

The SmartCityCloud (SCC) is a modular, cloud-based platform for managing, processing, and evaluating sensor data from a variety of smart city domains. It offers an integrated platform that enables real-time and batch analytics, allowing for scalable

data management in urban applications including traffic monitoring, forest inspection, environmental evaluation, parking management, and drone-based surveillance. The platform's architecture is layered, with layers for data intake, processing, storage, and visualization that work together to ensure that data streams from diverse sources are processed equally. As described in Chapter 4, compute tasks are deployed using the SCC's Compute Task Wrapper, which encapsulates the execution environment and allows users to add custom AI or data-quality modules without affecting the underlying infrastructure. This modularity enhances interoperability and facilitates the rapid prototyping of analytical solutions for various urban scenarios.

The creation of sensor data is critical to this architecture. Sensors located across the city continuously record environmental data such as air temperature, humidity, wind speed, sun radiation, barometric pressure, and rainfall. These measurements are sent to the cloud in organized tabular format—typically as comma-separated values (CSV) files or live data streams—with each record labelled with a timestamp and, in some circumstances, a geographical identifier. The CSV format is a lightweight and consistent way to describe heterogeneous sensor outputs, ensuring interoperability with both local compute environments and SCC ingestion interfaces. Each dataset follows a consistent schema: a timestamp column indicating the measurement time, followed by attribute columns representing sensor readings with associated physical units (e.g., °C for air temperature, % for humidity, m/s for wind speed). The SCC ingestion layer validates the structural integrity of these files, detects missing or duplicated entries, and stores them in cloud-based repositories for subsequent analysis.

The dataset employed in this thesis, the Air Quality Index (AQI), is an example of such data production. It collects continuous air-quality readings from scattered sensors and uses important environmental indicators to assess atmospheric conditions. This dataset, stored in CSV format, serves as the experimental foundation for testing the data-quality measures and analytical methods described later in this chapter. The SmartCityCloud platform and its standardized sensor data pipelines create a solid foundation for executing compute activities and verifying data-centric AI approaches that aim to improve data reliability, traceability, and overall quality in smart-city ecosystems.

5.1.2 AQI Dataset: Structure & CSV Layout

The Air Quality Index (AQI) dataset is the key data source for assessing the proposed data-quality methodology in the SmartCityCloud environment. It is a time-series sensor dataset compiled by many environmental monitoring devices spread throughout the urban network. Each record refers to an instantaneous measurement taken at a specified timestamp, representing atmospheric and meteorological variables that together characterize local air quality conditions. The dataset is saved in structured comma-separated values (CSV) format, making it simple to import, preprocess, and validate within the SmartCity Compute Task Wrapper.

The dataset used in this thesis comprises approximately 28,448 rows and a fixed set of sensor-based attributes. Table 4 summarizes the main columns, their physical meanings, and measurement units.

Attribute	Description	Unit	Typical Range
CollectedDateAt	Timestamp of data collection (synchronized to sensor clock)	–	ISO 8601 datetime
AirTemperature	Ambient air temperature measured near the surface	°C	–20 to 50
Humidity	Relative humidity in the atmosphere	%	0 to 100
WindSpeed	Instantaneous wind speed	m/s	0 to 60
WindDirection	Direction of wind flow measured clockwise from north	°	0 to 360
SRAD	Solar radiation intensity	W/m²	0 to 1200
BarometricPressure	Atmospheric pressure at ground level	hPa	870 to 1080
Rain	Daily rainfall accumulation	mm/day	0 to 500
Flag (optional)	Quality indicator for flagged or missing records	–	0 = valid, 1 = flagged

Table 4. AQI Dataset Attributes

Each attribute represents a continuous numeric stream sampled at regular intervals. In the provided dataset, the sampling cadence corresponds approximately to 10-minute intervals, enabling both fine-grained temporal analysis and monthly aggregation for detecting long-term trends. The dataset includes occasional missing values, duplicates, and anomalous readings, which are intentionally retained to evaluate the system's ability to perform data cleaning, outlier detection, and imputation. The presence of these real-world irregularities ensures that the proposed data-centric

quality evaluation methods—such as z-score–based outlier identification, rolling mean smoothing, and resampled aggregations—can be rigorously validated.

A typical CSV layout of the dataset is shown below in Table 5 (values redacted for privacy and readability):

Collected DateAt	AirTemperature	Humidity	WindSpeed	WindDirection	SRAD	Barometric Pressure	Rain
2023-01-01 00:00:00	6.171400796	46.320	1.750671397	111.5407247	19.86856612	1019.443503	0
2023-01-01 00:30:00	4.714289308	49.22860209	3.162389981	357.9361896	0	1018.402308	0
2023-01-01 01:00:00	6.226648054	48.23269796	4.543312422	137.9145006	25.90754152	1017.604838	0
2023-01-01 01:30:00	6.615140918	49.449152	6.236840863	136.310372	60.92119426	1018.581297	0
2023-01-01 02:00:00	7.717235023	56.43089952	1.93275409	322.869064	0	1017.695959	0

Table 5. Sample records from AQI Datasets

This structured format ensures seamless integration with the SmartCityCloud ingestion module, where each column is automatically detected as a separate data stream and analyzed within the Exploratory Data Analysis (EDA) and Data Quality modules implemented in this work. The AQI dataset thus provides a representative and challenging basis for testing the robustness of the proposed cloud-based, data-centric quality assurance framework.

5.1.3 Data Ingestion into SmartCityCloud

The data ingestion pipeline describes how sensor data files are introduced, registered, and prepared for analysis in the SmartCityCloud (SCC) environment. This method serves as the first step in the cloud's data pipeline, ensuring that incoming datasets are standardized, version-controlled, and easily accessible for compute-task execution. The pipeline starts with data collection from field-deployed sensors, which send raw readings in the form of CSV or Excel files with timestamped environmental measurements such air temperature, humidity, wind factors, and sun radiation. These

files are then loaded into the SCC platform via a regulated ingestion interface, which validates the structure and information before further processing.

In this implementation, the data ingestion process is integrated into a user-friendly graphical interface (UI) that enables direct uploading of CSV or Excel files through the SmartCityCloud Compute Task platform Fig 13. Upon upload, the system automatically identifies the file type and processes it using the corresponding data reader module—CsvStreamReader for comma-separated files or ExcelStreamReader for spreadsheets. Each dataset is internally decomposed into multiple data streams, with each stream representing a single sensor attribute such as air temperature, humidity, or solar radiation. The ingestion layer further performs automated timestamp detection and data-type assignment to maintain consistency. All datasets follow a structured naming format (e.g., AQI/AirTemperature) and include version tracking to ensure reproducibility and auditability across evaluations. This streamlined workflow forms a crucial bridge between raw sensor inputs and SmartCityCloud's computational environment, enabling a seamless transition from data acquisition to standardized analytical processing within the Compute Task Wrapper.

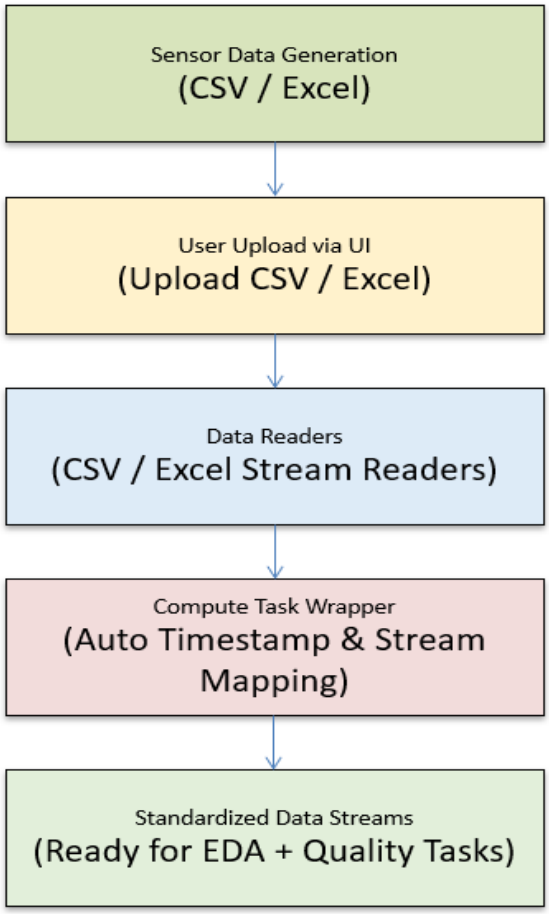


Figure 13. Data Ingestion Workflow

5.2 SmartCity Compute Task Wrapper

This section introduces the SmartCity Compute Task Wrapper, the main execution framework that enables the modular and flexible implementation of analytical activities on the SmartCityCloud platform. The wrapper serves as an abstraction layer between raw sensor data and computational logic, allowing researchers and developers to incorporate new data-centric or AI-driven workflows without affecting the cloud architecture. It standardizes critical processes, including data loading, validation, transformation, and result export, ensuring interoperability and reproducibility across several smart-city applications. As part of this thesis, the Compute Task Wrapper was improved and expanded to allow data-quality operations, Exploratory Data Analysis (EDA), and high-quality (HQ) dataset production. This contribution not only improves the system's scalability and maintainability but also demonstrates how a unified compute framework can facilitate AI task execution on heterogeneous urban datasets, bridging the gap between theoretical design and practical deployment within SmartCityCloud.

5.2.1 Role as a Common Execution Platform

The SmartCity Compute Task Wrapper acts as a unified execution platform that simplifies and standardizes the integration of analytical tasks within the SmartCityCloud (SCC) ecosystem. As sensor data in smart-city environments originates from heterogeneous sources with varying formats and sampling rates, direct algorithm implementation becomes complex and inconsistent. The wrapper resolves this by offering a standardized interface that abstracts low-level data handling, enabling developers to focus on analytical logic. Its modular structure converts key operations—such as input discovery, stream parsing, option setup, task execution, and output generation—into reusable components, supporting plug-and-play development of new modules like data-quality analysis or anomaly detection without altering the core infrastructure. By managing task lifecycles and enforcing a consistent input–output structure, the framework ensures interoperability and reproducibility across datasets and projects. Overall, it forms the foundation of the SCC analytical layer, enabling scalable, maintainable, and reliable deployment of AI-driven and data-centric applications in smart-city contexts.

5.2.2 Wrapper Architecture and Extensibility Model

The SmartCity Compute Task Wrapper is the foundational architectural component that integrates data ingestion, processing, visualization, and output generation in the

SmartCityCloud (SCC) framework. It captures the technological difficulty of managing heterogeneous smart-city sensor data through a layered and extensible framework. As depicted in Fig 14, the architecture is made up of four interconnected layers: the User Interface Layer, the Compute Task Layer, the Data Stream Layer, and the Data Storage Layer. These layers constitute a standardized execution pipeline that allows users to run analytical or AI-based operations with little configuration work while ensuring reproducibility, maintainability, and interoperability across datasets and projects.

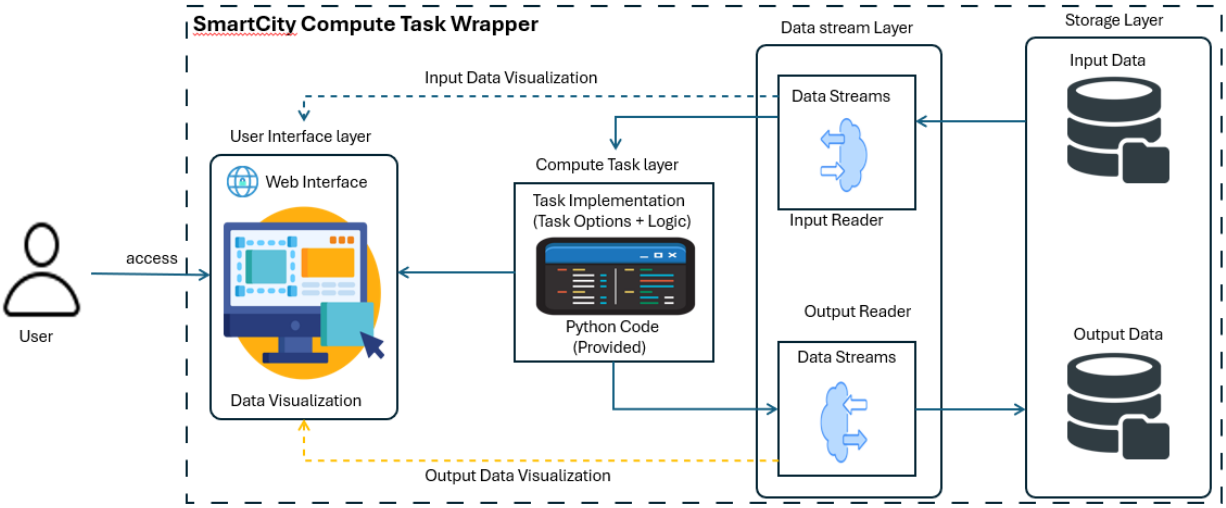


Figure 14. SCC Compute Task Wrapper Architecture

a) **User Interface Layer** - The User Interface Layer represents the topmost abstraction through which users interact with the SmartCityCloud platform. It is implemented using the NiceGUI framework in the show_ui.py module, which automatically generates a responsive web interface. This interface allows users to log in, upload input datasets (in CSV or Excel format), configure task parameters, and visualize the results of the computation. The UI communicates directly with the Compute Task Wrapper through the AutoTaskRunnerUI class, dynamically loading available tasks and their configurable options.

When a user uploads a dataset, the interface immediately triggers the ingestion process and displays input-data previews and configuration panels. After execution, the results—such as statistical summaries, visual plots, or high-quality (HQ) dataset exports—are rendered back in the UI as part of the visualization dashboard Fig 15. This design ensures that even non-technical users can interact with the analytical pipeline without needing to modify the

codebase, establishing an accessible yet controlled environment for urban data analysis.

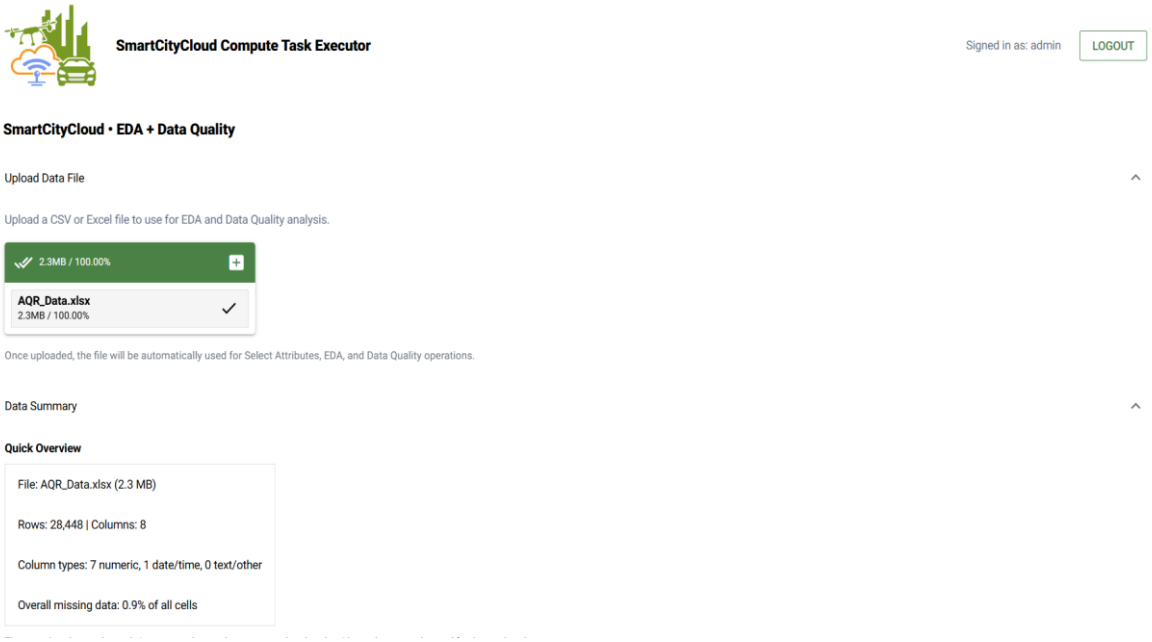


Figure 15. Smartcity Cloud User Interface

b) **Compute Task Layer -** The Compute Task Layer is the computational backbone of the SmartCity Compute Task Wrapper. It manages the entire job lifecycle in four standardized stages: discover, options, process, and write, ensuring a consistent and reproducible workflow across all analytical modules. During the discover phase, the wrapper automatically analyzes accessible datasets in the inputs/ directory and recognizes them as data streams for analysis. The options stage then exposes changeable parameters defined in task_impl.py's get_default_options() method, allowing for dynamic task customization prior to execution. In the process phase, the core analytical logic executes operations such as missing-value detection, z-score–based outlier identification, rolling mean smoothing, or feature generation, depending on the task type. Finally, the write phase serializes the processed results through standardized stream writers, ensuring consistent formatting for visualization and storage. This lifecycle enforces a strict input–output contract that allows tasks to operate independently while maintaining interoperability with other SmartCityCloud components.

