Uncertainty Quantification in Security Aware Data Pipelines

Alberta O. Dadeboe

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Paul K. Ampadu, PhD, Chair
Angelos Stavrou, PhD
Yang Yi, PhD

May 9, 2025
Blacksburg, Virginia

Keywords: uncertainty quantification, security, provenance, entropy, sensitivity analysis

Uncertainty Quantification in Security Aware Data Pipelines

Alberta O. Dadeboe

(ABSTRACT)

With the recent rise in connected devices through the Internet of Things and interconnected cyberphysical systems, the diversity and volume of data have expanded. Proper management of sensitive information collected and processed through data pipelines is crucial. Traditional data pipelines usually perform error analysis of the final pipeline output after a detection model. As a result, they miss malicious attacks or data corruption that occur earlier in the pipeline. Providing assurance of security throughout all stages of pipeline processing can improve credibility at a more fine-grained level. This thesis introduces a combination of data pipeline augmentation capabilities aimed at estimating the uncertainty of computations with constant monitoring of trends in shifts in data at every pipeline stage. The proposed framework integrates uncertainty quantification (UQ), data provenance tracking, sensitivity analysis, and tunable alerts to understand parameter influence on function outputs, methodically detect potential corruptions, maintain a meticulous audit trail, and prompt observers during suspicious activity. This contribution advances conventional data pipeline anomaly detection by providing combined fault-sensitive execution and full-fault traceability with continuous estimation of uncertainty for each pipeline stage.

Uncertainty Quantification in Security Aware Data Pipelines

Alberta O. Dadeboe

(GENERAL AUDIENCE ABSTRACT)

In practically every industry in the world, data is a commodity that is integrated in almost every operation. This data is constantly moving between different manipulations to make observations for drawing conclusions and assisting in decision making. The combined movement and manipulation constitutes data pipelines, with some manipulations mimicking real life systems for isolated testing and modification.

Considering the fact that most data pipelines handle data from different sources and may be private or sensitive, the security of these pipelines is of utmost importance. Also, companies and individuals who rely on the outputs provided by data pipelines are in need of some kind of guarantee that the results are reliable.

Just like surveillance precedes threat response in normal security operations, active monitoring of changes and trends in the data pipeline during operation is necessary for informed reactions and subsequently improvements to the data pipeline operation. This work explores the use of uncertainty quantification, a kind of confidence measure, to provide assurance on the reliability of every computation stage in a pipeline. The quantification of uncertainty provides all involved parties with some understanding of the quality of the outputs they receive from each data evaluation made within the pipeline.

The architecture proposed in this work provides continuous feedback on how uncertainty and data statistics change from stage to stage in the pipeline. This coupled with an alert system ensures timely attention is drawn to potential attacks aimed at corrupting the data or computations of stages in the pipeline.

Acknowledgments

I would like to extend my gratitude first to God and to the amazing supervisory team: Prof. Paul Ampadu, Prof. Angelos Stavrou, and Prof. Cindy Yi Yang for their guidance and direction in making this work possible through their review responses and presentation feedback. My utmost appreciation also goes to the members of the A2 Labs and ThinkSense, Inc. team, especially to Farzaneh Mansourifard and Shridatt Sugrim. Research collaboration with the team proved to be seamless and productive.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Within Security Operation Centers (SOCs), personnel efforts, when appropriately supported by relevant tools, can improve security efficiency by improving threat prevention, detection, and response. An IBM report on global security operations [1] obtained from SOC team members spanning 10 countries showed that 46% admitted to a an increase in the average time spent to detect and respond to security incidents over the past 2 years before the survey was conducted in 2023. This emphasizes the need for improvements in security supportive technology to make the work of such teams more efficient especially with the increase in attack technique enhancements.

- Challenge 1: According to the 2024 Klynveld, Peat, Marwick, Goerdeler (KPMG) SOC Leaders' Perspective Cybersecurity Survey [2], almost a third (32%) admit that their team experience difficulty assessing the severity of vulnerabilities. Data pipelines need to provide some quantifiable measure of integrity for every computation made by each component or stage.

- Challenge 2: Log management analytics takes the second position in the top 3 most important services and solutions for SOCs in organizations with 76% of SOC leaders indicating it as important [2]. In the scenario where an attacker makes an attempt

to corrupt computations made in the pipeline, there needs to be an audit trail to determine the target instances and timelines as well as an alarm to trigger response to resolve damages and identify the possible vulnerabilities that were exploited to prevent further attacks.

- Challenge 3: The report in [1] with regards to alert response estimates that SOCs spend 32% of their day resolving non-threatening incidents. It also reports that more than 1 in 8 SOC members report an 81% slow down in their threat response times due to manual threat investigation. This waste of resources can be alleviated by introducing tunable elements that allow analysts to triage alerts in order to prioritize more menacing threats.

- Challenge 4: Mechanisms for guiding the optimization of pipeline operating conditions such as resource configurations, data propagation and dynamic input handling of individual functions would be useful in upgrade or maintenance efforts.

1.2 Problem Statement

Data pipelines automate the movement and transformation of data between systems, from raw data collection to a usable format for analysis or decision-making. Depending on the use case for a specific pipeline, critical information can be found in this information transfer chain and are prone to corruption as a result of security vulnerabilities with the pipeline or with the data itself.

Without continuous updates on the trustworthiness of computations throughout the pipeline, it is impossible to determine if results should be trusted for subsequent processing phases. Additionally, any lack of automated calculation and chronological recording of data shifts to facilitate the existence of an audit trail presents a challenge for the proactive neutralization

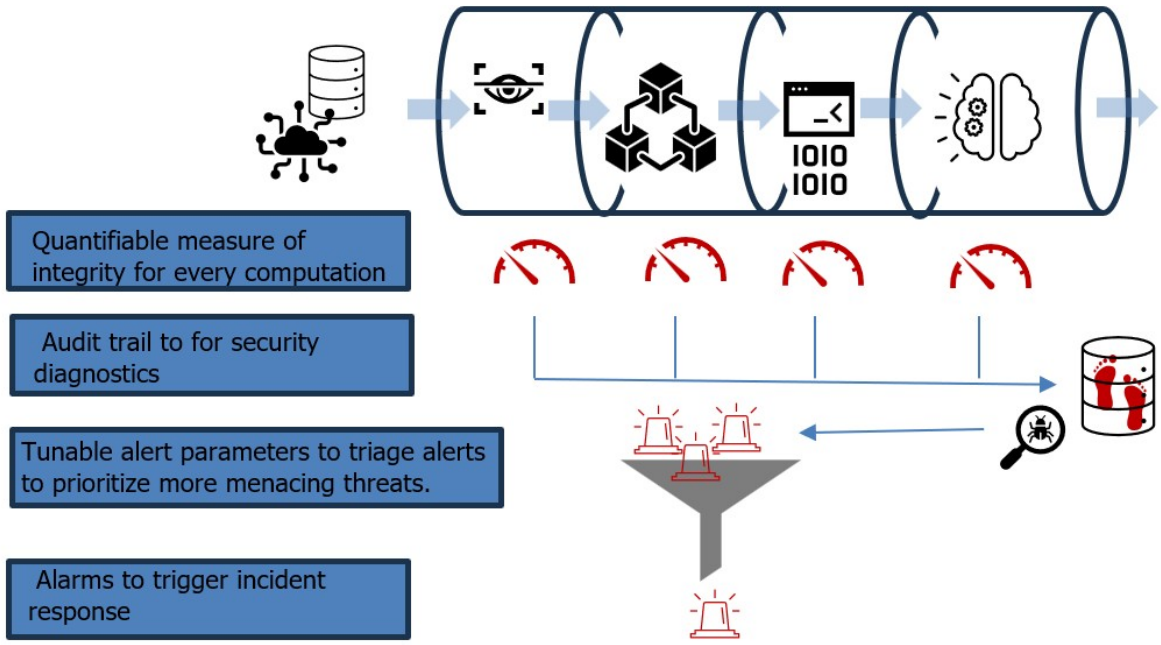


Figure 1.1: Illustration of Proposed Solutions

of attacks.