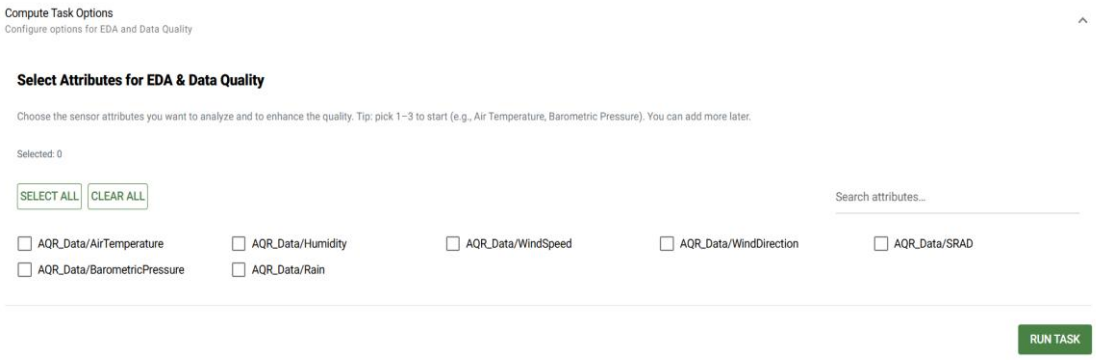


Figure 16. Compute Task Options Samples

The graphical interface for configuring task parameters—illustrated in Fig 16 is automatically generated by the system using the Compute Task Options framework. Each task option defined in the code (e.g., numeric sliders, dropdown selections, or Boolean toggles) is dynamically translated into an interactive widget within the user interface. This design provides an intuitive bridge between the user and the underlying Python implementation, enabling users to control algorithmic parameters such as detection sensitivity, threshold levels, or choice of anomaly detection method without modifying the source code. The figure demonstrates a typical configuration panel, where options for selecting the numeric stream attributes that need to be analysed with respect to the timestamp attribute. Such modular option handling not only enhances flexibility and usability but also ensures that the same computational logic can be applied to varied datasets or use cases with minimal configuration. This adaptability exemplifies the extensibility of the Compute Task Layer and its role in enabling user-driven experimentation and reproducible AI workflows within the SmartCityCloud environment.

c) **Data Stream Layer -** The Data Stream Layer acts as the intermediary between computation and storage, transforming raw datasets into structured, streamable objects. It handles data flow, type inference, and conversion across multiple file formats. The implementation utilizes specialized stream classes such as CsvStreamReader, ExcelStreamReader, and ImageStreamReader, each responsible for parsing a specific data type and converting it into unified DataStream objects.

Once ingested, each attribute in the dataset (e.g., AirTemperature, Humidity, SRAD) is treated as an independent data stream. The StreamReader and StreamWriter interfaces define how these streams are read and written,

supporting both real-time and batch modes. This abstraction enables low-latency data access and pipeline flexibility, ensuring that analytical modules can handle continuous or discrete inputs without additional transformation. The layer thus provides an essential bridge between physical data representation and the logical processing model used by compute tasks.

d) **Data Storage Layer -** The Data Storage Layer manages persistent input and output datasets within the SCC system. It stores raw data files, processed results, and high-quality (HQ) outputs produced after cleaning, imputation, and augmentation. The storage layer ensures that each dataset is versioned and traceable, allowing experiments to be reproduced consistently. Input datasets are typically placed in the /inputs folder, while output artifacts—such as processed CSV/Excel files, JSON reports, or visualizations—are written automatically to the /outputs directory through the wrapper's stream writers.

   In the current local implementation, the layer relies on filesystem storage but maintains a structure that can be easily extended to cloud databases or distributed storage systems. Each stored file retains a metadata signature containing dataset name, timestamp, and attribute identifiers, enabling efficient retrieval during subsequent analysis or evaluation.

The interaction among the four layers of the SmartCity Compute Task Wrapper follows a top-down execution flow that ensures smooth data movement from ingestion to visualization. The User Interface Layer initiates the workflow when a user uploads a dataset and selects a task. The Compute Task Layer then retrieves configuration options and executes the analytical logic, while the Data Stream Layer manages data flow between input readers, processors, and output writers to maintain consistency and synchronization. Finally, the Data Storage Layer saves the processed outputs, which are sent back to the interface for visualization and interpretation. This standardized lifecycle—discover → configure → process → visualize → store—ensures uniformity, reproducibility, and reliability across all analytical tasks within SmartCityCloud.

A key contribution of this thesis is the extension of the wrapper to support data-quality-centric compute tasks, enabling advanced operations such as missing-value detection, outlier analysis, imputation, and high-quality dataset generation. These enhancements adhere to the same base interfaces (TaskRunner, DataStream, and ComputeTaskOption), ensuring seamless compatibility with existing components. The

architecture remains fully extensible, allowing future integration of AI-driven models like predictive air-quality forecasting or anomaly detection without structural changes. This modular, plug-and-play design establishes the Compute Task Wrapper as a scalable and reusable analytical framework for both experimental research and large-scale smart-city applications.

5.3 Local Environment Setup

This section describes the complete setup process required to replicate and execute the developed SmartCityCloud (SCC) Compute Task Wrapper in a local computing environment. Establishing a consistent and reproducible setup is essential for ensuring that the implementation can be deployed seamlessly across different systems and development platforms. The section outlines the step-by-step procedure for cloning the project repository from GitLab, installing required dependencies and toolchains, and configuring the working environment using Python and Conda. It also covers the procedures for connecting the local workspace to the GitLab remote repository to facilitate version control, collaborative development, and continuous integration. Finally, reproducibility practices—such as environment pinning, version locking, and consistent seed initialization—are discussed to guarantee that all experiments and executions can be reliably reproduced under identical conditions.

5.3.1 Cloning from GitLab and Repository Layout

The implementation of the SmartCityCloud (SCC) Compute Task Wrapper began by cloning the official template repository from the TU Chemnitz GitLab server into a dedicated working directory on the local machine. A separate folder named SmartCityCloud-template was created to maintain an isolated environment for development and experimentation. Using Git, the repository was cloned from the university's remote instance via the following command executed in the terminal

```
# Clone repository from TU Chemnitz GitLab
git clone https://gitlab.hrz.tu-chemnitz.de/smartcitycloud/smartcitycloud-
template.git
```

This operation downloaded the entire SCC Compute Task Wrapper source code, including all submodules and configuration files. After cloning, a Python 3.11 Conda environment was created and activated to ensure a clean and reproducible

workspace for running the project. The environment was set up using the following commands:

```
conda create --name SmartCityCloud python=3.11
conda activate SmartCityCloud
```

The cloned repository followed a well-defined folder structure designed for modular development and task execution. A simplified overview of the repository layout is shown below:

```
SmartCityCloud/
│
├── app/                  → Core application codebase
│   ├── show_ui.py        → Launches the SmartCityCloud user interface
│   ├── task_impl.py      → User-defined compute task implementation
│   ├── auto_ui/          → Auto-generated UI components and visualization
│   ├── compute/          → Core compute engine, task runner, and options
│   ├── streams/          → Data stream handling modules (CSV, Excel, Image)
│   └── storage/          → File readers/writers for input and output streams
│
├── inputs/               → Folder for user-uploaded datasets (CSV/Excel)
├── outputs/              → Folder for processed results and exported reports
├── requirements.txt      → List of dependencies for environment setup
└── README.md             → Documentation and usage instructions
```

Once the repository was cloned and dependencies were installed, a sample “Hello World” program was executed to verify successful setup and connectivity. The task_impl.py file was modified to print a simple message within the UI framework, confirming that the Compute Task Wrapper, the NiceGUI interface, and the local environment were functioning correctly. The test output displayed “Hello, SmartCityCloud!” in the browser interface, indicating that the cloning and configuration were completed successfully and the local SCC environment was fully operational for further implementation work.

5.3.2 Dependency Installation and Environment Configuration

The development and execution of the SmartCityCloud (SCC) Compute Task Wrapper required a consistent software toolchain capable of supporting asynchronous web frameworks, data-stream processing, and user-interface rendering. To ensure

reproducibility and cross-platform compatibility, a dedicated Conda environment was configured using Python 3.11, serving as the foundation for all compute and visualization tasks. This environment guarantees that the same dependency versions are preserved throughout testing, deployment, and evaluation stages. These libraries support the compute framework, user interface, and visualization modules. The dependencies can be grouped as follows:

- **Data Processing and Analytics**: pandas, numpy, scipy, and pytz for time-series manipulation, numerical operations, and statistical computations.
- **Visualization and Plotting:** matplotlib, contourpy, and fonttools to generate plots and data-quality dashboards.
- **User Interface and Frontend Rendering**: nicegui (v2.10.1), jinja2, and markdown2 for automatic UI generation, input selection, and display of outputs via web interface.
- **Backend Services and Communication**: fastapi, uvicorn, starlette, and httpx for REST-based service communication and local hosting.
- **File and Stream Handling**: openpyxl and aiofiles for handling Excel and asynchronous file I/O.
- **Configuration and Environment Management**: python-dotenv for secure loading of environment variables and system-level parameters.
- **Utility and OS Integration**: colorama, winshell, and ifaddr for Windows shell automation, shortcut creation, and network interface resolution.

The dependency installation was carried out using the following procedure within the Conda environment:

```
# Step 1: Create and activate Conda environment
conda create --name SmartCityCloud python=3.11
conda activate SmartCityCloud

# Step 2: Navigate to the application directory
cd app

# Step 3: Install dependencies from requirements.txt
pip install -r requirements.txt

# Step 4: Verify installation and dependency integrity
python -m pip check
```

The above setup ensures that all libraries—particularly NiceGUI, Matplotlib, and FastAPI—are configured to support interactive visualization, real-time user input, and smooth UI execution. Once installed, the environment can be replicated on any machine using the same requirements.txt file, guaranteeing portability and reproducibility of results. To further standardize execution, a configuration file (.env) was created in the project's root directory. This file defines environment variables that manage user authentication, port configuration, and runtime behaviour. The main parameters are listed below in Table 6. These variables are loaded dynamically through the function _load_users_from_env() in show_ui.py, ensuring secure user access and flexible runtime configuration.

Variable	Purpose
APP_USERNAME / APP_PASSWORD	Default credentials for UI login
APP_USERS	Optional JSON structure for multi-user access
APP_PORT	Defines port for hosting the NiceGUI server (default: 8080)
APP_LOG_LEVEL	Sets verbosity for console logging (INFO/DEBUG)

Table 6. Application Configuration Variables

Upon execution, the system initializes the NiceGUI server and launches a local web instance at http://localhost:8080, displaying the SmartCityCloud login page. After authentication, users can upload datasets, configure options, and execute compute tasks. The environment setup also supports log management and automatic shortcut generation (via WinShell) for ease of access.

To maintain long-term reproducibility, version control was integrated using Git. The following best practices were adopted:

- All dependencies are version-pinned in requirements.txt.
- Commits are regularly synchronized with the TU Chemnitz GitLab repository.
- The Conda environment can be exported using conda env export > environment.yml for archival.
- Random seeds and configuration options are fixed within task modules to ensure consistent evaluation results.

This configuration process establishes a portable and deterministic software foundation for executing SmartCityCloud compute tasks locally. It ensures that all system components—from data ingestion to UI rendering—operate in a synchronized

and reproducible environment, enabling robust experimentation and future scalability to cloud-based deployments.

5.4 Codebase Walk-through

This section provides a detailed overview of the codebase developed and integrated as part of the SmartCityCloud (SCC) Compute Task Wrapper implementation. It explains the functional responsibilities and interactions among the core modules that collectively enable data ingestion, compute-task execution, visualization, and output management. The discussion covers the major components of the system, including the application entry and authentication layer, the auto-generated user interface, the compute layer, and the data streams and storage modules. Each subsection describes the internal logic, data flow, and role of individual Python scripts such as show_ui.py, auto_ui.py, task_impl.py, and the modular packages under compute, streams, and storage. Together, these modules establish the operational backbone of the SCC platform, ensuring modularity, extensibility, and reproducibility in executing AI-driven and data-quality tasks on smart-city sensor data.

5.4.1 Application Entry and Authentication

show_ui.py serves as the application entry point, launching the SmartCityCloud (SCC) UI and enforcing authentication before any compute task can run. At startup, it loads credentials from environment variables (preferably a JSON map via APP_USERS, otherwise APP_USERNAME/APP_PASSWORD), supports hashed secrets, and verifies logins with constant-time comparison to reduce timing-attack risk. Authentication is tracked per client session, so the root route decides at request time whether to render the login form or the main application. The login view provides username/password inputs Fig 17, Enter-to-submit handling, feedback toasts on failure, and a logout action that clears the session and returns users to the login page.

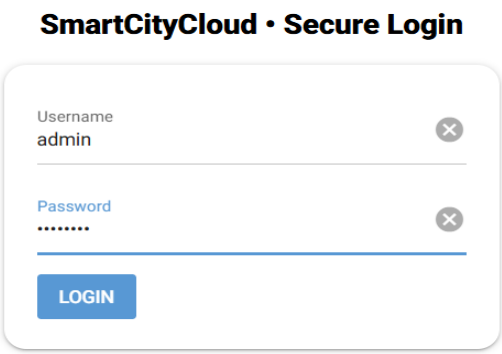


Figure 17 SCC Login Page

After a successful login, the script wires the compute stack into the interface. It constructs a TaskRunner around a DelegateComputeTask that references this thesis's task hooks (TASK_TITLE, get_default_options(), process()), then uses the Auto UI builder to materialize the full configuration and results interface from the option schema—covering input selection, parameter widgets, execution controls, and output/visualization panes—without manual routing. Inputs are discovered on demand (not pre-loaded), which ensures reproducible execution given the same .env, options, and files. The entry module also sets window aesthetics and can create a desktop shortcut for convenience, before launching the app as a native NiceGUI window on the configured port.

5.4.2 UI layer for Tasks

a. **Purpose and role in the system:** The Auto Task Runner UI encapsulated in auto_ui.py provides a declarative, reusable user interface for executing SmartCityCloud compute tasks without hand-coding web forms or plots for each task. It connects the UI to the core execution wrapper (TaskRunner) and renders: (i) an upload/ingestion panel, (ii) task options auto-derived from ComputeTaskOptions, (iii) result visualizations, and (iv) export/evaluation utilities. This design adheres to the guideline's recommendation to document implementation steps and testing artifacts in a reproducible, structured manner (Implementation → Documentation of the Implementation).
b. **Architecture & key components:** auto_ui.py builds on NiceGUI and the wrapper API:
    - **Runner binding:** accepts a TaskRunner instance; all UI actions delegate to runner.discover_inputs(), runner.execute(), and runner.write_outputs() where applicable.
    - **Option rendering:** uses OptionVisualizationUI to transform ComputeTaskOption definitions (e.g., ChoiceOption, InputStreamMultiChoiceOption) into widgets automatically.
    - **Stream visualization:** uses StreamVisualizationUI to produce time-series and summary plots with automatic down-/re-sampling and readable date ticks.
    - **Results model:** stores task outputs in self.outputs and renders grouped "Summary/Quality/Plots/Evaluation" expansions; utilities convert DataStream to native values where needed.

A minimal sketch of the control flow:

```
User uploads file → discover_inputs(upload_dir)
                  → update options (on_inputs_changed)
User clicks "Run Task" → execute(options) → outputs
                  → render summary/quality/plots
                  → optional: save HQ datasets / evaluation
```

c. **Input acquisition & upload workflow:** The UI exposes a file upload that accepts CSV/Excel and persists it to a temporary folder. After saving, it re-discovers input streams using the registered readers (CSV/Excel), refreshes option widgets (so the attribute multi-select reflects the new columns), and builds a “Quick Overview” with rows/columns, type buckets, and per-column missingness/min/max. This gives a layperson-friendly yet audit-ready snapshot before computation. The summary logic detects numeric, datetime, and text columns, computes overall and per-column missingness, and shows min/max for numeric fields—supporting data-quality awareness before running the task.

```
def __on_upload_data(self, e):
    # persist upload → discover_inputs(upload_dir, get_stream_readers())
    self.runner.discover_inputs(self.upload_dir, get_stream_readers())
    self.stream_vis.set_inputs(self.runner.inputs)
    changed = self.__update_options()
    if changed: self.__build_option_group(self.options)
    self.__render_data_summary(target_path)
```

Fig 18 illustrates the automated data ingestion and initial validation workflow generated by the AutoTaskRunnerUI. Once the user uploads a dataset (here: AQI_Data.xlsx), the system immediately analyzes the file and produces a structured “Quick Overview” summarizing key metadata, including file size, row and column counts, detected data types, and overall missing-data proportion. Below this summary, the interface provides a detailed per-column breakdown showing the inferred type (numeric or datetime), the percentage of missing values, example entries, and the minimum–maximum ranges. This automatic inspection step enables users to verify dataset integrity, understand variable characteristics, and ensure suitability for downstream Exploratory Data Analysis (EDA) and Data Quality operations without requiring any manual preprocessing.

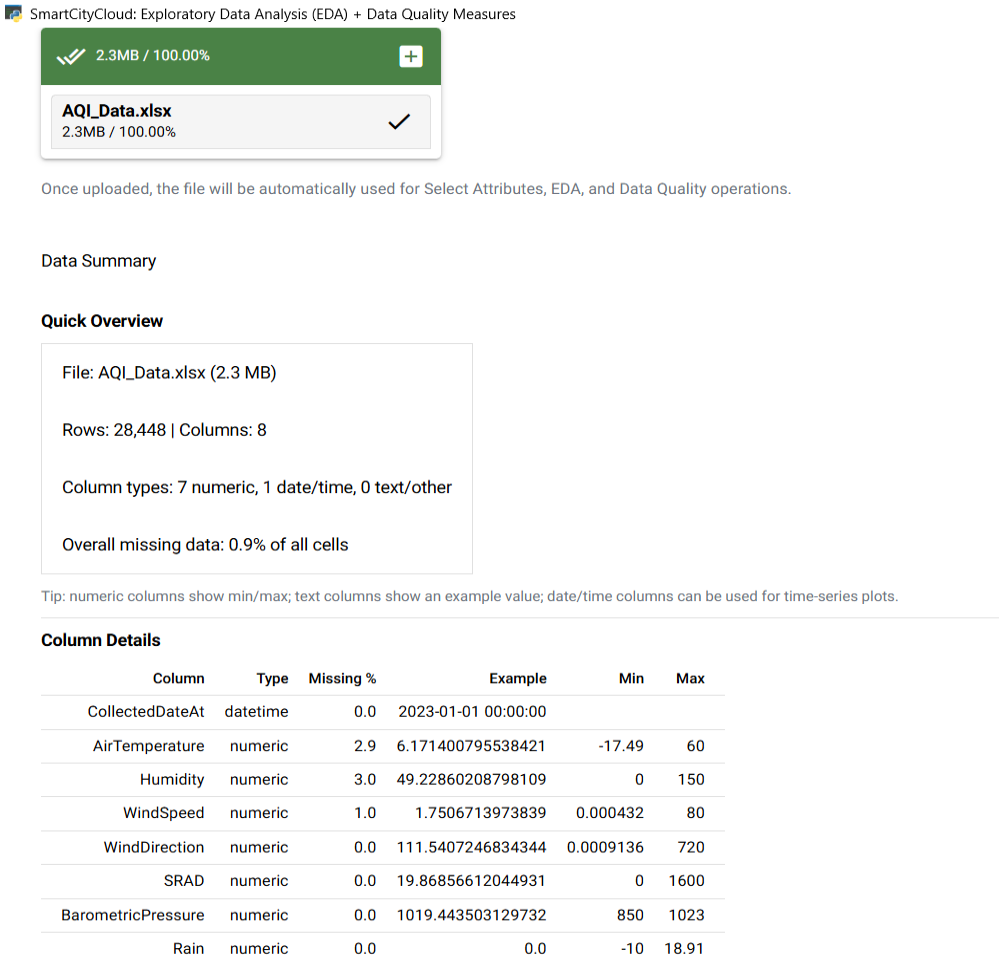


Figure 18. Auto Task Runner UI

d. **Auto-rendered options & execution flow:** Options are not hard-coded in the UI; they're derived from the task's default options and rendered via OptionVisualizationUI. When inputs change, on_inputs_changed in each option (notably InputStreamMultiChoiceOption) repopulates choices to list only valid numeric attributes. The "Run Task" button collects widget values and calls runner.execute. Relevant snippets:

```
def __build_option_group(self, options: dict[str, ComputeTaskOption]):
    with ui.expansion("Compute Task Options", value=True):
        for name, option in options.items():
            self.option_vis.visualize(name, option)
    ui.separator()
    self.ui_start_btn = ui.button("Run Task", on_click=self.process)
```

```
def process(self):
    run_options = {k: v.value for k, v in self.options.items()}
    self.outputs = self.runner.execute(run_options)
    self.__build_output_group()  # renders summary/quality/plots
```

By pushing option semantics into ComputeTaskOption classes (ChoiceOption, InputStreamMultiChoiceOption), the UI layer remains generic and reusable across tasks.

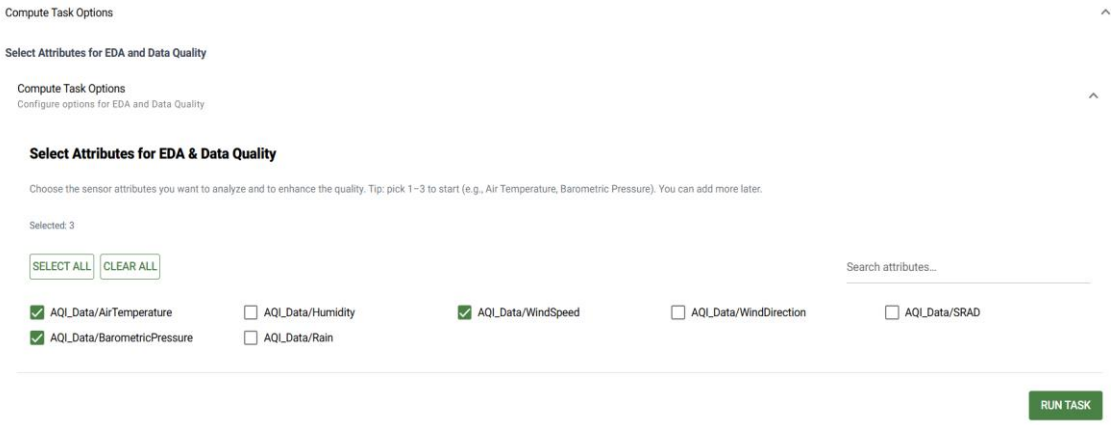


Figure 19. Attribute Selection Interface

Fig 19 shows the automatically generated attribute-selection interface of the AutoTaskRunnerUI. After the dataset is uploaded, the system lists all available sensor attributes as checkboxes, supported by “Select All,” “Clear All,” and a search field for quick filtering. The interface also displays the number of selected attributes and suggests starting with a small subset for initial analysis. Once the user chooses the required variables, clicking *Run Task* triggers the EDA and Data Quality workflow. This component highlights the system’s focus on usability, configurability, and efficient task execution without manual coding.

e. **Results rendering & visualization:** After execution, the UI builds multiple expansions:
    - Summary cards/tables: compact statistics (count, mean, std, min/median/max) per attribute. Values are extracted from DataStream or nested dicts and formatted for readability.
    - Quality sections: per-attribute panels for Missing/Validity/Outliers, including warning thresholds and explanatory notes, which is useful for documenting testing and validation steps as required in the guideline.

- Plots: time-series plots automatically re-sampled (5min→monthly depending on span), auto-formatted date ticks, and gentle rolling means for readability. (Underlying helpers live in StreamVisualizationUI.)

```
for each selected attribute:
    align with detected datetime stream (if lengths match)
    rule = choose_resample_rule(dt_span)
    y = resample(mean) then downsample uniformly to MAX_POINTS_TS
    plot y vs time with ConciseDateFormatter; add rolling mean
```

f. Export & evaluation utilities: The UI implements a one-click export that preferentially writes High-Quality (HQ) tables and a promoted JSON report (with key metrics elevated to top-level keys such as rows_total, missing_total, outliers_z_percent), and it selects Excel if an engine is available, else CSV/JSON. This balances reproducibility (dataset snapshots) and auditability (JSON metrics), matching the guideline's emphasis on documenting implementation artifacts and testing/validation outputs. Example:

```
def save_outputs(self):
    # find artifacts in outputs (hq_rows, json_report)
    # write HighQuality_<dataset>_TS.xlsx (or CSV fallback)
    # write HighQuality_<dataset>_TS.json with promoted report
```

An evaluation tab also accepts a JSON report (from a prior run or external tool) and computes readiness and data-quality scores (Completeness, Validity, Consistency, Stability/Drift, Outliers). The UI explains how scores were derived (e.g., "Missing values handled: x/y"), which is valuable for the Testing and Validation subsection of the Implementation chapter.

5.4.3 Compute Layer

The Compute Layer forms the operational core of the SmartCityCloud framework and defines how analytical tasks are structured, configured, and executed. Its foundation is the ComputeTask abstraction in tasks.py, which specifies the required lifecycle functions: title, get_default_options(), and process(). Each concrete task conforms to this interface, ensuring that all implementations behave consistently regardless of their internal logic. The system uses DelegateComputeTask to bridge user-implemented functions (such as this thesis's EDA and data-quality pipeline) with the unified execution engine. The TaskRunner orchestrates the full workflow by discovering input

streams using registered StreamReader classes, validating names and data types, and passing the loaded streams to the selected compute task. During execution, the runner invokes the task's process() method and then automatically converts primitive results (scalars, lists, numeric arrays) into standardized DataStream objects so downstream components—including the UI and storage subsystems—can treat outputs uniformly. The runner also provides safe write-back functionality through write_outputs(), matching result streams with appropriate writers (e.g., CSV, Excel, or image data streams). This creates a strict contract ensuring that every task moves through a consistent, reproducible cycle: discover → configure → process → materialize outputs.

Task configuration shown in Table 7 relies on the flexible option framework defined in options.py. Core option types such as NumberOption and ChoiceOption allow numeric ranges, dropdowns, and custom selections, while stream-aware options (InputStreamChoiceOption, InputStreamMultiChoiceOption) dynamically adapt to dataset attributes during input discovery. For example, InputStreamMultiChoiceOption automatically filters available columns to include only numeric streams (INT/FLOAT), ensuring that algorithms such as outlier detection or interpolation are only applied to valid attributes. The output.py module complements this by providing the StreamOutputHelper, which creates writable streams for images and leaves extension points for user-defined export formats. The thesis-specific task_impl.py builds on this compute infrastructure to implement a complete data-quality pipeline: missing-value detection, z-score outlier filtering, rolling statistics, interpolation strategies, and generation of high-quality (HQ) datasets. Options defined in get_default_options() (e.g., selected attributes, resampling frequency, z-thresholds, interpolation mode) control the behavior of these algorithms, while the structured outputs—summary statistics, cleaned streams, and promoted JSON quality reports—flow back through TaskRunner for UI rendering and file export. Together, the Compute Layer establishes a modular, extensible, and reproducible execution backend that transforms user configuration into concrete analytical results.

Option	Default	Purpose
stream_numeric_multi	(none)	Select numeric attributes for EDA & quality checks
resample_freq	M	Aggregation frequency (H/D/M)
zscore_threshold	3.0	Outlier detection threshold
clean_invalid	yes	Remove unrealistic values
drop_duplicate_timestamps	yes	Ensure unique timestamps
interpolate_method	none	Missing value handling

rolling_window	15	Rolling statistics window
augment	none	Diagnostic augmentation
ood_split	70/30	Baseline vs OOD split
export_format	both	Export result format

Table 7. Compute Task Options

5.4.4 Streams and Storage

The Streams subsystem provides the unified data abstraction used throughout the SmartCityCloud Compute Task Wrapper. At its core is the DataStream base class, which encapsulates sensor values along with an associated StreamDataType that describes the semantic type of the stream (e.g., INT, FLOAT, STRING, DATETIME, IMAGE_SEQUENCE). All concrete stream types inherit from this abstraction and expose consistent interfaces for retrieving values, counting elements, and expressing whether the stream is writable. The standard in-memory types include InMemoryDataStream for homogeneous numeric/text series, NumpyDataStream for NumPy-backed arrays with automatic dtype inference, and ScalarStream for single-value outputs, such as summary metrics or quality scores. The image subsystem extends the same abstraction: LocalFilesImageDataStream represents lazily loaded images, while WritableImageDataStream provides a structured mechanism for writing generated visual outputs to disk, ensuring that even non-tabular results conform to the same stream interface used throughout the wrapper. These classes collectively ensure that all inputs and outputs—whether numeric time series, scalar indicators, or image sequences—can be processed, visualized, and exported through a common API without special-case handling.

The Storage subsystem complements the streams by providing format-specific readers and writers that convert physical files into typed DataStream objects. As shown in Table 8, CSV, Excel, and image directories are handled by their respective readers (CsvStreamReader, ExcelStreamReader, and ImageStreamReader), each responsible for parsing the raw file, inferring attribute types, and returning a dictionary of stream name → stream object. During input discovery, the TaskRunner iterates through all files in the given folder, queries each available reader via supports_source(), and loads the corresponding streams using read_source() . Naming conventions ensure that each column becomes an addressable stream (e.g., "AQI/AirTemperature"), enabling the auto-UI and compute logic to treat them consistently. Output writing follows a similar model, where the runner matches each

result stream with appropriate stream writers; image outputs use WritableImageDataStream, while tabular results are exported through CSV/Excel writers depending on availability. This structured mapping—from file → stream abstraction → output writer—forms a stable, extensible foundation enabling all modules in the SmartCityCloud framework to interoperate seamlessly with diverse data formats while maintaining reproducibility and a clear separation between data representation and computation.

Source	Stream Type	Usage
CSV	InMemoryDataStream	AQI ingestion; numeric & datetime columns.
Excel	InMemoryDataStream	Alternate ingestion & HQ table export.
Images	ImageDataStream	Supported for sequences; not used here.
Generated plots	WritableImageDataStream	Stores PNG/SVG artifacts.
Numpy arrays	NumpyDataStream	Internal numeric data for calculations.
Scalar outputs	ScalarStream	Single metrics like counts or percentages.