1.3 Approach

To address the issue of the need for reassurance on the integrity of computations in pipeline stages, this work proposes providing a measure of confidence in the form of an Uncertainty Quantification (UQ). Entropic measures of UQ in addition to shifts in data are computed for every pipeline stage output distribution post-computation. These values are observed for sudden changes of inconsistencies to determine instances of anomalies. As explained in the methodology, mechanisms have been put in place to control the extent to which attention should be placed on observed changes to warrant investigation.

The problem of a reliable audit trail is resolved in this approach by establishing a chain of provenance which captures the metadata associated with the data shifts in a keyed format

that makes it possible to trace the specific change points responsible for any abnormality. Additionally, this approach provides an avenue for active improvements to function computations by analyzing the impact of parameters on the function output to determine informed optimizations. The optional sensitivity analysis feature allows users to understand the influence of certain parameters on function outputs to independently modify the pipeline for improved performance based on their own discretion. The discussion of these mechanisms is summarized in Figure 1.1and provided in this thesis which is outlined as follows:

Chapter 1 emphasizes the motivation behind this work, the associated challenges, and the approach proposed as a solution.

Chapter 2 provides background information on the relevant topics related to data pipelines and uncertainty quantification to provide the reader with a foundational understanding. It delves into topics such as data ingestion and integration techniques including ETL, ELT and other derivatives. It also introduces a baseline understanding of the general framework of data pipelines and the idea of workflow orchestration for establishing and managing data pipelines. A general framework for uncertainty quantification is also discussed in this chapter with a brief categoric explanation of techniques.

Chapter 3 is a summary of the state of the art with respect to data pipeline security. It begins with approaches related to prevention/deterrence, tolerance, and recovery and highlights existing approaches relating to effective monitoring which this thesis is based on.

Chapter 4 highlights architecture for the implementation of suggested solutions and details the experimental setup to drive the derivation of results. It breaks down the methodology for the pipeline configuration with a demonstration of the workflow, the framework for computing uncertainty and tracking statistics, and the structure for analyzing sensitivity. The alert framework and the mechanism for controlling the volume of alerts is provided here. The dataset used for testing and results for all experiments are included in this chapter.

Chapter 5 is a discussion of the key elements in this work and how it compares to related

solutions in the literature. The chapter also provides a direction for future improvements in a different application.

Chapter 6 concludes and summarizes the impacts of the work on data pipeline security.

The discussions in this work is a combination of material from work done and published in the project report for the funded project and in the related research paper [3].

Chapter 2

Background

2.1 Data Pipelines

The rise in demand of Software as a Service (SaaS) and large amounts of data for the training and testing of machine learning models has necessitated the need for data pipelines to ease analytics [4]. Additionally, majority of cyber physical systems, physical components are monitored by extracting and making observations on data transferred between and within subsystems. For a more streamlined process, models of the system can be made and the data managed within a data pipeline. A data pipeline is an chain of interconnected processes that begins with data ingestion, continues with operations such as aggregation, transformation, integration and usually concludes with decision making. They provide automation, scalability, and the capability to experiment on possible modifications to improve without direct testing on the physical system.

2.1.1 Data Integration

Modern systems have data pooling from multiple sources making it necessary for a reasonable technique that allows for the integration of all these data sources for further processing. The integration process makes use of three main procedures (Extraction, Transformation, Loading) to make all required data available in an accessible data warehouse. Extraction

and load are collectively referred to as data ingestion [4]. Transformation was introduced to allow the data to be modified into a much easily analyzable format at the destination.

- Extraction: As the first step in the data ingestion process, extraction involves accessing, profiling, and copying the source data for further manipulation [5]. The ownership of every data source provides an understanding to the extent and mode of access [4]. This influences whether or not direct access is given, the level of granularity, and the ability to customize the selection during ingestion [4].

- Transformation: This involves applying changes to the data to meet the operational requirements of the target data pipeline. The aim is to improve the quality of the data and make it more suited to a predetermined schema. This includes conversion to standard formats, eliminating duplicates, and modifying the representation (adding new features, assigning more descriptive names, etc.) [5, 6].

- Loading: The transfer of data from a source to a readily accessible target destination for processing in a pipeline concludes with the loading step. The transfer process can be instantaneous where all the required data is transferred to the destination, described as full loading. Alternatively, the transfer can be periodical where recent data is continuously added incrementally [6].

ETL versus ELT

The order of Transformation and Loading is determined by the data engineers based on whether or not data should be made available in a raw or modified state when made available for subsequent pipeline operations.

- ETL: This approach structuring and modifying the data from the source after extrac-

tion and making it suitable for analysis in the target pipeline. The final result of this structured data is combined into a data warehouse. Data warehouses contain data that is optimized for analysis [4].

- ELT: This architecture involves a mechanism where data is pooled from diverse sources and are not modified in any way before being loaded and made available in their raw states for pipeline processes. In this approach, the combined storage of pooled raw data forms a data lake [4]. A data lake constitutes data that is not structured for the interest of a particular data pipeline analysis [4]. In ELT, data is extracted and loaded into a data lake before transformation occurs within the data pipeline [4, 6].

There are pros and cons to either integration architecture and the selection criteria is based on a specific use case. ETL controls the data ingestion process by ensuring data quality and initial error control in transformation. Unfortunately, this introduces limitations in terms of delay since the interim process of transformation before loading takes time depending on the computations required. This delay can be further exaggerated in the event of an error in the transformation logic. Once any lapse is detected, all previously transformed data will have to be re-extracted and re-run through the transformation logic after being fixed [6]. To resolve these issues, the ELT approach was developed. Saving the transformation as a part of the post-ingestion analysis provides flexibility and widens the options [4] for how the raw data is modified. This means that in the event of an error in the transformation logic or if a different logic needs to be tested, the time-consuming extraction process does not have to be repeated [6].

Another architecture described as EtLT, aims to combine the benefits of both, allowing some level of transformation before loading to make the data conducive and easily manageable for loading and subsequently more-refined transformation within the pipeline [4]. Basic

pre-loading transformation computations may include basic data cleaning like eliminating corrupt or incomplete data, fixing wrong formatting, or masking sensitive data [4]. Multiple adaptations of ETL and ELT architectures have been explored in the literature to encourage flexibility while maintaining high performance.

2.1.2 Data Pipeline Infrastructure and Maintenance

The components of a pipeline are dependent on the specific use case. Beyond the basic brief data integration pipeline architecture, there are additional functionalities that may be relevant to suit specific scenarios. Popular examples include a pipeline for machine learning application which typically involve integration and ingestion, pre-processing, training, and testing. Pipelines are also setup for data governance to ensure the secure exchange of information from distributed sources. In such an application, encryption, decryption, and reasonably isolated storage techniques become critical stages in the pipeline [7]. Regardless of these differences, all pipelines rely on some common infrastructural sequence.

Data Pipeline Generic Architecture

Every pipeline is made up of data sources (usually a diverse range of types), one or more computation steps/stages composed of algorithms/functions which rely on parameters, and data outputs/sinks [8]. The orientation of this could be linear with each stage executed once for every data sweep. A complex orientation on the other hand could have some steps overlapping or being executed in a loop based on certain imposed conditions [8].