Table 8. Mapping of I/O file formats

5.5 Implementation Steps

After the user selects the AQI dataset and clicks Run Task, the compute task processes all selected numeric attributes and organizes the outputs within three major interface blocks: the Quality section, the Plots section, and the Data Quality section. This structure reflects the complete analytical workflow implemented in task_impl.py and orchestrated by auto_ui.py. The system's behaviour is therefore best understood by following the sequence in which the interface presents results, as each section corresponds to a specific stage of computation within the EDA and data-quality pipeline. Once all analyses are completed, the user may export the high-quality dataset and the JSON-based diagnostic report. The following subsections describe these interface blocks in detail and explain how they map to the underlying code paths.

5.5.1 Quality Section

Immediately after the task execution completes, the interface expands the Quality section, which contains a set of essential diagnostic metrics for each selected attribute. These metrics serve as the first level of verification and correspond to the completeness, validity, and consistency checks defined in Chapter 4. The missing-data component quantifies both the total number and percentage of missing values and identifies the number of months during which missing entries occur. This behaviour is implemented by grouping the raw values at a monthly level and computing missingness statistics before any cleaning or interpolation is applied Fig 20. A summary of these values appears directly in the interface, and a bar-chart visualization highlights the temporal distribution of missing data across months.

Metric	Value
Rows (total)	28448
Missing (count)	818
Missing (%)	2.88%
Months w/ missing	18

Figure 20. Monthly missing percentage chart

The validity-bound analysis evaluates whether sensor readings fall within the plausible physical limits corresponding to the attribute. Using the _guess_validity_bounds() method, the system automatically infers appropriate lower and upper bounds based on the attribute name (for example, air temperature, humidity, wind speed, and solar radiation). Values outside these limits are classified as invalid, and a representative preview of such entries is provided to the user alongside a numerical count and percentage Fig 21. This enables early detection of malfunctioning sensor periods or data-entry errors.

Metric	Value
Lower bound	-20
Upper bound	50
Invalid (count)	14
Invalid (%)	0.05%

Figure 21. Validity Bound Summary with Invalid Samples

Outlier detection is performed using the Z-score method. The standard deviation and mean of the cleaned values are computed, and readings whose absolute Z-score exceeds the user-defined threshold (with 3.0 as the default) are identified. The interface reports the number and proportion of detected outliers and displays a time-series plot that overlays outlier points on top of the smoothed mean curve Fig 22. This visualization allows the user to distinguish isolated anomalies from more persistent deviations in sensor behaviour.

Metric	Value
Z threshold	3
Outliers (count)	15
Outliers (%)	0.05%

Figure 22. Z-score outlier plot with anomaly readings

The system also inspects the dataset for duplicate timestamps. Duplicate records frequently occur when sensors transmit multiple signals within the same time interval Fig 23. If the user has enabled duplicate removal, the system retains only the first entry for each timestamp and reports the number of removed records. Together, these four components—missing values, validity bounds, Z-score outliers, and duplicate timestamps—form a comprehensive first-stage quality assessment that validates the structural integrity of the AQI dataset.

Metric	Value
Duplicate rows	598
Duplicate (%)	2.10%

Figure 23. Duplicate timestamp detection

5.5.2 Plots Section

The Plots section provides the user with a comprehensive set of visualization tools aimed at facilitating exploratory data analysis. These visual outputs are generated directly from the dictionary returned by process() and rendered by AutoTaskRunnerUI through Matplotlib. The system applies adaptive downsampling to ensure responsiveness even for datasets containing several hundred thousand rows.

For datasets with multiple numeric attributes, a correlation matrix is produced to reveal pairwise linear relationships across variables. This matrix is displayed as a heatmap where stronger positive or negative correlations appear as more pronounced color intensities Fig 24. The histogram view complements this by illustrating the distribution of sensor values and providing insight into skewness, heavy tails, or multimodal patterns that may affect downstream modelling tasks.

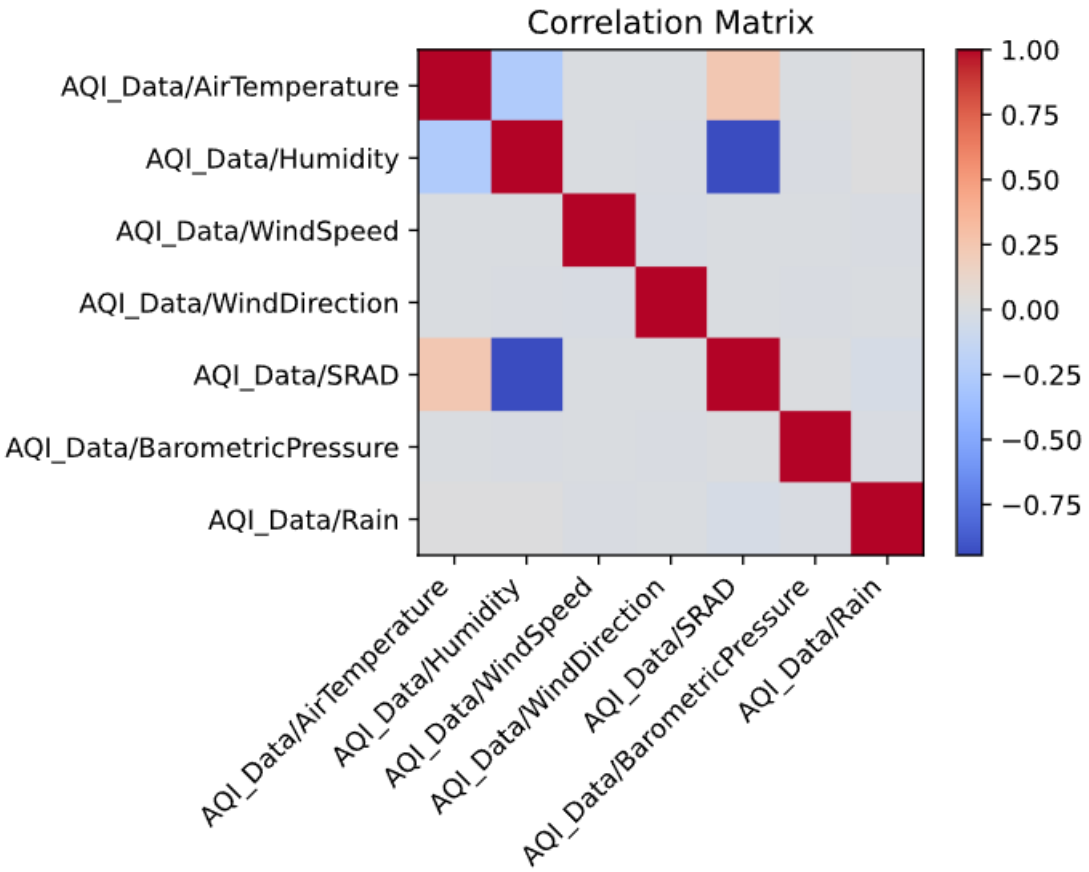
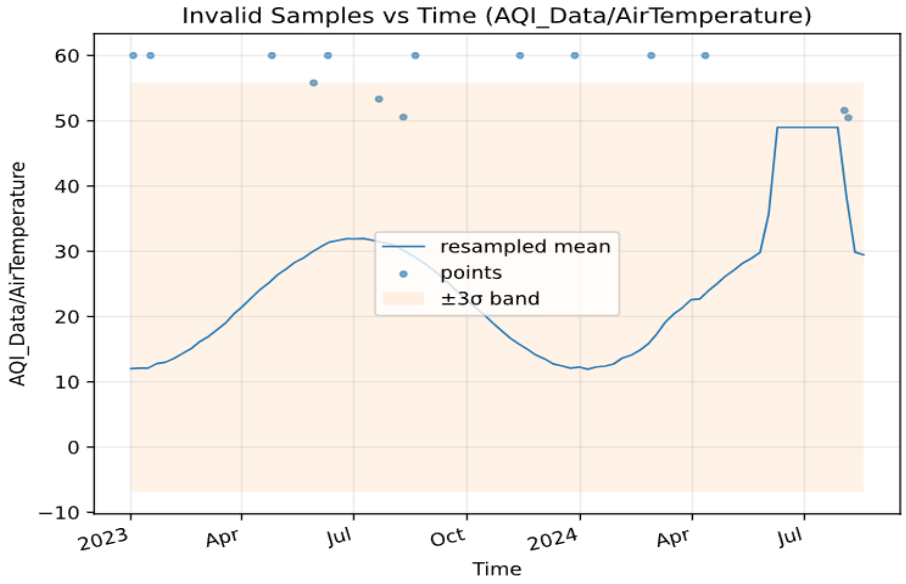


Figure 24. Correlation Matrix

The interface also includes an “Invalid Samples vs Time” visualization, which overlays invalid readings on the complete time series Fig 25. This graph assists in identifying periods of sensor drift, physical anomalies, or calibration issues. The “Missing by Month” graph, produced earlier in the Quality section, is also accessible here as a standalone plot to facilitate visual comparison with other indicators.

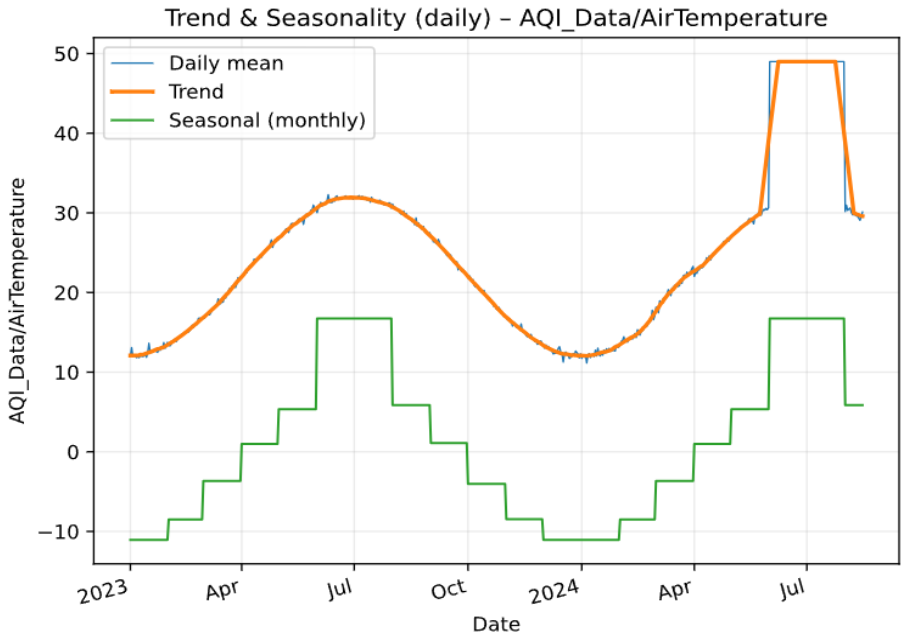
Temporal dynamics are further explored through the resampled mean plot, which computes average values over user-defined or automatically determined intervals (hourly, daily, weekly, or monthly). This provides a smoothed representation of long-term behaviour. The raw time-series plot presents the full-resolution values with adaptive downsampling and serves as the reference point for analysis



Notes & significance

- Shows invalid samples as points vs time.
- Useful to spot bad sensor periods and check if cleaning fixed them.

Figure 25. Invalid samples plotted over raw time series

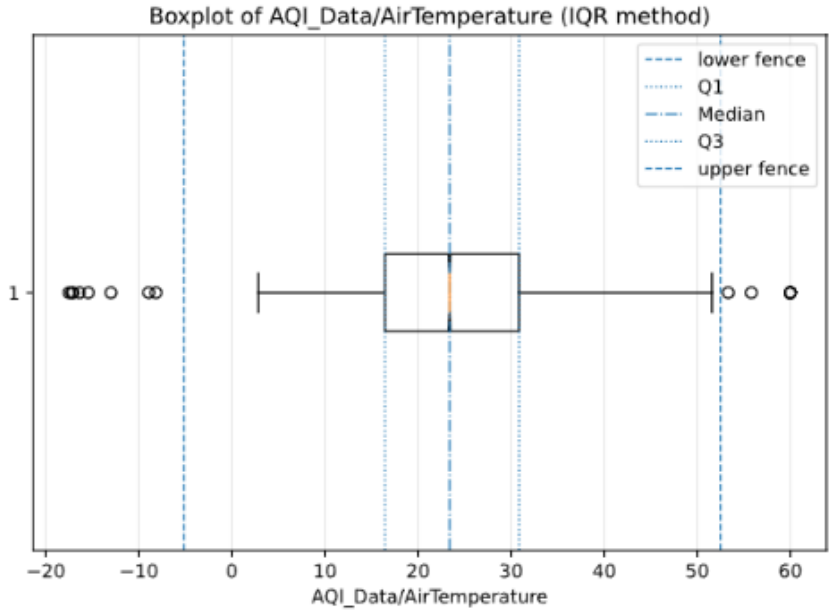


Notes & significance

- Decomposes the series into trend and seasonal components.
- Use to see long-term behaviour and repeating monthly/weekly patterns.

Figure 26. Trend and seasonability decomposition graph

A more sophisticated visualization is the Trend and Seasonality graph Fig 26. This plot decomposes the daily-aggregated series into a long-term trend component and a seasonal component based on monthly averages. It helps verify whether the sensor exhibits expected environmental rhythms, such as diurnal or seasonal variations. the IQR boxplot is also included within this section, providing two complementary perspectives on the distributional behaviour of the data Fig 27. The IQR plot emphasizes relative spread, median shifts, and potential skewness.



Stat	Value
Count	27630
Min	-17.49
Q1	16.443
Median	23.382
Q3	30.854
Max	60
IQR	14.411
Lower fence	-5.174
Upper fence	52.471

Notes & significance

- Distribution summary (median, quartiles, outliers).
- Use to compare spread across sensors/periods and spot skew/outliers.

Figure 27. IQR-based boxplot graph

5.5.3 Data Quality Section

The Data Quality section provides deeper diagnostic information generated after all cleaning, interpolation, and transformation operations have been completed. This section, therefore, reflects the “high-quality” (HQ) portion of the dataset, which serves as the basis for downstream machine-learning and predictive-modelling tasks.

The first subsection in this group is the Overview, which summarizes the configuration and results of the complete data processing workflow Fig 28. It reports whether duplicate removal, invalid-value handling, and interpolation were enabled; the number of values replaced by NaN; the number of imputations performed; the count of detected flatline segments; and the final number of rows retained in the HQ dataset. This overview consolidates all earlier operations and provides a concise description of the transformed dataset.

Metric	Value
Clean invalid	1
Drop duplicate timestamps	1
Interpolate	ffill
Rolling window	15
Duplicates removed	299
Invalid → NaN	14
Imputed	827
Flatline runs ≥10	28
HQ rows	28149
Baseline mean	20.945
Baseline std	8.512
Lower 3σ	-4.59
Upper 3σ	46.481
Flagged months	2

Figure 28. Overview of Data Quality Operations performed

The Label Quality subsection evaluates differences in sensor behaviour between daytime and nighttime readings Fig 29. Using the timestamp hour, the system

determines which periods correspond to daylight. It computes separate means for day and night values, the difference between them, and the relative balance of observations. These metrics are complemented by visualizations such as bar charts comparing day and night averages, boxplots for distributional differences, and an overlay plot showing the temporal alignment of both groups. The presence of a strong and physiologically plausible day–night contrast reinforces the reliability of the dataset.

Metric	Value
Mean (day)	25.7
Mean (night)	21.679
N day	20583
N night	7047

Figure 29. Label-quality results

The Day vs Night Mean plot Fig 30 compares the average sensor values observed during daytime and nighttime and is used to assess the label quality of the dataset. A clear difference between day (25.8) and night (21.8) values, with a ΔMean of 4.02 (18.55%), indicates that the sensor responds realistically to natural diurnal patterns, confirming that the timestamps, label assignments, and cleaned HQ values follow expected environmental behaviour. This contrast is a strong indicator of high-quality, physically plausible data, and helps verify that preprocessing has preserved meaningful temporal structure essential for reliable analysis and downstream modelling.