Figure 2.1 illustrates a comprehensive generic architecture of data pipelines. As not otherwise mentioned, this comprises of a combination of descriptions provided in [4, 8, 9]

Data Sources: Available data for pipeline processing can be organization-owned or derived

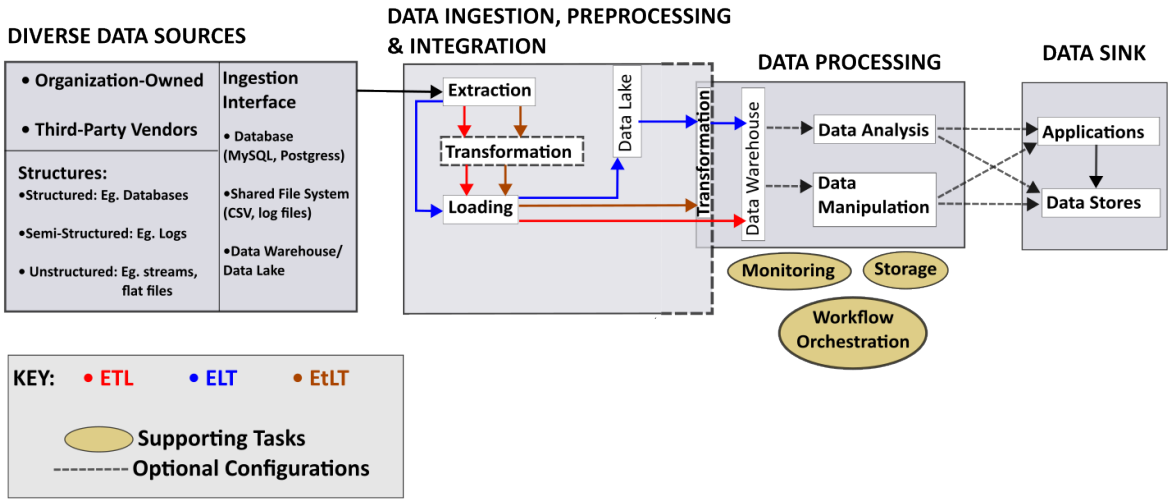


Figure 2.1: Generic Data Pipeline Architecture

from third-party vendors.

Organization-owned data have fewer restrictions to the amount, type, granularity, and the regularity of access. Limitations are very likely to be enforced by system administrators to control the movement of data especially with regards to the sensitivity and security risks involved.

Third-party vendors tend to have a lot of restrictions to data access due to proprietary barriers, privacy concerns, and purchase privilege control besides the more common security risks. The avenue from which the data is ingested is described as the ingestion interface. A variety of options exist some of which include databases behind other applications (eg. MySQL, Postgres), some shared network file systems (consisting of logs, comma-separated value (CSV) files, etc.), data lakes, or data warehouses [4]. The format in which this data is made available may be described as structured, semi-structured, or unstructured. Data is considered structured when it is represented in a manner that is organized and easily-readable with easily-traced components regardless of the time or volume. The less structure the formatting has, the more effort is needed to extract relevant elements for subsequent processes. Optimization of the extraction process to approach poorly structured data is

constantly evolving.

Data Ingestion, Preprocessing, and Integration: Transporting data from the source ingestion interface for optional preprocessing and for integration constitutes ingestion. The ease of this process is dictated by what is known as the 3Vs of big data; volume, velocity, and variety [10].

The more content there is to ingest and integrate, the more time and resources are expended to accommodate the large volume. An increase in volume does not necessarily imply high quality [4].

Variety here refers to the assortment of data representations. Semantic heterogeneity associated with the diversity of data sources from different systems poses a big challenge to the integration of data. This is often due to the utilization of different structures, formatted terminologies across various data sources [11, 12].

Handling of data can also be done at different speeds or frequencies. Resolutions of the velocity of ingestion and processing ranges from real-time and near real-time to periodic and batch levels [12]. After ingestion, the integration procedure can involve a bit of preprocessing through transformation before making the data available for processing. The options include the use of either the ETL, ELT, or EtLT architectures as detailed in 2.1.1. Pre-transformed data is stored in a data lake while post-transformed data is appropriately stored in a data warehouse. The difference between these two is explained in 2.1.1.

Data Processing: This stage is where the data is actually utilized in an applied setting.Processing may or may not result in alterations to the data.

Observations can made through analysis to provide insights on the source or transport channel of the data. Identified trends are mainly used to assist in decisions on the target application. Modifications can also be made on the data through one or more functions or algorithms to generate new information for inferred insights.The order of execution of com-

putations is dependent firstly on the desired end result then by the optimization outcome[8] prioritizing speed and resource consumption.

During processing, supportive tasks like intermediate monitoring and storage are constantly being performed to identify possible irregularities and also keep evolving records. The maintenance of the order and layering of the computations is managed by workflow orchestration tools discussed later in this section.

Data Sinks: The final output(s) of the pipeline is/are reconciled at the data sink. The output is either fed directly into another application or stored for subsequent assessment.

Workflow Orchestration

In simple terms, workflow orchestration is the process of defining the order and nested hierarchy of multiple tasks for the automated and coordinate running of a data pipeline. Automation is made possible with meticulous scheduling and the centralized management administration and management promotes a streamlined pipeline process with high efficiency [13]. There are tools available that provide engineers and analysts with capabilities for managing a pipeline efficiently. The general framework of these tools are composed of basic elements that combine to form a directed acyclical graph described in Table 2.1 [14].

Table 2.1: Elements of a Workflow Orchestration Tool

Element	Description
Tasks	Functions that operate on parameters to generate outputs
Flow	A series of tasks running as a single unit
Schedules	Interval specifications for tasks and flows
Deployment	Metadata dictating how to run a flow

The orchestration consists of a well-outlined protocols for execution of computations in a pipeline. Typical guidelines include but are not limited to the following as derived from [13, 14]:

- Configure all tasks in entire workflow with all required parameters
- Ensure centralized monitoring to allow unified control and management
- Enable automatic re-execution mechanisms to account for issues with tasks not connecting with sources or targets.
- Log all event executions to ensure traceability for diagnostics
- Put contingency measures in place in case of faults or failures in computations to minimize errors.

Data Pipeline Maintenance

Beyond building data pipelines, they require monitoring and maintenance. In order to establish a reliable maintenance mechanism, there is a need to understand the factors that influence pipeline performance. According to [9], the influence can be categorized into 5 themes: Data, Development and Deployment, Infrastructure, Lifecycle Management, and Processing. Within each of these themes, there are factors attributed to each. Factors belonging to development and deployment, and processing themes were reported to be the most influential based on the evaluation of experts in the field.

To ensure proper maintenance of data pipelines, challenges associated with these influencing factors have to be addressed and included in maintenance frameworks.

With regards to data quality, the common challenges faced by data engineers include the loss

of data files in transmission and misidentification or mislabeling of content [7]. Maintaining the quality of data means putting measures in place to detect incomplete transmissions and identify inconsistencies in related content. In the field of big data, integration is also a major pain point for the data theme. Major challenges stem from the heterogeneity of sources due to varying data formats and terminology, scaling to accommodate real-time processing, and maintaining privacy [12]. Solutions range from establishing ontology frameworks for structured semantics to enabling parallel processing of different data across multiple nodes to address issue with heterogeneity and scalability [12, 15] Addressing privacy challenges in particular has seen the proposal of anonymity models which use encryption to prevent linking sensitive information across different sources [16]. Other approaches rely on ascertaining the reliability of data providers by filtering out dishonest providers [17].

Challenges facing development encompasses issues with ensuring completeness with very detailed specifications, achieving a wide scope of testing to ensure full functionality, and drifts in data and code equality between development and production [9]. Timely review of code versions to ensure compatibility and thorough testing to maintain full expected functionality are remedying strategies to consider.

As inferred from the definition, infrastructural challenges are usually associated with the technological tools and environment used for servicing a pipeline. Quality of the tools used depends primarily on their compatibility, reliability, and operational accessibility [9]. Ultimately, the selection of tools should be based on independent assessments based on the use cases and their specific objectives and specifications.

To ensure the long-term sustenance of a pipeline, monitoring its performance overtime as well as the continuous optimization of resources is vital. This can be manifested with consistent updates, version control of software, code or configurations [9].

Finally, ensuring optimal computational capabilities of the pipeline processes in dependent on factors relating to proper sequencing, synchronization, and configuration, all of which can be attributed to proper automation procedures. The quality of data and infrastructure is also relevant to the outcome of the processing stages [9].

2.2 Uncertainty Quantification

The British statistician George Box famously said, "All models are wrong, but some are useful." This means that no model can fully capture the complexities of the real world. The imperfections are due to adjustments made to accommodate available resources and possibly imperfections in the data ingestion process can cause errors in data collection.

It is therefore important to quantify how imperfect a model is or how useful its output to provide a user with some measure of confidence. Uncertainty Quantification (UQ) helps provide an understanding and a measure of the uncertainties in models, allowing us to assess how useful they are. UQ can be classified as aleatoric, due to intrinsic randomness or epistemic, resulting from incomplete knowledge [18]. The goal of UQ is to estimate the statistics from the distribution of the output such that it reflects the uncertainty at the input and vice versa. Forward uncertainty quantification, also known as uncertainty propagation, progresses from input to output while Inverse UQ progresses from output to input.

2.2.1 Uncertainty Propagation

By simple definition, forward UQ is an estimation of uncertainty in the computational output of a model. To compute uncertainty, the procedure is summarized as follows:

- Derive the distribution of input parameters derived from educated guesses/estimates

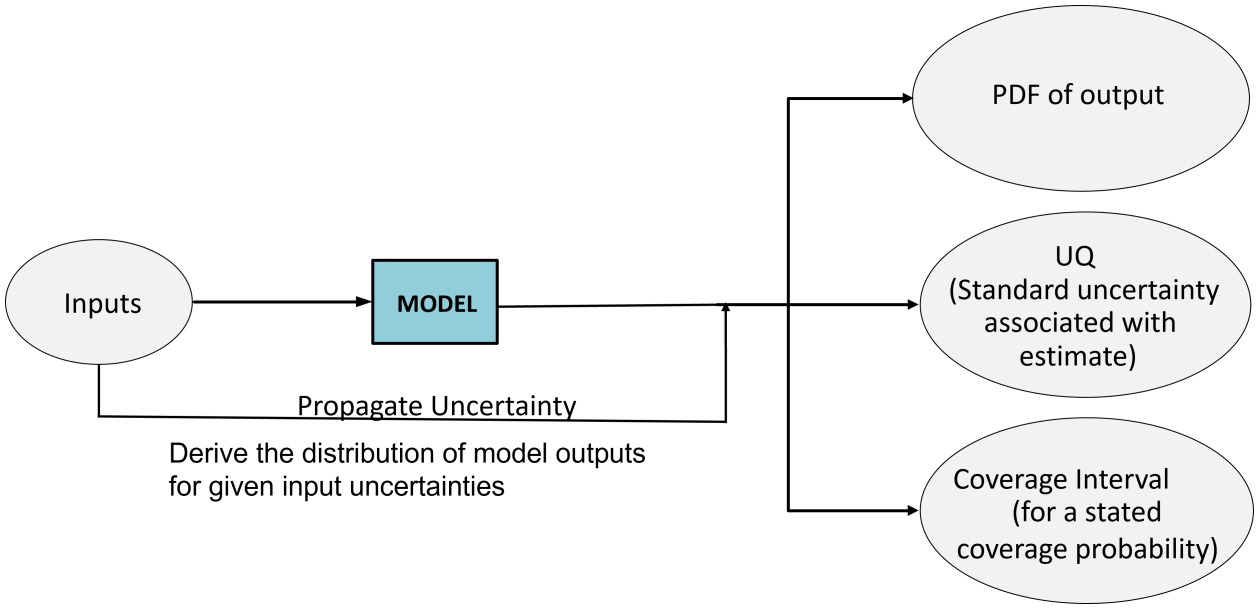


Figure 2.2: Basic Procedure of Uncertainty Quantification

or a series of actual measurements.

- Evaluate the model function based on the distribution of the input parameters.
- Identify useful statistics including mean, and variance from the series of output results from the model evaluation to describe the output probability density function.
- Derive the uncertainty through statistical inference from the probability density function of the output(s)

As summarized in Figure 2.2, given a model which takes samples of inputs, outputs for these samples are produced. Over a given sampling window, it is possible to derive a probability distribution for the outputs, the UQ from this distribution, and the coverage interval for a given coverage probability [19].

The challenge is with how often model evaluations have to be performed to derive the statistical measures for describing the output probability density function (pdf).

Mathematically, given a model of system $\mathcal{Y} = M(\mathcal{X})$ with uncertain input parameters $X \in \mathcal{X}$ and probability density function $p_x(x)$, estimating statistics such as expectation $E[\mathcal{M}] = \int_{\mathcal{X}} \mathcal{M}(x)p(x)dx$, variance $V[\mathcal{M}] = E[\mathcal{M}^2] - E[\mathcal{M}]^2$, and the output pdf $p_y(y)$ [18]. Since the integration cannot be done over all possible input values in the distribution, the output distribution has to be characterized by using samples within the input distribution.

For reliable results, the Monte Carlo technique requires the model to be evaluated a very large number of times for a given input parameter set to derive a series of model outputs from which statistical measures are derived. It performs model computations for several samples of an input parameter based on the distribution. For n samples of an input parameter producing outputs $\{y_1, y_2, ..., y_n\}$, the mean can be estimated as $E[Y] = \frac{1}{n}\sum_{i=1}^{n} y_i$

If the model is computationally expensive, the Monte-Carlo technique would consume a lot of resources. Different adaptations of the Monte Carlo techniques have evolved to alleviate this computational cost. The quasi-Monte Carlo works by strategically selecting input sample sequences using variance reduction techniques that aim at expanding the coverage of distribution with fewer samples [20]. Other adaptations include the multi

The approach used here allows non-intrusiveness due to

To what extent can the uncertainty be propagated? What are the limitations?

There is a limitation associated with what information is required versus what is available including the fact that propagation might not include all possible contributions of the inputs to the output distribution. In addition, propagation also takes a lot of resources (deterministic forward simulations have high computational expenses). Also, not all input uncertainty has to be propagated. Some uncertainties can be considered negligible, eliminating the need to always compute a perfect UQ to evaluate the model's performance.

How can these limitations be overcome?

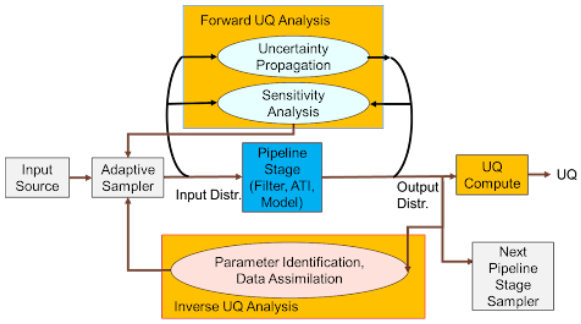


Figure 2.3: General Framework for Uncertainty Quantification

- The resource burden for the propagation can be reduced by omitting inputs that do not make so much of a difference in the final UQ.
- To make meaningful selections, sensitivity analysis can be performed to develop a ranking of the importance of input parameters according to their contribution to output uncertainties
- Inverse UQ propagation can be utilized to be able to identify unobserved or variable model inputs as well as fixed but unknown forward model parameters like hyperparameters that determine the distribution of inputs to the model. Examples include sampling rate, learning rates in optimization parameters, and grid resolution

All these considerations combined can deduce a general framework for computing UQ as shown in Figure 2.3

Input data is ingested and sample in fixed/variable time intervals to derive the input distribution. The framework propagates the uncertainty from these input distributions to the model output with uncertainty propagation. Then with sensitivity analysis, the framework analyzes both input and output distributions to identify highly contributing inputs and modify the sampling accordingly.

From the output distribution, the framework can identify trends to highlight and quantify

the uncertainty. This can be followed up with inverse UQ analysis to identify unknown or unobserved hyperparameters through parameter identification and data assimilation to integrate observed output data into the computational model to improve the estimation of these unknown model parameters.

2.2.2 Sensitivity Analysis

Sensitivity Analysis consists of a study of the impacts of the input parameters on the outputs of a mathematical model or system [21]. This provides us with a ranking of different inputs and their corresponding importance rates derived from their contribution to uncertainties at the output, motivating strategies for system optimization which can include iteratively controlling the uncertainty of input factors or completely ignoring those with very low importance rankings for less dimensionality [21].