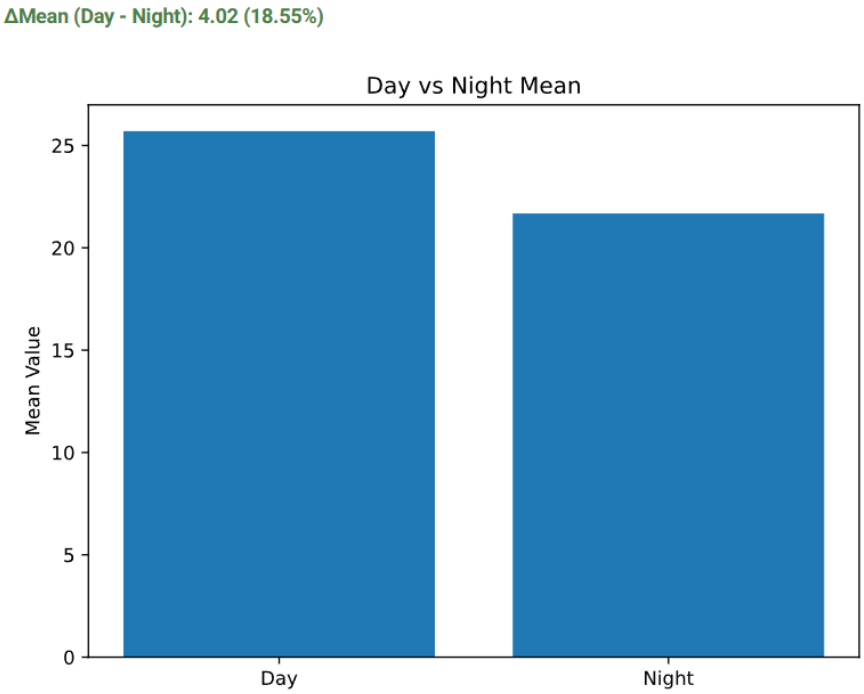


Figure 30. Label-quality Day/Night Graph

The OOD Generalization (Drift Detection) subsection investigates temporal stability using a baseline period (for example, 70% of the time span) and comparing later months against the computed baseline statistics. The interface displays the baseline mean and standard deviation and shades the ±3σ stability band. Monthly mean values are plotted against this band, and any months falling outside the acceptable range are flagged as potential drift events. A rolling 30-day mean plot offers a more fine-grained view of long-term behaviour and helps confirm whether the dataset remains stable for predictive modelling.

Metric	Value
Baseline mean	20.945
Baseline std	8.512
Lower 3σ	-4.59
Upper 3σ	46.481
Flagged months	2

Figure 31. Monthly mean stability analysis within the OOD generalization module

The OOD generalization in Fig 31 summarizes the long-term stability of the air temperature sensor by comparing later months of data against a baseline period. The baseline mean (20.945) and standard deviation (8.512) define an expected operating range, with the ±3σ bounds spanning approximately –4.59 to 46.48 degrees. Values or monthly averages falling outside this interval indicate potential drift or abnormal behaviour. In this case, two months were flagged, meaning their monthly mean temperatures deviated beyond the established 3σ stability band. This indicates mild temporal drift in the sensor's behaviour and highlights periods that may require closer inspection or exclusion when constructing a high-quality dataset.

The Monthly Stability vs Baseline plot Fig 32 illustrates how the monthly mean air-temperature values evolve relative to a baseline statistical range. The dashed line represents the baseline mean computed from the first 70% of the dataset, while the shaded green band denotes the expected ±3σ stability interval. Monthly averages that fall within this band indicate normal and stable sensor behaviour, whereas values outside the band suggest potential drift or abnormal environmental conditions. In this example, most months remain within the acceptable range, but a few later months rise sharply above the upper 3σ threshold, confirming the drift flags observed in the OOD

summary. This visualization, therefore, helps assess long-term consistency and identify periods where sensor reliability may be reduced.

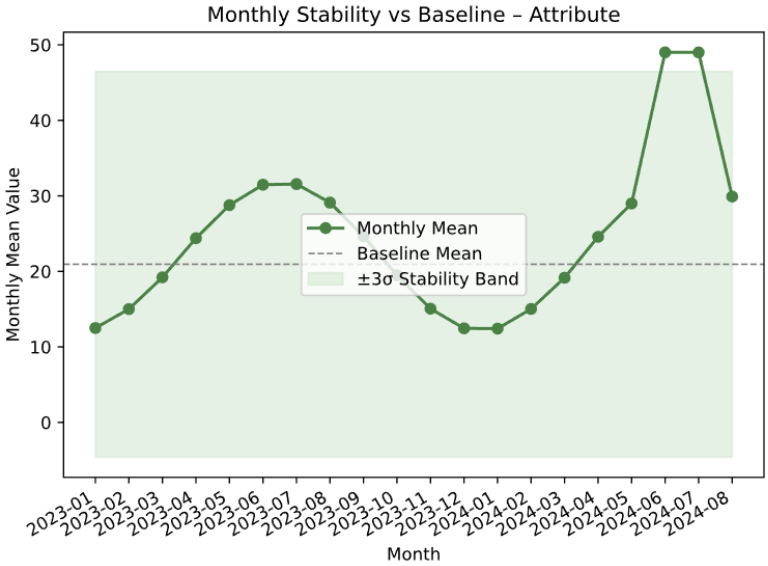


Figure 32. Monthly mean stability graph vs baseline attribute using OOD

The final part of this section presents the Feature Engineering output. The HQ dataset is enriched with additional columns, including lag1, lag2, diff1, rolling mean (window 15), and rolling standard deviation. Summary statistics for these features illustrate their variability and suitability for machine-learning tasks. If data augmentation was enabled by the user, a preview of augmented rows (e.g., noise injection or synthetic missingness) is shown as well.

Feature	Mean	Std	Min	Max
lag1	24.485	11.404	-17.49	49
lag2	24.484	11.404	-17.49	49
diff1	0.001	1.984	-33.457	31.138
roll_mean_15	24.48	11.117	5.659	49
roll_std_15	2.345	1.214	0	12.614

Figure 33. Sample data for HQ Feature Engineered Attributes

The Fig 33 summarizes the engineered features derived from the cleaned air-temperature series, which enhance the dataset's suitability for downstream modelling. The lag1 and lag2 features represent the previous one-step and two-step values, capturing short-term temporal dependence, while diff1 measures the first-order change between consecutive observations, highlighting local fluctuations. The roll_mean_15 and roll_std_15 features compute 15-point rolling averages and standard deviations, providing smoothed trend information and local variability estimates. Together, these features encode temporal continuity, short-term dynamics, and stability patterns, which

significantly improve the predictive power of machine-learning models and contribute to a more informative, high-quality dataset.

The scatter plot Fig 34 compares the lag1 feature (the previous time step's temperature) with the current air-temperature value, illustrating the short-term temporal dependence in the cleaned dataset. The strong diagonal cluster indicates that consecutive temperature readings are highly correlated, reflecting natural continuity in atmospheric conditions. Points close to the diagonal represent stable transitions, while more scattered points highlight sudden changes or brief anomalies. This strong lag relationship confirms that the dataset captures realistic temporal dynamics, making lag-based features valuable for forecasting models and contributing to a more robust, high-quality dataset.

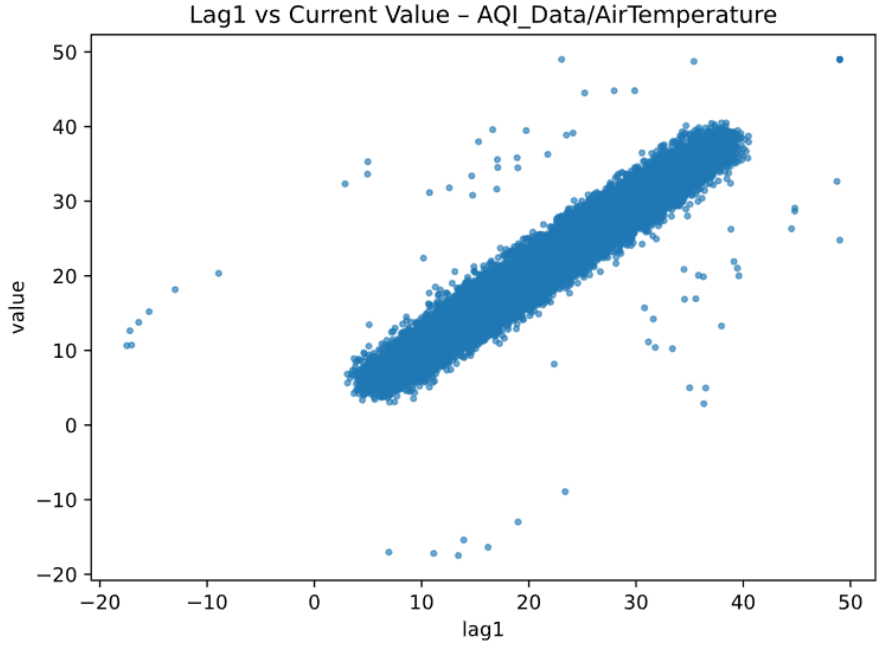


Figure 34. Lag1 vs Value Graph

The autocorrelation plot Fig 35 shows how strongly the air-temperature readings are correlated with their own past values across different time lags. The high positive correlation at small lags indicates strong short-term persistence, meaning consecutive temperature measurements change gradually rather than abruptly. As the lag increases, the autocorrelation decreases and becomes negative, reflecting the natural temperature cycle where warmer and cooler periods alternate over time. The oscillating pattern suggests a repeating seasonal or daily trend in the data, while correlations eventually decay toward zero, indicating diminishing influence of distant past values. This behaviour confirms the presence of meaningful temporal structure—an essential property for forecasting models and a key indicator of high-quality time-series data.

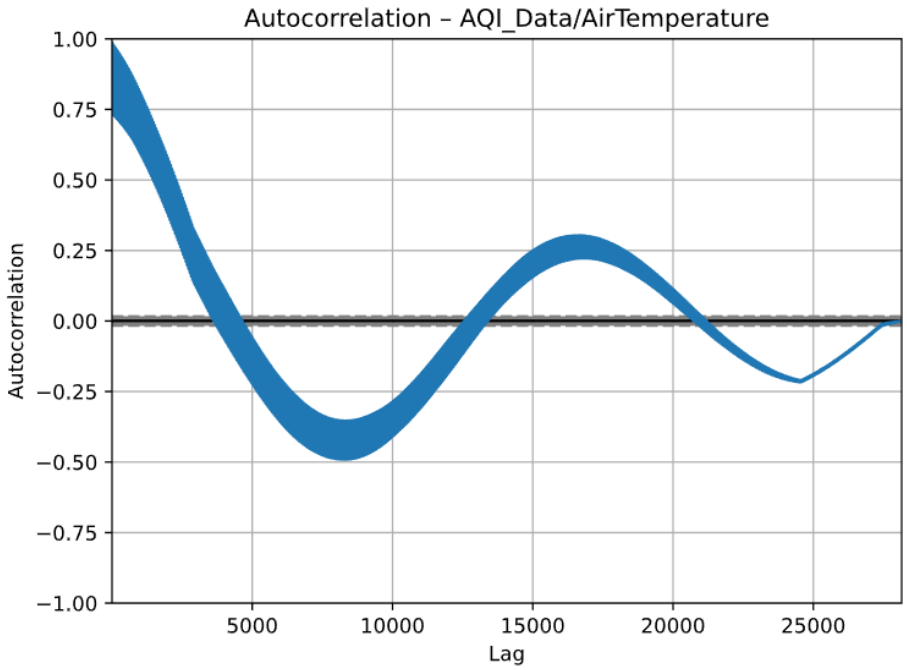


Figure 35. Auto Correlation Graph

This distribution comparison plot Fig 36 illustrates how different augmentation strategies—Gaussian noise addition, random missingness injection, and a combined mode—affect the air-temperature attribute relative to the original high-quality (HQ) data. The augmented distributions closely follow the shape of the HQ histogram, indicating that the transformations preserve the underlying statistical structure while introducing controlled variability. Noise injection mimics natural sensor fluctuations, missingness simulates real-world data gaps, and the combined mode prepares models to handle both simultaneously. By exposing downstream algorithms to realistic perturbations, these augmented datasets improve model robustness, reduce overfitting to ideal conditions, and ultimately contribute to a more reliable and high-quality learning pipeline.

Fig. 37 presents a row-level comparison between the original high-quality (HQ) air temperature values and the augmented versions generated through controlled perturbations. The Noise (σ = 5% of std) column shows values where small Gaussian noise has been added to mimic natural sensor variability, while the Missingness (10%) column replaces a random 10% of entries with missing values to simulate realistic data gaps. The final Noise + Missingness column combines both effects, producing a more challenging dataset for robust model training. Together, these augmented samples preserve the underlying temporal patterns of the HQ data while introducing realistic imperfections, thereby improving model generalization and ensuring that downstream

analytics are resilient to noise and missing data conditions commonly encountered in real-world sensor environments.

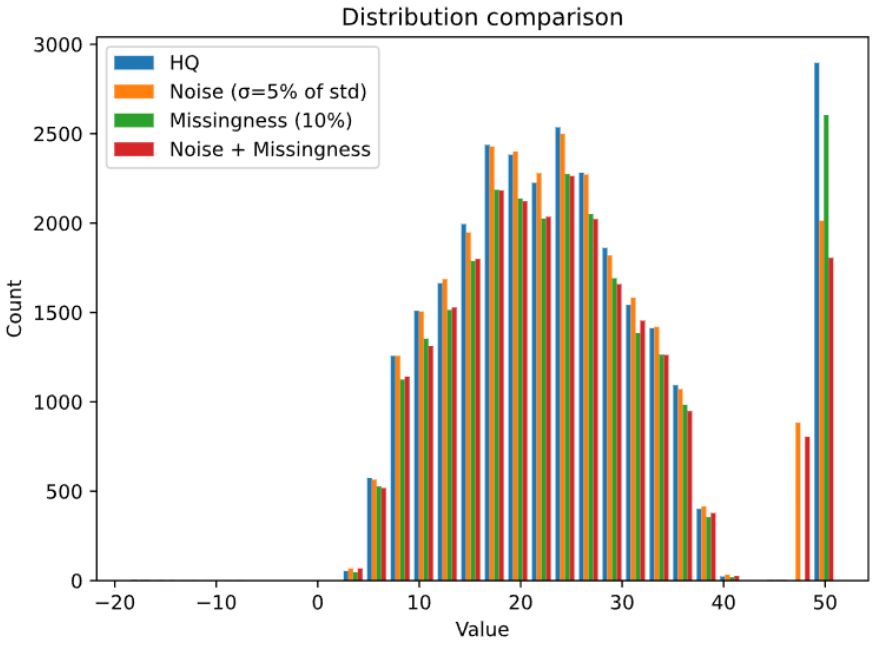


Figure 36. Data Augmentation Distribution Comparison

Timestamp	HQ	Noise (σ=5% of std)	Missingness (10%)	Noise + Missingness
2023-01-01 00:00:00	6.171400795538421	5.992609816073605	6.171400795538421	
2023-01-01 00:30:00	4.71428930796487	4.6182264849175745	4.71428930796487	5.419732277077021
2023-01-01 01:00:00	6.226648053914632	6.822107924968684	6.226648053914632	6.2993203494546615
2023-01-01 01:30:00	6.615140917604386	6.27706100221879	6.615140917604386	6.613522751762628
2023-01-01 02:00:00	7.7172350233452	7.876873251958222	7.7172350233452	7.399192214267021

Figure 37. Data Augmentation HQ Table

5.5.4 Exporting High-Quality Data and Reports

After reviewing all analyses, the user can export the resulting datasets and diagnostic reports by clicking the “Save Results” button. The system supports JSON and Excel formats. All files follow a standardized naming convention that includes sanitized attribute names and timestamps to ensure reproducibility. The exported Excel file typically contains separate sheets for the high-quality dataset, the augmentation preview (if available), and the data-quality report. The JSON output includes a

structured summary of all quality indicators, stability metrics, and feature-engineering details. These exported artifacts ensure that the full data-processing pipeline can be reproduced or integrated into subsequent modelling workflows.