Types of sensitivity analyses techniques:

- Local Sensitivity Analyses: Analyzing the effects of local changes of a parameter in the system [21]. ie. For input parameters i = 1,2,..m, find dy/dxi of the model response y = f(x) [21].
- Global Sensitivity Analyses: Evaluate the entire parameter space and measure the average contribution of input variables to the output [21]. Makes use of statistical methods [22]. Global sensitivity analysis seeks to decompose a complex model—or function—into a sum of a constant plus main effects plus interactions [23]. Methods:
  - Variance-based methods - Use variance as a measure of the importance of an input parameter in contributing to the overall uncertainty of the response [21].
    - Monte Carlo - detailed in uncertainty propagation section

    - Sobol Method: The complex model is decomposed into constant function and interdependent functions [23]
    - Correlation ratio-based methods
- Moment-independent importance measures - sensitivity measures based on discrepancies between density functions, cumulative distribution functions, and value of information. Example: Density based sensitivity, cumulation based sensitivity
- Reliability-based statistical analyses

Chapter 3

Review of Literature

This section comprises a review of the state of the art approaches with regards to security in data pipelines followed by a focus on anomaly detection techniques that utilize uncertainty quantification.

3.1 Security-Aware Data Pipelines

The self-driven control of data flow and processing in a well-established pipeline increases the tendency for an undetected vulnerability to be exploited continuously for multiple cycles until noticed. Once the trustworthiness of a pipeline has been disrupted, the data is prone to unauthorized exposure, computations are corrupted, and resources wasted. Proper frameworks for the three key computer security objectives, confidentiality, integrity, and availability need to be present in any pipeline considered to be secure. As described with the taxonomic illustration in Figure **??**, the mechanisms that ensure that these objectives are met for trusted functionality include but are not limited to access control and obfuscation/masking for confidentiality, authentication and non-repudiation for integrity, and accessibility and traffic control for availability.

3.1.1 Confidential Computing

Reducing the attack surface of systems by providing a clearly security boundary to isolate sensitive data and computations is what defines the concept of confidential computing [24]. Some researchers propose the setting up of a hardware, software, or hybrid trusted execution environment (TEE) for secluded executions. TEEs are isolated execution environments in a processor that ensure the protection of data at rest, in transit, or in use [25]. Authors in [26] propose a hardware approach which utilizes enclaves provided by Intel SGX TEE. They take advantage of caching to reuse trusted runtimes to reduce the cost of initialization and effectively reduce execution time. [27] discusses solutions geared towards the protection of machine learning pipelines from confidentiality attacks such as Data Reconstruction Attack (DRA) where attackers observe models or their gradients to reconstruct input data, Attribute Inference Attacks (AIA) where attackers estimate the value of user private properties within training data, and Membership Inference Attacks (MIA) where attackers track the existence of specific data instances in training data. In the Machine Learning (ML) pipeline, both the server and the client can be considered untrusted and as such have TEEs setup for confidentiality. Server-side hosts providing inference or training as a service (IaaS and TaaS) have to provision the data secretly into the TEE. If the model has external ownership, it also has to be provisioned secretly into the TEE [27]. Clients are also required to deploy models securely into TEEs for on-device inference and personalization to protect the intellectual property of external model owners [27].

3.1.2 Access Control

As a mechanism for ensuring confidentiality, access control aims to guarantee that only authorized entities have access to protected data or processes. [28] propose a data governance-

based access control architecture to facilitate the protection of sensitive data attributes throughout big data analytics pipelines, from ingestion to analytics. The movement of data is regulated by policies based on the identity of the user/service with their associated privileges and annotations on the ingested data, During ingestion, the policy-aware ingestion is based on the ETL approach where transformations are selected and triggered by the access control policies and range from pruning and reshaping to full/partial anonymization. The flexibility of policies vary, some allowing full access with no transformation, and others enforcing temporal and spatial transformation. During training for analytics, multiple different models are trained on different datasets generated based on different intensities of access policy enforcement at ingestion. Results are supplied from either model based on the privileges of the requesting user.

Multi-party Computation (MPC) is a privacy-preserving technique that has been developed for data confidentiality with pipelines dependent on heterogeneous sources. It involves providing alternative values for sensitive data to the parties utilizing shared information that is not theirs in lieu of the actual values. In [29], researchers employ imputation in an MPC framework including mean, median, regression, and k-Nearest-Neighbor for the preprocessing stage in ML data pipelines.

3.1.3 Authentication and Non-repudiation

A blockchain-based framework was proposed by [30] which infused signature chains with blockchain receipts for authentication and accountability. The approach comprises of a smart contract that enforces identity management and key distribution for non-repudiation and signature verification to prevent impersonation and denial of involvement. Additionally, a reputation system based on frequency of active integrity provisioning to discourage intentional delayed data transmission.

3.1.4 Accessibility

In active attacks such as denial of service which target data or pipeline resource availability, techniques have been developed to resolve delays or disruptions. [31] strategically uses a hybrid Storage-as-a-Service model which involves dynamic selection from multiple cloud storage providers. To select a cloud provider, five (5) parameter are typically taken into account; proximity, network performance, impact of encryption for security, user preferences, and cost. For various pipeline deployment scenarios, storage options are ranked using an decision-matrix evaluation algorithm. In a high variability scenario where there is an unpredictable rise in demand from a denial of service (DoS) attack, the dynamic selection prioritizes the prevention of any holdup in applications. This is done by placing the most weight on performance, followed by proximity, impact of encryption, and cost. Another proposition which involves creating virtual models of physical processes, known as digital twins, was developed by [32]. In their approach, replicas of critical components are modeled as digital twins and integrated into a data pipeline with elements for security analysis, monitoring, and high volume data processing. This architecture create an off-premise environment for isolated training of ML models for DoS attack detection without disrupting the running processes of the actual critical infrastructure [32].

Some approaches are an amalgamation of multiple mechanisms. Combining obfuscation, and authentication, authors in [33] propose a blockchain-enabled ML approach with cryptographic elements for secure big data analytics. The framework distributes Elliptic Curve Cryptography keys to device and models nodes for secure communication between entities. Differential privacy is applied by segregating private and non-private features and appending reasonable ”noise” to private data. The integrity of data is protected by blockchains with signature attachments.

3.2 Effective Monitoring in Secure Data Pipelines

According to the 2025 IBM X-Force Threat Intelligence Report, data theft, at 18% incidence, was more predominant since there is pressure for swift exits among attackers. This venture of passive attacks makes anomaly detection more relevant. Effective monitoring is also still relevant for preventative techniques that require pattern identification to attribute to malicious activities. The 2017 Open Web Application Security Project (OWASP) also reported that insufficient monitoring and logging is one of the top ten security issues, which progressed one position higher and expanded to include more failures in the 2021 report [34].

All previously discussed security techniques rely on monitoring to deter, detect, or recover from attacks. Effective monitoring techniques can contribute to achieving all three (See Figure **??**):

- When it comes to deterrence, effective monitoring can help establish a modus operandi, or trend, for particular attacks and embedded those patterns in future defense mechanisms
- The primary benefit of effective monitoring is detection. Anomalies that differ from the norm can be flagged and investigated if monitoring is done effectively.
- Recorded trends observed during monitoring can provide a template to guide recovery efforts after an attack event.

The dynamic nature of data pipelines due to constantly changing inputs and outputs makes monitoring a vital part of security. Even more essential is a quantifiable measure of trustworthiness to attribute to pipeline operations to assess all computations. Uncertainty Quantification does just that. Quantifying uncertainty for each pipeline stage estimates the risk of an inaccurate computation or data flow.