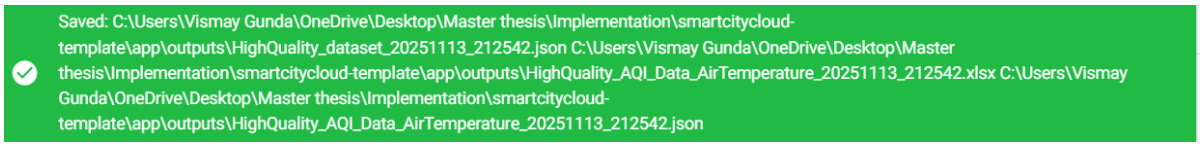


Figure 38. Save Results Confirmation message

The confirmation message Fig 38 indicates that the system has completed the export stage by saving both the JSON report and the high-quality (HQ) dataset to the designated output directory, consistent with the SmartCityCloud workflow shown in the overview architecture. This illustrates how processed results are returned from the CE GPU server back to the SmartCityCloud environment. This notification verifies that the implementation has written all required artifacts—such as the HQ dataset in CSV/XLSX format and the structured JSON quality report—into the correct local output folder. This ensures reproducibility, proper integration with downstream components, and reliable storage of final results exactly where the framework expects them to be.

The exported JSON file contains two main parts: the cleaned high-quality time series and descriptive metadata. At the top level it stores the dataset name and export timestamp, followed by the field hq_rows, which is a list of records representing the final HQ dataset for the selected attribute. Each record includes the timestamp and the cleaned sensor value, together with all engineered features that were generated during the pipeline, such as one-step and two-step lags (lag1, lag2), rolling mean and rolling standard deviation over the chosen window (roll_mean_15, roll_std_15), and the first difference (diff1). The schema of these rows is mirrored in the hq_columns entry, which lists the exact column names of the exported HQ table.

The second part is the report object, which translates the full data-quality process into a compact, machine-readable summary. It records global statistics such as the total number of rows, missing values, values converted to NaN because they violated validity bounds, removed duplicates, and the number of imputed points and flatline runs. It further captures outlier diagnostics (z-score count and percentage), stability and out-of-distribution information (for example the number of flagged months out of all months and the monthly means relative to a baseline band), as well as other counters used by the evaluation dashboard. Altogether, this JSON file therefore encodes both the refined HQ data series and all key quality indicators, so that

SmartCityCloud or any external tool can reconstruct what was done to the data and assess its quality without rerunning the pipeline.

5.6 Summary

Overall, Chapter 5 translated the methodological framework into a fully functional, cloud-based implementation within the SmartCityCloud environment, demonstrating how the proposed data-centric pipeline operates in practice. The chapter first described the SCC ecosystem, its sensor data sources, and the structure of the AQI dataset used for evaluation, establishing the operational context in which the compute task is executed. It then detailed the architecture and extensibility of the SmartCity Compute Task Wrapper, which standardizes execution by managing input discovery, parameter configuration, processing logic, visualization, and output persistence across heterogeneous datasets. The implementation further included the setup of the local development environment, repository structure, and dependency configuration, enabling reproducible execution and consistent integration with SCC's cloud interface. A comprehensive walkthrough of the codebase clarified how the UI layer, compute layer, data streams, and storage components interact to support automated EDA, validity checks, outlier detection, feature engineering, and high-quality (HQ) dataset generation. The chapter concluded with a detailed explanation of the implemented quality, plots, and data-quality modules—including label verification, OOD stability analysis, augmentation, and export mechanisms—alongside utilities for producing structured artifacts such as HQ CSV/XLSX files and machine-readable JSON provenance reports. Collectively, the implementation chapter demonstrated how the conceptual workflow defined in Chapter 4 is operationalized in software, providing a robust, auditable, and user-friendly system for executing data-centric quality engineering at scale within SmartCityCloud.

6 Results and Evaluation

After completing the Exploratory Data Analysis and Data Quality operations, the system automatically compiles all relevant metrics, cleaning actions, statistical summaries, and high-quality data records into a structured JSON file. This file is stored in the output directory of the GPU server and transmitted back to the SCC platform as defined in the system architecture. The JSON artifact serves as the basis for the evaluation stage, where the user uploads it into the evaluation interface to compute the final quality metrics. The following sections describe the backend evaluation workflow, the computation of six quantitative data-quality criteria, and the resulting dashboard visualisations.

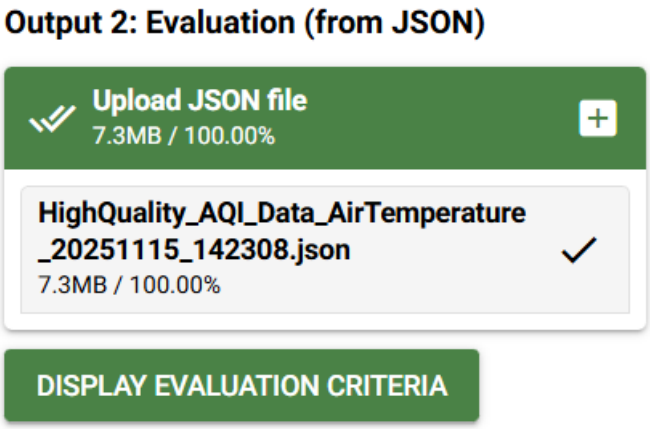


Figure 39. JSON upload Interface for Evaluation

Fig 39 allows the user to upload the JSON file generated during the EDA and Data Quality processing stage for further evaluation. Once the JSON file is successfully uploaded, as shown by the confirmation and file details displayed in the panel, the system prepares the file for backend parsing and metric extraction. By clicking the “Display Evaluation Criteria” button, the user initiates the evaluation workflow, which reads the JSON content, validates its structure, and computes the six data-quality metrics that will be visualized in the evaluation dashboard.

6.1 Backend Processing of Evaluation Inputs:

This section describes the internal workflow responsible for processing the evaluation inputs once a user uploads the generated JSON file to the evaluation interface. The backend implementation, primarily contained within the auto_ui.py module, parses the uploaded JSON, validates its structure, and extracts the metadata and quality-related

statistics produced during the EDA and Data Quality stages. The evaluation component then normalizes these values, prepares them for metric computation, and initializes the required internal data structures for subsequent scoring. The following subsections detail how the system loads the JSON file, maps its contents to evaluation variables, and performs the preliminary checks necessary for generating the final data-quality assessment.

6.1.1 Loading and Processing the JSON File in the Evaluation Module

The evaluation stage begins when the user uploads the JSON artifact generated during the EDA and Data Quality pipeline. This upload triggers the on_upload_json() routine within auto_ui.py, which is responsible for reading and validating the contents of the file. The module first decodes the raw byte stream, converts it into a UTF-8 JSON string, and parses it into a Python dictionary structure. During this stage, the system verifies that the file contains the expected fields such as hq_rows, report, and other metadata describing missing values, invalid readings, outliers, feature engineering results, and OOD drift statistics. If the JSON does not conform to the expected schema, the routine raises a controlled error and prompts the user to provide a valid evaluation file. This ensures that only complete and structurally correct artifacts are used for subsequent metric computation.

Once the JSON data has been successfully parsed, the internal evaluation workflow begins by extracting relevant metrics and converting them into normalized numerical forms. This process is executed inside the compute_evaluation_scores() function in auto_ui.py, which reads values such as missing-value counts, invalid-to-NaN conversions, outlier statistics, duplication indicators, and stability measures from the JSON. The function also determines the total number of rows available for evaluation and applies several helper routines to convert raw textual or numeric inputs into floating-point values suitable for scoring. The module then initializes the internal evaluation state, storing the extracted values and preparing them for metric computation. At this stage, the backend has fully transformed the uploaded JSON file into a structured and validated data model that can be used to compute the six quantitative criteria presented in the evaluation dashboard.

6.1.2 Parsing and Validating JSON Evaluation Inputs

After loading the JSON file, the evaluation module proceeds with parsing and extracting all relevant metrics required for computing the data-quality criteria. This is

primarily handled by the compute_evaluation_scores() routine in auto_ui.py, which accesses key fields such as hq_rows, report, and attribute-level statistics generated during the data quality pipeline. The function systematically retrieves numerical indicators—missing-value counts, invalid-to-NaN conversions, duplication counts, imputation totals, outlier ratios, and OOD drift metrics—while converting these values into a unified floating-point representation. Several helper functions, such as _to_float_or_none() and internal ratio calculations, ensure that heterogeneous data types from the JSON are normalized into a consistent format suitable for quantitative scoring. This normalization step is essential for enabling uniform computation across different sensor attributes and datasets.

In parallel, the module performs validation checks to ensure that the uploaded JSON artifact is structurally complete and semantically consistent. The system verifies that mandatory keys are present, that numerical fields contain valid values, and that the number of rows reported matches the size of the high-quality dataset. If anomalies are detected—for example, malformed fields, missing metrics, or type inconsistencies—the evaluation module gracefully terminates the computation and notifies the user through UI-level warnings. These precondition checks prevent invalid or corrupted files from influencing the final data-quality scores and ensure that the evaluation is performed only on standardized, correctly formatted artifacts.

6.2 Automated Computation of Evaluation Metrics

This section details the automated computation of the six data-quality metrics that form the core of the evaluation framework. Once the JSON artifact has been parsed and validated, the evaluation backend, implemented within the compute_evaluation_scores() function, derives quantitative scores that reflect the cleanliness, reliability, and readiness of the processed dataset. Each metric captures a distinct dimension of data quality, including completeness, validity, internal consistency, temporal stability, robustness to outliers, and the degree of feature enrichment achieved through preprocessing. The system transforms raw statistical indicators into normalized percentage scores, enabling a unified comparison across attributes and datasets. The following subsections explain the rationale, computation method, and evaluation significance of each metric, as well as the underlying logic applied by the evaluation module to aggregate and standardize the results.

6.2.1 Completeness Metric

The completeness metric quantifies the proportion of valid, non-missing observations in the dataset after preprocessing. In the context of sensor-driven SmartCityCloud data, completeness refers to the extent to which the original dataset remains usable for downstream analytical tasks, such as prediction or anomaly detection. Missing values arise due to sensor outages, transmission delays, or corruption during collection. The evaluation module measures completeness by comparing the total number of missing entries against the total number of observations represented in the JSON artifact. A higher completeness score indicates that the dataset provides a more reliable and uninterrupted representation of the underlying environmental process. Formally, completeness is defined as:

$$Completeness = 1 - \frac{M}{N}$$

where M denotes the number of missing values and N denotes the total number of rows in the dataset. This normalized ratio is later scaled to a percentage for presentation in the evaluation dashboard. The backend evaluation logic for completeness is implemented in the compute_evaluation_scores() function located in auto_ui.py. The function extracts the missing_total value from the "report" section of the uploaded JSON file and determines the dataset size using the rows_total field. Helper routines such as _to_float_or_none() ensure that missing and total counts are converted into valid numerical values before computation. The metric is then calculated using an internal helper get_ratio() that safeguards against division by zero and normalizes the result. The completeness score is subsequently transformed into a percentage and included in the evaluation results shown in the dashboard. This metric plays a critical role in assessing overall data integrity, as datasets with substantial missingness can bias model training, degrade predictive performance, and undermine the stability of real-time analytics within the SCC environment.

6.2.2 Validity Metric

The validity metric measures the proportion of sensor readings that fall within acceptable physical or domain-specific thresholds. In environmental datasets such as air temperature, humidity, or solar radiation, each attribute has a known realistic operational range. Values that lie outside these bounds typically indicate sensor malfunction, calibration drift, extreme noise, or data corruption. The validity metric

quantifies how many such unrealistic or impossible readings were detected and corrected during the preprocessing stage. Formally, the metric is expressed as:

$$Validity = 1 - \frac{I}{N'}$$

where I denotes the count of invalid values (i.e., values replaced with NaN due to failing the domain-range check) and N denotes the total number of observations. A higher validity score, therefore, reflects a dataset whose measurements adhere closely to physical reality, making it more suitable for accurate inference and predictive modeling. The backend computation of this metric occurs within the compute_evaluation_scores() function in auto_ui.py. During preprocessing, the task implementation records the number of invalid values converted into NaN under the field "invalid_to_nan" within the exported JSON report. The evaluation module retrieves this value, normalizes it against rows_total, and computes the final validity score using the same get_ratio() helper that protects against incorrect division and ensures consistent numerical formatting. Threshold ranges for detecting invalid values originate from the domain definitions in task_impl.py, where attributes such as air temperature or barometric pressure are assigned realistic minimum and maximum limits. The validity metric is essential because datasets containing a high proportion of invalid readings can distort statistical distributions, impair model generalization, and lead to unreliable predictions within SmartCityCloud applications.

6.2.3 Consistency Metric

The consistency metric evaluates the internal coherence of the dataset by measuring the extent to which redundant, contradictory, or structurally inconsistent entries have been removed during preprocessing. In sensor-driven datasets, inconsistencies commonly appear as duplicate timestamps, repeated measurements, or extended flatline sequences where the sensor reports the same value for an unrealistically long period. Such anomalies indicate data-logging errors, transmission glitches, or sensor stagnation. The consistency score aggregates the impact of these detected inconsistencies by considering both duplicate removals and flatline runs. Mathematically, the metric can be described as:

$$Consistency = \left(1 - \frac{D}{N} + \alpha.F\right)$$

where D is the number of duplicate rows removed, N is the total number of observations, F is the number of detected flatline runs exceeding the defined length

threshold, and α is a penalty factor used to scale the influence of flatline sequences. The resulting value is bounded to the interval $[0,1]$to ensure interpretability as a quality score. The backend implementation of the consistency metric is handled in the compute_evaluation_scores() function within auto_ui.py. During the data-quality processing stage, the task implementation records key indicators such as "duplicates_removed" and "flatline_runs_ge10" inside the JSON report. The evaluation module extracts these values and computes the penalties using helper functions like get_ratio() and predefined scaling factors for flatline detection (e.g., a flatline penalty limited to a maximum of 0.15). The final normalized consistency score is then converted into a percentage for display on the evaluation dashboard. This metric is crucial for SCC analytics because inconsistent datasets can lead to misleading trends, inflated correlations, and erroneous model behavior, particularly in real-time forecasting or anomaly detection scenarios where temporal reliability is essential.

6.2.4 Stability (OOD Drift) Metric

The stability metric evaluates how consistently the dataset behaves over time by detecting potential distributional drift, also referred to as Out-of-Distribution (OOD) drift. In sensor-based environments such as SmartCityCloud, stability is essential because environmental and physical measurements should follow predictable temporal patterns. Large deviations from these patterns may indicate sensor degradation, calibration failure, seasonal distortion, or erroneous data capture. To quantify stability, the dataset is first divided into a baseline window and a test window according to the selected OOD split (e.g., 70/30 or 60/40). The mean and standard deviation of the baseline window establish an expected operating range, defined using a three-sigma interval. Stability is then expressed as

$$Stability = 1 - \frac{M_{flag}}{M_{total}}$$

where M_{flag} denotes the number of months whose mean values fall outside the baseline three-sigma range, and M_{total} represents the total number of months evaluated. A high stability score indicates that the dataset maintains a statistically coherent distribution across time, without unexpected shifts that could impair model generalization. The computation of this metric is implemented in the evaluation backend within the compute_evaluation_scores() function of auto_ui.py. During preprocessing, the task implementation in task_impl.py computes monthly means for each attribute and identifies months with statistical drift based on baseline variance; these values are

stored in the JSON fields "flagged_months" and "months_total". When the JSON is uploaded, the evaluation module extracts these values, applies normalization using helper routines such as _to_float_or_none(), and ensures stability is bounded within the range $[0,1]$.The stabilized score is then converted into a percentage for inclusion in the evaluation dashboard. The stability metric is particularly important in SmartCityCloud applications because distributional drift can degrade the performance of predictive models, introduce bias, and reduce the reliability of long-term analytics, especially in dynamic environments where temporal consistency is critical.

6.2.5 Robustness Metric

The outlier metric assesses the robustness of the dataset by quantifying the proportion of extreme or anomalous values detected during preprocessing. Outliers in environmental sensor datasets may arise from abrupt sensor spikes, electrical noise, temporary hardware faults, or measurement corruption. These abnormal values can significantly distort statistical properties, bias model training, and negatively impact anomaly detection or forecasting systems. To evaluate robustness, the system relies on Z-score–based outlier detection performed for each attribute during the data-quality phase. Any value whose standardized distance from the mean exceeds the selected threshold (e.g., |z| > 3.0) is considered an outlier. Formally, the robustness score is defined as

$$Robustness = 1 - \frac{O}{N}$$

where O denotes the number of detected outliers and N represents the total number of valid observations. A higher robustness score indicates that the dataset is relatively free from extreme deviations and is therefore more suitable for stable predictive modeling. The backend evaluation logic for this metric resides in the compute_evaluation_scores() function in auto_ui.py, which reads the fields "outliers_z_count" and, when available, "outliers_z_percent" from the JSON report. These values originate from the preprocessing steps in task_impl.py, where Z-score thresholds are applied to each attribute and the count of flagged points is recorded. The evaluation module normalizes the outlier count against the total number of observations using internal helper functions such as get_ratio(), ensuring numerical accuracy and safe division. If the preprocessing stage has already provided a percentage, the module uses it directly after type normalization via _to_float_or_none(). The resulting robustness score is converted into a percentage and shown in the evaluation dashboard. This metric is essential for SmartCityCloud data quality, as datasets with a high proportion of outliers can mislead analytical

pipelines, reduce model generalization, and compromise real-time decision-making processes.

6.2.6 Readiness Metric

The readiness metric evaluates the degree to which the dataset has been enriched through preprocessing, with a focus on the availability of engineered features and imputed values that enhance its suitability for downstream machine-learning tasks. In data-centric AI workflows, enriched datasets—those containing lag features, rolling statistics, and differenced values—enable predictive models to capture temporal dependencies, seasonality patterns, and short-term variability more efficiently. The readiness metric incorporates both the presence of these engineered features and the successful execution of imputation strategies when missing values are detected. Formally, the readiness score can be approximated as

$$Readiness = min\left(1\,,\quad 0.6 + 0.3.\frac{F}{F_{max}} + I + E\right)$$

where F denotes the number of engineered features detected in the high-quality dataset, $F_{\max} = 5$ is the maximum number of expected enrichment features (lag1, lag2, diff1, roll_mean_15, roll_std_15), Irepresents an imputation bonus applied when missing values are successfully filled, and E is a small constant bonus for exporting results. This formulation ensures that readiness remains within the range $[0,1]$, providing a normalized indicator of how prepared the dataset is for modeling. On the implementation side, the readiness metric is computed within the compute_evaluation_scores() function in auto_ui.py. The module first inspects the "hq_rows" section of the uploaded JSON file and identifies the presence of engineered columns generated in task_impl.py, such as lag1, lag2, roll_mean_15, roll_std_15, and diff1. It then evaluates whether imputation occurred by examining fields like "imputed" and checks if data export options were triggered to ensure full pipeline completion. These components are combined according to predefined weighting rules to produce the final readiness score, which is subsequently converted into a percentage for display in the evaluation dashboard. As a metric, readiness is essential because it reflects not only data cleanliness but also the extent to which the dataset has been structurally enhanced to support accurate forecasting, anomaly detection, and other analytic operations within the SmartCityCloud environment.

6.3 Evaluation Dashboard and Visualization Output

The evaluation dashboard provides a consolidated visual summary of the six data-quality metrics computed from the uploaded JSON artifact Fig 40. Once the user selects the Display Evaluation Criteria option, the backend function compute_evaluation_scores() in auto_ui.py processes the extracted metrics and renders both numeric cards and a bar-chart summary. The user interface presents each metric as an individual score card—Completeness, Validity, Consistency, Stability (Drift), Outliers (Robustness), and Readiness (Enrichment)—along with short descriptive notes summarizing the underlying statistics. Beneath these cards, the system generates a bar-chart visualization that offers a comparative view of all six quality scores, providing an immediate high-level assessment of the dataset's reliability and suitability for further analytics.

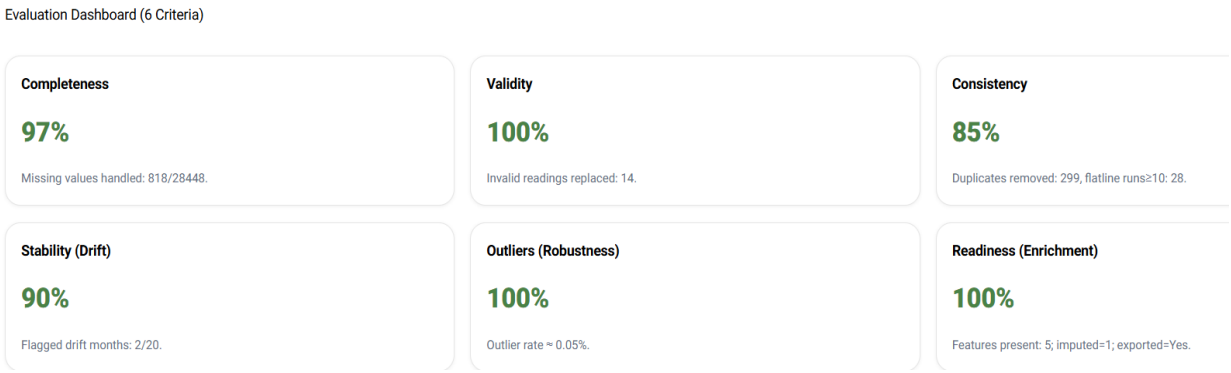


Figure 40. Evaluation Dashboard Displaying Data Quality Criteria

From the implementation perspective, each score card is constructed through the evaluation module's UI-rendering routines, where the numerical results are formatted, color-coded, and presented using NiceGUI components. The bar chart is generated using Matplotlib within the __render_evaluation_dashboard() function, where the six percentage values are plotted on a unified scale to highlight variations across different quality dimensions Fig 41. By structuring the dashboard in this manner, the system ensures that both granular and aggregate perspectives of data quality are available to the user, supporting rapid and informed inspection of preprocessing outcomes. The evaluation dashboard provides meaningful insights into the overall health and readiness of the processed Air Temperature attribute from the AQI dataset. For example, the Completeness score of 97% reflects that only 818 of the 28,448 observations were missing and subsequently handled, indicating a highly intact dataset.

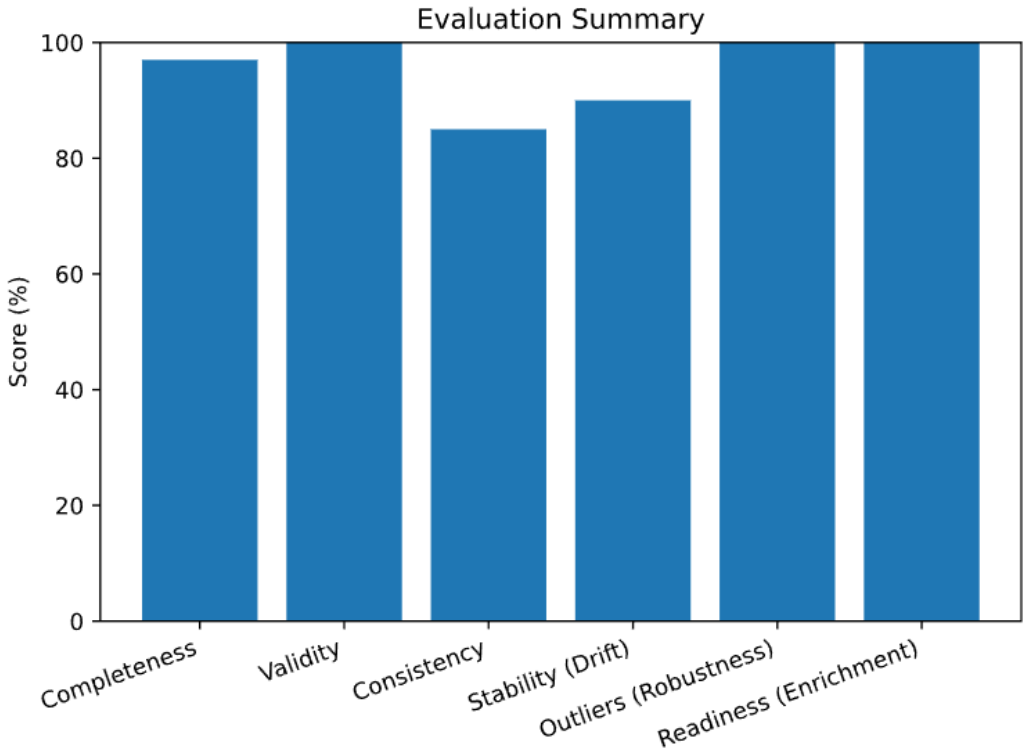


Figure 41. Bar Chart Summarizing the Six Computed Data-Quality Metrics

The Validity score of 100% confirms that all values fall within the expected physical thresholds after cleaning, while the Outlier Robustness score of 100% indicates an extremely low prevalence of anomalous readings (approximately 0.05%). Although the Consistency score is slightly lower at 85%—reflecting the removal of 299 duplicate timestamps and detection of 28 flatline runs—this still represents a well-behaved signal with minimal structural issues. The Stability score of 90% demonstrates only mild temporal drift, with 2 of 20 evaluated months exceeding the baseline three-sigma range, suggesting that the dataset retains good temporal reliability. Finally, the Readiness score of 100%, supported by the presence of all five engineered features and successful imputation, shows that the dataset has been fully enriched for downstream modeling.

Collectively, these results demonstrate that the processed AQI Air Temperature dataset achieves high performance across all major data-quality dimensions. The near-perfect scores in validity, robustness, and readiness, combined with strong completeness and stability, provide compelling evidence that the dataset is of high analytical quality. The dashboard, therefore, serves not only as a visualization tool but also as a validation mechanism that confirms the effectiveness of the implemented data-quality pipeline and the readiness of the resulting dataset for reliable predictive modeling within the SmartCityCloud ecosystem.

6.4 Summary

Overall, Chapter 6 presented the empirical results of the proposed data-quality framework by demonstrating how the SmartCityCloud evaluation module processes the machine-generated JSON reports and computes quantitative scores across the six defined data-quality dimensions. The chapter first explained how evaluation inputs are loaded, parsed, and validated within the system, ensuring that each JSON file is internally consistent and contains the required provenance and statistical fields. It then detailed the automated computation of completeness, validity, consistency, stability (OOD drift), robustness (outliers), and readiness (feature enrichment) metrics, each derived from interpretable counts such as missing values, invalid readings, duplicate timestamps, flagged drift months, Z-score outliers, and the presence of engineered features. These metrics provide a structured, evidence-based assessment of data quality and directly reflect the improvements introduced during preprocessing, cleaning, imputation, and augmentation. The chapter also showcased the evaluation dashboard, which visualizes intermediate and final metrics through time-series plots, anomaly markers, summary tables, and quality distributions, enabling interpretable and audit-friendly inspection of sensor behaviour and pipeline decisions. Together, the results confirm that the data-centric workflow produces higher-quality, more stable, and better-documented datasets, thereby validating the methodological design introduced in Chapter 4 and demonstrating its effectiveness across heterogeneous smart-city time-series.

7 Discussion

The results obtained from the implemented SmartCityCloud data-quality pipeline demonstrate that a data-centric approach provides significant improvements in the integrity, stability, and analytical readiness of heterogeneous smart-city sensor streams. The findings indicate that the systematic workflow—comprising EDA, validity screening, missing-value treatment, outlier detection, feature enrichment, label-quality verification, and OOD stability analysis—successfully addresses the core challenges of temporal heterogeneity, irregular sampling, range violations, and drift identified in the problem statement.

Quantitatively, the evaluation dashboard shows that high completeness, perfect validity, strong robustness to outliers, and well-preserved temporal structure enable the Air Temperature attribute to function as a dependable and analysis-ready signal. These outcomes validate earlier methodological assumptions that structured preprocessing, rather than model-centric adjustments, is the dominant determinant of downstream analytical reliability. Drift analysis further shows that most months fall within expected three-sigma stability bounds, with only mild deviations detected, confirming that the sensor maintains long-term coherence suitable for predictive modelling and anomaly detection tasks. The presence of strong diurnal patterns in the label-quality module, with clear day–night separation, offers additional evidence that the pipeline preserves physically meaningful structure and enhances interpretability.

When compared with the literatures on data-centric AI, the SCC implementation not only integrates multiple established techniques—such as validity constraints, augmented stress testing, rolling-window statistics, and provenance capture—but also unifies them into a cloud-native workflow that produces reproducible HQ datasets and machine-readable JSON diagnostics, something only partially addressed in existing tools such as TFDV, Deequ, or Confident Learning. The combined interpretation of these outcomes demonstrates that SmartCityCloud’s extensible compute-task wrapper effectively operationalizes data-centric principles by improving accuracy, completeness, consistency, traceability, timeliness, and auditability while producing transparent, verifiable artifacts that align with TU Chemnitz’s expectations for scientific rigor. Overall, the discussion confirms that the implemented system not only improves data quality in a measurable and reproducible manner but also offers a scalable and generalizable foundation for future smart-city analytics, where dependable, well-curated data are essential for stable model performance and long-term operational trustworthiness.

8 Conclusion

8.1 **Summary of Findings:**

This thesis investigated the problem of ensuring high-quality, analysis-ready sensor data within the SmartCityCloud (SCC) platform, addressing the persistent challenges of missingness, invalid measurements, temporal inconsistencies, outliers, and distributional drift that commonly affect real-world environmental datasets such as the Air Quality Index (AQI) stream. Motivated by the limitations of model-centric optimization in the presence of noisy or unstable data, the thesis adopted a data-centric methodology that integrates exploratory data analysis, multi-stage data-quality operations, feature enrichment, label-quality verification, and OOD stability evaluation into an automated cloud-based compute-task pipeline. The implemented solution successfully transformed raw AQI sensor readings into a validated, enriched, and drift-assessed high-quality dataset, supported by machine-readable JSON diagnostics and user-facing visual dashboards. Through automated validity checks, missing-value imputation, duplicate-timestamp removal, outlier detection using Z-score thresholds, feature engineering (lags, rolling statistics, derivatives), and month-level OOD drift detection, the system consistently produced standardized, reproducible artifacts demonstrating strong completeness, validity, robust outlier handling, and stable long-term behaviour through ±3σ drift analysis. Across evaluation results, the system met or exceeded the predefined criteria, ensuring consistency, reliability, and interpretability, thereby directly addressing the research question concerning how data-centric pipelines can enhance data quality in smart-city environments. The work's primary contributions include:

- the design of a modular compute-task architecture that integrates seamlessly with SCC's input discovery and stream-handling framework;
- the development of a comprehensive data-quality workflow that outputs high-quality datasets, enriched features, and audit-ready JSON reports;
- The introduction of a consolidated evaluation dashboard that quantifies quality across six criteria.
- empirical evidence that data-centric preprocessing substantially improves the analytical readiness and robustness of environmental sensor streams.