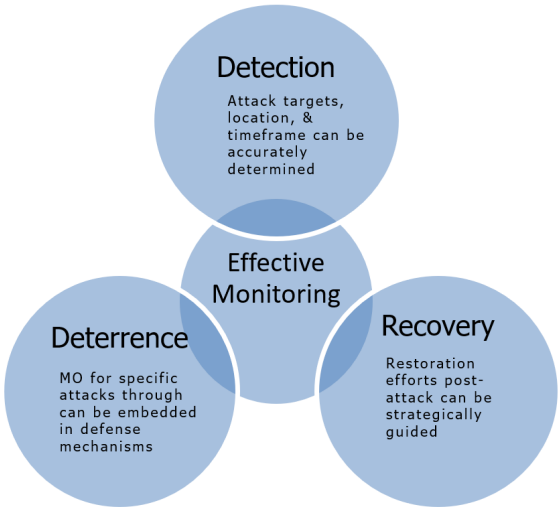


Figure 3.1: Illustration of the Contribution of Effective Monitoring to Attack Deterrence, Detection, and Recovery

In light of this, the effort in this work is directed towards ensuring effective anomaly detection for securing data pipelines through a combination of mechanisms centered around uncertainty quantification. The application for effective monitoring are numerous. In ML pipelines for IoT applications such as driver behavior analysis and traffic flow prediction [35], UQ can be applied to detect attacks meant to corrupt predictions made by the model. For hardware architectures like FPGA accelerators [36], UQ can applied to assess computations made by IP blocks.

Chapter 4

Experimentation and Results

4.1 Methodology

4.1.1 Uncertainty Quantification (UQ)

In the State-of-The-Art (SOTA) UQ, models are handled individually. In data pipelines, doing this might cause uncertainty from prior stages to be lost and not propagated. In this approach, we compute UQ in a stage-wise manner propagating uncertainty from stage to stage. (See Figure 4.1).

Typically, the UQ statistics observed are mean and variance. Our solution includes standard deviation, IQR, and also Entropy for a much wider range of perspective to fortify observa-

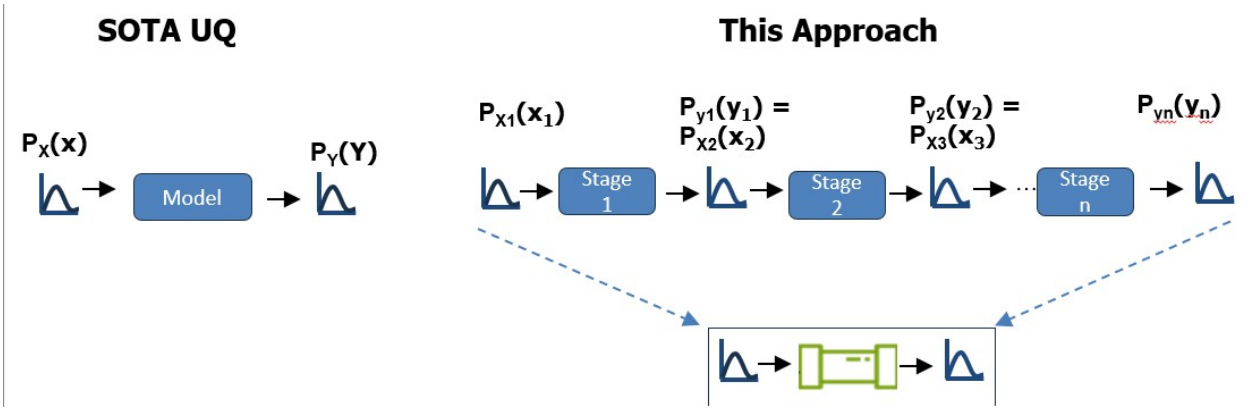


Figure 4.1: Stage-wise Uncertainty Quantification
Uncertainty is propagated throughout pipeline stages

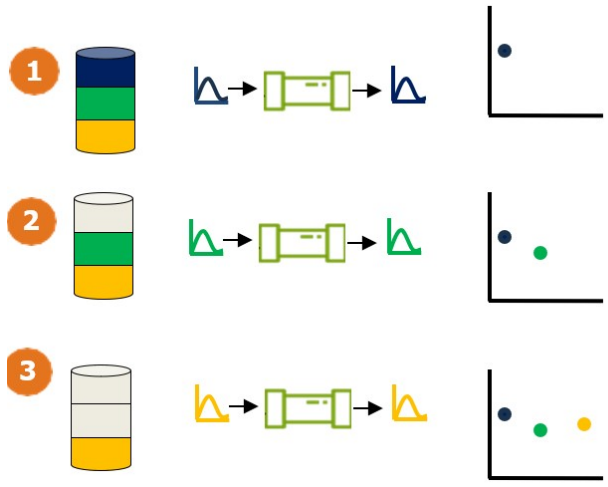


Figure 4.2: Batchwise Data Processing with showing UQ Statistics Recorded and Appended to Plot for Batchwise UQ Observation
Plots are produced for all stages in the pipeline showing the distributional changes across data batches.

tions. In SOTA, an assumption is made on the probability distribution of inputs. Parameters and samples based on this distribution are fed into the model repeatedly to determine uncertainties. Providing a definitive description input distribution is prone to inconsistencies. In our approach, recorded data is fed into the pipeline in batches. For each batch, data is fed into the pipeline and the UQ metrics recorded. This is done for each stage and repeated for all data batches. We observe changes in the plotted UQ statistics across batches to detect distributional shifts for anomalies. This is illustrated in Figure 4.2.

Why are distributional shift observations more reliable? The mean, also described as the expected value of the model output, gives us an idea about the best average estimate for a variable of interest. While the mean provides a location which outcomes might cluster around, it does not provide an idea about how they might spread out. That is when the variance comes in, telling us how much the output varies around the mean. It gives an understanding of how the outcomes typically deviate from the best guess. From face value, interpretations to these two statistics might be summarized as follows:

Small variance implies outcomes closer to the means, creating a low uncertainty, high confidence effect.

Large variance implies outcomes further from the mean or best guess, creating a high uncertainty, low confidence effect.

The problem with relying solely on this interpretation is that consistent corrupt values can produce a consistent average, producing an unreliable mean value. Variances compared to this mean cannot determine whether there is high or low uncertainty.

Description of UQ Stages

The uncertainty quantification (UQ) approach is applied at each stage throughout the data pipeline. (See Table 4.1).

Table 4.1: Description of Individual Pipeline Stages

Pipeline Stage	**Output**
Ingestion: Data from source system	Raw unprocessed mixed-format data.
Filtering: Isolate relevant data features suitable for specific use case	Temperature features/data points
Cleaning: Handle missing data, remove outliers, smooth, and normalize data	Cleaned temperature readings
Aggregation: Resample with varying intervals	Temperature data distributions for varying intervals
Integration: Combine random samples from additional data into existing pipeline data	Temperature readings with newly integrated data
Fault Detection: ML model used for detection faults in system from which readings are being derived	Indication of fault condition

The UQ computations (mean, variance, IQR, standard deviation, and entropy) are represented as UQ1, UQ2, UQ3, and UQ4 which represent the associated uncertainty metrics at

specific modification points in the pipeline.

- UQ1: UQ stats derived post-filtering, which involves isolating relevant temperature values.
- UQ2: Post-cleaning UQ stats after noise and anomalies have been removed from the data.
- UQ4: UQ stats derived after cleaned data is aggregated and resampled at varying intervals.
- UQ4: UQ stats recorded after integration prior to the fault detection model

The framework illustrating how UQ is computed is shown in Figure 4.3. An input source provides data for ingestion into the pipeline for processing with extraction done in fixed time intervals and data fed into the pipeline. Uncertainty associated with these timed inputs is propagated through the computations made in the subsequent pipeline stage to produce outputs from which distributions are established. With sensitivity analysis as an optional feature, both input and output distributions are examined to categorize inputs based on their influence on the function outputs to modify ingestion for optimized performance. UQ statistics are computed from the output distribution from which observations are made throughtout the running of the pipeline to identify trends highlighting uncertainty. As explained in Section 4.1.2, the data provenance chain updates itself by storing the UQ statistics for each stage to ease diagnostics during auditing of the pipeline.

The combination of the described capabilities in this framework enables the quantification and tracking of uncertainty through the pipeline by analyzing shifts in central tendency and data variability statistics.

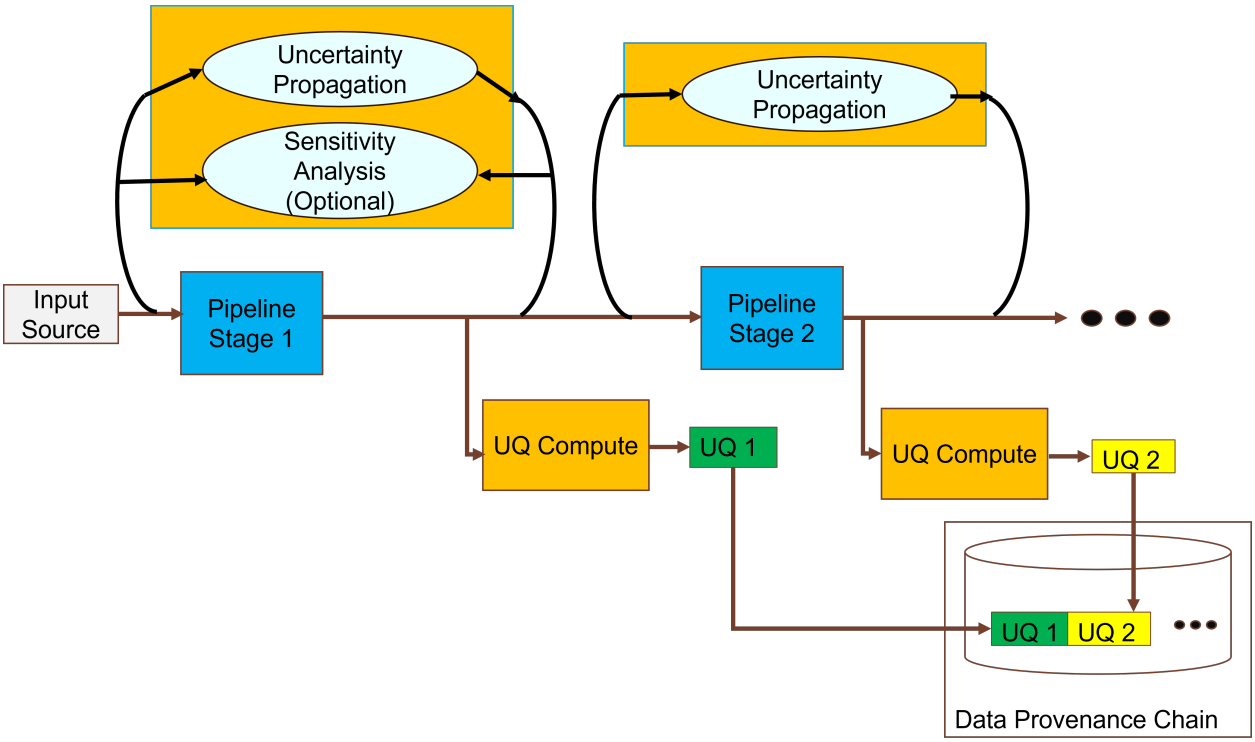


Figure 4.3: Uncertainty Quantification Framework in Pipeline

The deep yellow colors indicate UQ components. Pipeline components with the model of interest are in deep blue

Entropic Measures of UQ

Entropy provides measures for uncertainty based on the probability distribution of random variables. It quantifies the spread or change in readings for Quantities of Interest (QOIs) for various processing cycles [37]. Shannon introduced a measure for entropy derived from communication theory. In this description according to [37], a "word" is portrayed as a sequence of binary digits of length "n". For a combination of multiple words with that same length, the number of elements, N, becomes 2^n This would mean the characterization of one element, depicted by the number of digits, can be described with Equation 4.1 [37].

$$n = log_2 N \tag{4.1}$$

If the initial combination of **N** words were separated into subgroups each having N_i words, the probability of selecting an element from the i-th subgroup is denoted with $p_i = \frac{N_i}{N}$. Multiplying this by the number of digits in that i-th subgroup, $log_2 N_i$, gives you the complete probability of selecting any element in that subgroup. For all subgroups, the total probability is represented as $\sum_i \frac{N_i}{N} log_2 N_i$ which can be simplified into Equation 4.2, representing the average information needed to represent an element in any group. [37].

$$\sum_i p_i \log_2 p_i + \log_2 n \tag{4.2}$$

If we had the full set of words, and not the distributed subgroups, the available information needed to characterize an element is already given as 4.1.

Since information from all possible subgroups is not readily available due to this distribution, the missing information is Equation 4.2 subtracted from 4.1. Entropy or the missing information is then estimated as Equation 4.3 [37].

$$-\sum_i p_i \log_2 p_i \tag{4.3}$$

Sensitivity Analysis

The goal of sensitivity analysis is to produce a quantitative estimate of the effects or contribution that input parameters have on the output of a function. The process of computing sensitivity analysis involves selecting parameters of interest, deriving the corresponding probability distributions, repeatedly sampling selected values from these distributions to be evaluated by the model or pipeline stage, and assessing the output to determine the sensitivity. The derived sensitivities can be compared and ranked based on influence on model output [38]

Most sensitivity analysis methods are variance-based and depend on strategic parameter sampling[39, 40]. The notion that variance is a good indicator of the influence an input parameter has on a function is the motivation for multiple variance-based techniques. A term described as sensitivity index is used as a measurable indication of the contribution an input parameter has on the makes to the result of a model computation. It is the relative contribution of the variance of the parameter of influence to the variance of the function output. Sobol analysis is one of the most widely recognized techniques due to its capability to evaluate the Total Sensitivity Index, a measure of the major influence of a parameter as well as all interactions of any degree related to it [39, 40]. The analysis utilizes variance decomposition to compute the sensitivity indices. The underlying principle behind this involves segregating the terms of the of the output function of the model into a summation of variances using input parameter combinations in increasing dimensionality. (Refer to Equation 4.4 [38, 39, 40]).

$$V(Y) = \sum_{i}^{d} V_i(Y) + \sum_{i<j}^{d} V_{ij}(Y) + ... + V_{1,2,...,d}(Y) \tag{4.4}$$

Equation 4.4. Variance decomposition of model output function

First, the variance of the variables in the function are calculated independently (Equation 4.5 [38, 39, 40]) and as a result of their interactions with other variables (Equation 4.6 [38, 39, 40]).

$$V_i(Y) = V[E(Y|x_i)] \tag{4.5}$$

Equation 4.5. Contribution of variable x_i to the variance

$$V_{ij}(Y) = V[E(Y|x_i x_j)] - V_i(Y) - V_j(Y) \tag{4.6}$$

Equation 4.6. Contribution of the interaction between variable x_i and x_j to the variance

To compute the sobol index, the ratio of the isolated and interactive variances relative to the total variance is evaluated:

- Equation 4.7a [38, 39, 40], represents the 1st order sobol index measuring the independent contribution of parameter x_i to the total variance
- Equation 4.7b [38, 39, 40] represents the 2nd order sobol index measuring the contribution of interactions between x_i and x_j
- Equation 4.7c [38, 39, 40] represents the total order sobol index measuring the total contribution of x^i (both independent and interaction-based) with other parameters