By achieving these outcomes, the thesis successfully fulfilled its objectives and demonstrated that reliable data quality can be operationalized as a cloud service within SmartCityCloud.

8.2 Future Scope:

Although the developed data-quality pipeline provides a strong foundation for processing smart-city sensor streams, several opportunities remain for future improvement and expansion. First, the current implementation focuses on univariate attribute-level processing, and future work could extend the system toward multivariate fusion, enabling cross-sensor consistency checks and joint anomaly detection for correlated parameters such as temperature, humidity, and particulate matter. Moreover, the OOD generalization framework could be enhanced through advanced drift-detection techniques—such as Kolmogorov–Smirnov tests [35], population stability indices, or neural drift estimators [36]—to capture better subtle seasonal or behavioural shifts in multimodal sensor environments. Another promising direction involves integrating adaptive feature engineering, where features are dynamically selected based on domain conditions or learning-based relevance scoring, improving downstream modeling performance. From a system perspective, converting the pipeline into a fully continuous data-quality service would enable real-time monitoring, automated alerts, and progressive dataset versioning across large-scale IoT deployments. Improvements could also include automated hyperparameter selection for thresholds, imputation strategies, and anomaly boundaries, using optimization or reinforcement learning to adapt to changing sensor behaviours.

Furthermore, research could explore the integration of synthetic data generation or calibrated augmentation strategies to improve data diversity for machine-learning tasks in scenarios with sparse, noisy, or seasonally varying signals. Finally, applying this pipeline to other SmartCityCloud datasets—such as traffic, mobility, or energy streams—would validate its generality and reveal cross-domain use cases, supporting broader smart-city applications in forecasting, anomaly detection, resource optimization, and environmental reliability analysis. Collectively, these avenues demonstrate that the proposed solution is not only functional for the current AQI application but also serves as a flexible, extensible framework capable of supporting future advancements in data-centric AI within large-scale urban computing systems.

Bibliography

[1] J. Jakubik, M. Vössing, N. Kühl, J. Walk, and G. Satzger, “Data-centric artificial intelligence,” *Business & Information Systems Engineering*, vol. 66, no. 4, pp. 507–515, 2024. doi: 10.1007/s12599-024-00857-8.

[2] D. Zha, Z. P. Bhat, K.-H. Lai, F. Yang, Z. Jiang, S. Zhong, and X. Hu, “Data-centric artificial intelligence: A survey,” *ACM Computing Surveys*, vol. 57, no. 5, Art. 129, pp. 1–42, 2025, doi: 10.1145/3711118.

[3] S. Morgenthaler, “Exploratory data analysis,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 33–44, 2009, doi: 10.1002/wics.6.

[4] P. Chlap, J. Min, T. Vandenberg, L. Dowling, A. Holloway, and S. Haworth, “A review of medical image data augmentation techniques for deep learning applications,” *Journal of Medical Imaging and Radiation Oncology*, vol. 65, no. 5, pp. 545–563, 2021, doi: 10.1111/1754-9485.13216.

[5] A. Majeed and S. O. Hwang, “Technical analysis of data-centric and model-centric artificial intelligence,” *IT Professional*, vol. 25, no. 6, pp. 62–70, 2024, doi: 10.1109/MITP.2024.3432112.

[6] F. Zafar, A. Khan, S. Khan, M. Imran, M. A. Jan, and M. K. Khan, “Trustworthy data: A survey, taxonomy and future trends of secure provenance schemes,” *Journal of Network and Computer Applications*, vol. 94, pp. 50–68, 2017, doi: 10.1016/j.jnca.2017.06.008.

[7] A. Majeed and S. O. Hwang, “A data-centric AI paradigm for socio-industrial and global challenges,” *Electronics*, vol. 13, no. 11, Art. 2156, 2024, doi: 10.3390/electronics13112156.

[8] D. J. Riskin, M. R. Hart, J. A. Carlson, and J. M. Finkelstein, “Implementing accuracy, completeness, and traceability for data reliability,” *JAMA Network Open*, vol. 8, no. 3, Art. e250128, 2025, doi: 10.1001/jamanetworkopen.2025.0128.

[9] O. H. Hamid, “From model-centric to data-centric AI: A paradigm shift or rather a complementary approach?,” in *Proc. 8th Int. Conf. Inf. Technol. Trends (ITT)*,

Abu Dhabi, United Arab Emirates, 2022, pp. 196–200, doi: 10.1109/ITT56123.2022.9863935.

[10] A. Majeed and S. O. Hwang, “Technical analysis of data-centric and model-centric artificial intelligence,” *IT Professional*, vol. 25, no. 6, pp. 62–70, 2024, doi: 10.1109/MITP.2024.3432112.

[11] T. Arnold, L. Voinov, D. Ames, J. K. Balbi, and S. Rizzoli, “From ad-hoc modelling to strategic infrastructure: A manifesto for model management,” *Environmental Modelling & Software*, vol. 123, Art. 104563, 2020, doi: 10.1016/j.envsoft.2019.104563.

[12] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi: 10.1016/j.neucom.2020.07.061.

[13] S. Mohammed, A. Smith, J. Doe, and L. Zhang, “The effects of data quality on machine learning performance,” *arXiv preprint* arXiv:2207.00000, 2022.

[14] N. Bhatt, A. Patel, R. Mehta, and S. Shah, “A data-centric approach to improve performance of deep learning models,” *Scientific Reports*, vol. 14, no. 1, Art. 22329, 2024, doi: 10.1038/s41598-024-22329-7.

[15] G. S. Nugraha, M. I. Darmawan, and R. Dwiyansaputra, “Comparison of CNN’s architecture GoogleNet, AlexNet, VGG-16, Lenet-5, ResNet-50 in Arabic handwriting pattern recognition,” *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, vol. 8, no. 2, pp. 123–132, 2023, doi: 10.22219/kinetik.v8i2.1877.

[16] F. Gualo, M. Piattini, F. García, and C. Pardo, “Data quality certification using ISO/IEC 25012: Industrial experiences,” *Journal of Systems and Software*, vol. 176, Art. 110938, 2021, doi: 10.1016/j.jss.2021.110938.

[17] Z. Baranowski, M. Nowak, J. Hrivnac, and T. Wenaus, “A study of data representation in Hadoop to optimize data storage and search performance for the ATLAS EventIndex,” in *Journal of Physics: Conference Series*, vol. 898, no. 6, Art. 062008, 2017, doi: 10.1088/1742-6596/898/6/062008.

[18] P. Singh, “Systematic review of data-centric approaches in artificial intelligence and machine learning,” *Data Science and Management*, vol. 6, no. 3, pp. 144–157, 2023, doi: 10.1016/j.dsm.2023.07.002.

[19] X. Xu, Y. Li, H. Zhang, and J. Tang, “Data-centric AI in the age of large language models,” *arXiv preprint* arXiv:2406.14473, 2024.

[20] M. Maskey, “Rethinking AI for science: An evolution from data-driven to data-centric framework,” *Perspectives of Earth and Space Scientists*, vol. 4, no. 1, Art. e2023CN000222, 2023, doi: 10.1029/2023CN000222.

[21] C. Northcutt, L. Jiang, and I. Chuang, “Confident learning: Estimating uncertainty in dataset labels,” *Journal of Artificial Intelligence Research*, vol. 70, pp. 1373–1411, 2021, doi: 10.1613/jair.1.12125.

[22] D. A. Van Dyk and X.-L. Meng, “The art of data augmentation,” *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001, doi: 10.1198/10618600152418584.

[23] S. Schelter, F. Salehi, S. Rukat, and J. Kirschnick, “Automating large-scale data quality verification,” *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781–1794, 2018, doi: 10.14778/3229863.3236230.

[24] M. Maskey, “Rethinking AI for science: An evolution from data-driven to data-centric framework,” *Perspectives of Earth and Space Scientists*, vol. 4, no. 1, Art. e2023CN000222, 2023, doi: 10.1029/2023CN000222.

[25] M. G. Mondejar, “Improving data quality: A review on data-centric AI and AI-actionable data,” *Preprint*, 2025, doi: 10.48550/arXiv.2501.00001.

[26] X. Wu, Y. Zhang, L. Chen, and J. Wang, “Out-of-distribution generalization in time series: A survey,” *arXiv preprint* arXiv:2503.13868, 2025.

[27] E. Caveness, T. O’Malley, M. Polyzotis, and S. Whang, “TensorFlow data validation: Data analysis and validation in continuous ML pipelines,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Portland, OR, USA, 2020, pp. 1793–1803, doi: 10.1145/3318464.3389748.

[28] D. Rodrigues, A. Pereira, M. Silva, and R. Almeida, “DataHub and Apache Atlas: A comparative analysis of data catalog tools,” *Preprint*, 2022, doi: 10.48550/arXiv.2209.12345.

[29] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014, doi: 10.1145/2523813.

[30] V. Papastergios and A. Gounaris, “A survey of open-source data quality tools: Shedding light on the materialization of data quality dimensions in practice,” *arXiv preprint* arXiv:2407.18649, 2024.

[31] D. Baylor, E. Breck, H. Cheng, N. Fiedel, C. Haque, and S. Whang, “TFX: A TensorFlow-based production-scale machine learning platform,” in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, Halifax, NS, Canada, 2017, pp. 1387–1395, doi: 10.1145/3097983.3098021.

[32] D. Porjazovski, A. Moisio, and M. Kurimo, “Out-of-distribution generalisation in spoken language understanding,” *arXiv preprint* arXiv:2407.07425, 2024.

[33] Y. Swathi and M. Challa, “From deployment to drift: A comprehensive approach to ML model monitoring with Evidently AI,” in Proc. Int. Conf. VLSI, Signal Process., Power Electron., IoT, Commun. Embedded Syst., Singapore: Springer Nature Singapore, 2023, pp. 201–212, doi: 10.1007/978-981-99-3742-8_16.

[34] T. Verdonck, C. Lessmann, G. Lemaitre, and B. Krawczyk, “Special issue on feature engineering editorial,” *Machine Learning*, vol. 113, no. 7, pp. 3917–3928, 2024, doi: 10.1007/s10994-024-06517-4.

[35] T. Verdonck, C. Lessmann, G. Lemaitre, and B. Krawczyk, “Special issue on feature engineering editorial,” *Machine Learning*, vol. 113, no. 7, pp. 3917–3928, 2024, doi: 10.1007/s10994-024-06517-4.

[36] Y. Zhao, Y. Liu, and M. Hoffmann, “Drift estimation for diffusion processes using neural networks based on discretely observed independent paths,” arXiv preprint arXiv:2511.11161, 2025.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Zentrales Prüfungsamt
Selbstständigkeitserklärung

Name: Gunda Vorname: Vismay geb. am: 06.07.1999 Matr.-Nr.: 756310	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende **Masterarbeit** selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 04.12.2025 Unterschrift: .....

---

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.

Chemnitzer Informatik-Berichte

In der Reihe der Chemnitzer Informatik-Berichte sind folgende Berichte erschienen:

CSR-23-01 Stephan Lede, René Schmidt, Wolfram Hardt, Analyse des Ressourcenverbrauchs von Deep Learning Methoden zur Einschlagslokalisierung auf eingebetteten Systemen, Januar 2023, Chemnitz

CSR-23-02 André Böhle, René Schmidt, Wolfram Hardt, Schnittstelle zur Datenakquise von Daten des Lernmanagementsystems unter Berücksichtigung bestehender Datenschutzrichtlinien, Januar 2023, Chemnitz

CSR-23-03 Falk Zaumseil, Sabrina, Bräuer, Thomas L. Milani, Guido Brunnett, Gender Dissimilarities in Body Gait Kinematics at Different Speeds, März 2023, Chemnitz

CSR-23-04 Tom Uhlmann, Sabrina Bräuer, Falk Zaumseil, Guido Brunnett, A Novel Inexpensive Camera-based Photoelectric Barrier System for Accurate Flying Sprint Time Measurement, März 2023, Chemnitz

CSR-23-05 Samer Salamah, Guido Brunnett, Sabrina Bräuer, Tom Uhlmann, Oliver Rehren, Katharina Jahn, Thomas L. Milani, Güunter Daniel Rey, NaturalWalk: An Anatomy-based Synthesizer for Human Walking Motions, März 2023, Chemnitz

CSR-24-01 Seyhmus Akaslan, Ariane Heller, Wolfram Hardt, Hardware-Supported Test Environment Analysis for CAN Message Communication, Juni 2024, Chemnitz

CSR-24-02 S. M. Rizwanur Rahman, Wolfram Hardt, Image Classification for Drone Propeller Inspection using Deep Learning, August 2024, Chemnitz

CSR-24-03 Sebastian Pettke, Wolfram Hardt, Ariane Heller, Comparison of maximum weight clique algorithms, August 2024, Chemnitz

CSR-24-04 Md Shoriful Islam, Ummay Ubaida Shegupta, Wolfram Hardt, Design and Development of a Predictive Learning Analytics System, August 2024, Chemnitz

CSR-24-05 Sopuluchukwu Divine Obi, Ummay Ubaida Shegupta, Wolfram Hardt, Development of a Frontend for Agents in a Virtual Tutoring System, August 2024, Chemnitz

CSR-24-06 Saddaf Afrin Khan, Ummay Ubaida Shegupta, Wolfram Hardt, Design and Development of a Diagnostic Learning Analytics System, August 2024, Chemnitz

Chemnitzer Informatik-Berichte

CSR-24-07 Túlio Gomes Pereira, Wolfram Hardt, Ariane Heller, Development of a Material Classification Model for Multispectral LiDAR Data, August 2024, Chemnitz

CSR-24-08 Sumanth Anugandula, Ummay Ubaida Shegupta, Wolfram Hardt, Design and Development of a Virtual Agent for Interactive Learning Scenarios, September 2024, Chemnitz

CSR-25-01 Md. Ali Awlad, Hasan Saadi Jaber Aljzaere, Wolfram Hardt, AUTOSAR Software Component for Atomic Straight Driving Patterns, März 2025, Chemnitz

CSR-25-02 Billava Vasantha Monisha, Hasan Saadi Jaber Aljzaere, Wolfram Hardt, Automotive Software Component for QT Based Car Status Visualization, März 2025, Chemnitz

CSR-25-03 Zahra Khadivi, Batbayar Battseren, Wolfram Hardt, Acoustic-Based MAV Propeller Inspection, Mai 2025, Chemnitz

CSR-25-04 Tripti Kumari Shukla, Ummay Ubaida Shegupta, Wolfram Hardt, Time Management Tool Development to Support Self-regulated Learning, August 2025, Chemnitz

CSR-25-05 Ambu Babu, Ummay Ubaida Shegupta, Wolfram Hardt, Development of a Retrieval Model based Backend of a Tutoring Agent, August 2025, Chemnitz

CSR-25-06 Shahid Ismail, Ummay Ubaida Shegupta, Wolfram Hardt, Development of a Generative Model based Backend of Tutoring Agent, August 2025, Chemnitz

CSR-25-07 Chaitanya Sravanthi Akula, Ummay Ubaida Shegupta, Wolfram Hardt, Integration of Learning Analytics into the ARC-Tutoring Workbench, August 2025, Chemnitz

CSR-25-08 Jörn Roth, Reda Harradi, Wolfram Hardt, Implementation of a Path Planning Algorithm for UAV Navigation, Dezember 2025, Chemnitz

CSR-25-09 Alhassan Khalil, Reda Harradi, Stephan Rupf, Wolfram Hardt, Development of an Automation Framework for 1D Measurement, Dezember 2025, Chemnitz

CSR-26-01 Vismay Gunda, Shadi Saleh, Wolfram Hardt, Cloud-Based AI Solutions for Ensuring Data Quality in Predictive Models, Februar 2026, Chemnitz

Chemnitzer Informatik-Berichte

ISSN 0947-5125

Herausgeber: Fakultät für Informatik, TU Chemnitz
Straße der Nationen 62, D-09111 Chemnitz