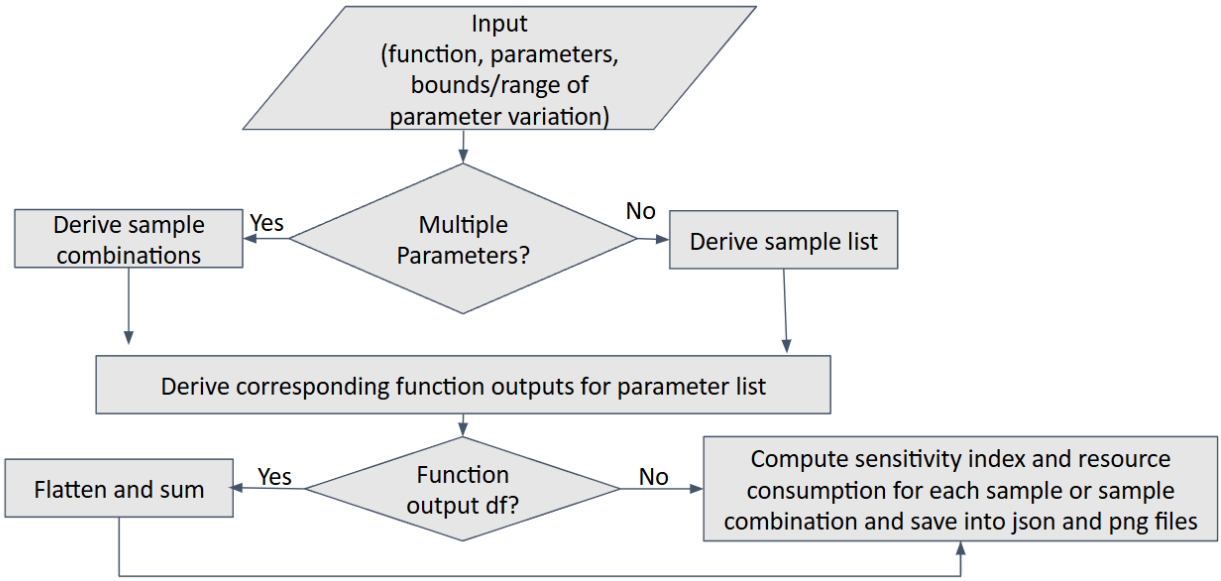


Figure 4.4: Sensitivity Analysis Implementation Flowchart

$$S_i = \frac{V_i(Y)}{V[Y]} \tag{4.7a}$$

$$S_i j = \frac{V_{ij}}{V[Y]} \tag{4.7b}$$

$$S_T = S_i + \sum_j S_{ij} + \sum_{j,k} S_{ijk} + ... = 1 - \frac{V[E(Y|x_{\sim i})]}{V[Y]} \tag{4.7c}$$

Equation 4.7. Sobol Indices

Sobol sensitivity analysis was implemented as an optional functionality within the pipeline. As mentioned in Section 2.2.2, there are two types of sensitivity analysis that can be performed: global analysis which evaluates the effects of different parameters and local sensitivity analysis which evaluates variations of a single parameter. Based on the desired analysis type, different output functions can be derived as shown in Figure t4.4. If there are multiple parameters for global analysis, different sample combinations for these multiple parameters

have to be generated. On the other hand, for local analysis, a list of sample ranges of the same parameter is generated. The corresponding function outputs can then be computed. For our implementation, the sensitivity analysis function performs local sensitivity analysis with the capability of being extended to adjusted to suit global analysis as detailed in Figure 4.4. Function outputs with dataframe formats are accommodated in this framework by flattening and summing to allow for sobol index computation. A variety of format types can be converted into dataframes for the purpose of generality.

4.1.2 Data Provenance Chain

Data provenance provides an avenue for improved diagnostics by providing traceable records of data shifts throughout the pipeline operations. In this approach, the provenance chain is implemented as a keyed container in a JSON dictionary format with unique keys associated with specific UQ statistics for selected features in pipeline stages within specific batch processes. While pipeline computations are underway, computed UQ values are consistently appended to the chain (Figure 4.3).

The ordering of pipeline stages dictates the order in which the provenance chain is updated. In our case, UQ data from the ingestion stage is calculated first (by choice, this is not appended to the chain since it is unfiltered and does not contain specific UQ for relevant features) followed by the filtering stage which has its UQ appended to the chain. Afterwards, the UQ information for the cleaning, aggregation, and integration stages are calculated and appended respectively.

Maintaining and referencing a provenance chain allows for tracking the origin and journey of data shifts through the pipeline, making it possible to perform debugging and anomaly investigation.

4.1.3 Alert Framework

In the event of significant data shifts, an alert system is needed to notify the data engineers' attention to potential system faults or intentional attacks. The framework in this implementation utilizes continuous observations on UQ statistical shifts within the provenance chain to call attention in the event of an anomaly.

Establishing value extremes that should not be exceeded, lower and upper threshold limits have to be provided. The framework references these thresholds when analyzing stored UQ in the provenance chain to determine whether or not an alert is justified.

Mechanisms have been put into place to allow for thresholds to be set either manually or dynamically.

- Manual: The user provides thresholds limits for UQ statistics for all features in each stage.
- Dynamic: When opted for, the system observes prior trends of UQ statistics as a reference for determining the upper and lower limit ranges for the features in every stage. This is continuously revised for successive batch runs. The extent to which the derived range of statistical values from historical UQ trends control subsequent dynamic thresholds is controlled by a buffer value chosen by the user. As observed in results discussed in Section 4.3, the buffer value is directly proportional to the alert sensitivity. The higher the alert sensitivity, the more alerts are produced.

4.1.4 Fault Prediction

A Random Forest (RF) [41] fault prediction model is designed to classify fault condition. Data used for training and testing are obtained from an HVAC dataset [42, 43] described in

Section 4.2. The data was introduced to the model in separate testing and training sections in 30-minute batch sizes, integrating faulty data from a variety of fault scenarios. Testing data included the first one hour of fault and faulty conditions (with 20 randomly sampled data points from the integration stage) while training data isolated the first hour of ingested data.

4.2 Dataset

The selection of the dataset to evaluate the functionality of the framework is inspired by the need to demonstrate some possible effects on cyber physical systems from inferences made from processed data retrieved from these systems due to uncertainty in the data itself or in the processing computations. The Lawrence Berkeley National Laboratory (LBNL) Fault Detection and Diagnostics (FDD) Data Set derived from simulating Fan Coil Unit (FCU) operations in HVAC systems [42, 43], proved suitable for this requirement. The architecture of the Fan Coil Unit, as shown in Figure 4.5 [42, 43], includes valves, heating/cooling coils, air inlets and outlets, and a fan that can be set to three speeds to move cool or warm air into a room to maintain temperature to specific setpoints. It includes data readings (temperature, humidity, flow rate, speed, valve/damper positions, and power consumption) collected in 1-minute intervals and formulated from circumstances involving various fault cases and fault severities and produces. The data set was developed to provide HVAC data during fault-free and faulty scenarios to help develop studies in Fault Detection and Diagnostics(FDD).

As illustrated in Figure 4.6, the UQ statistic, IQR, differs for data from the Normal condition and that for a Fault scenario. This reinforces the fact that statistics associated with the data can be useful indicators of anomaly.

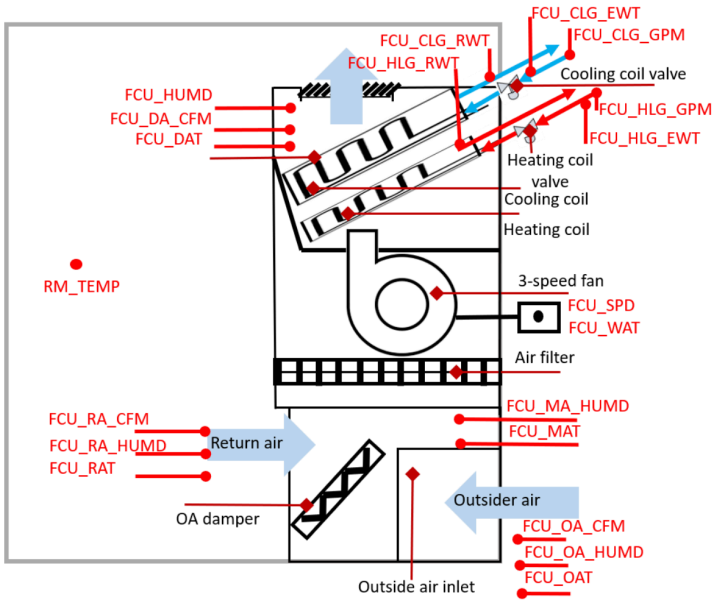


Figure 4.5: Schematic of a fan coil unit serving a room

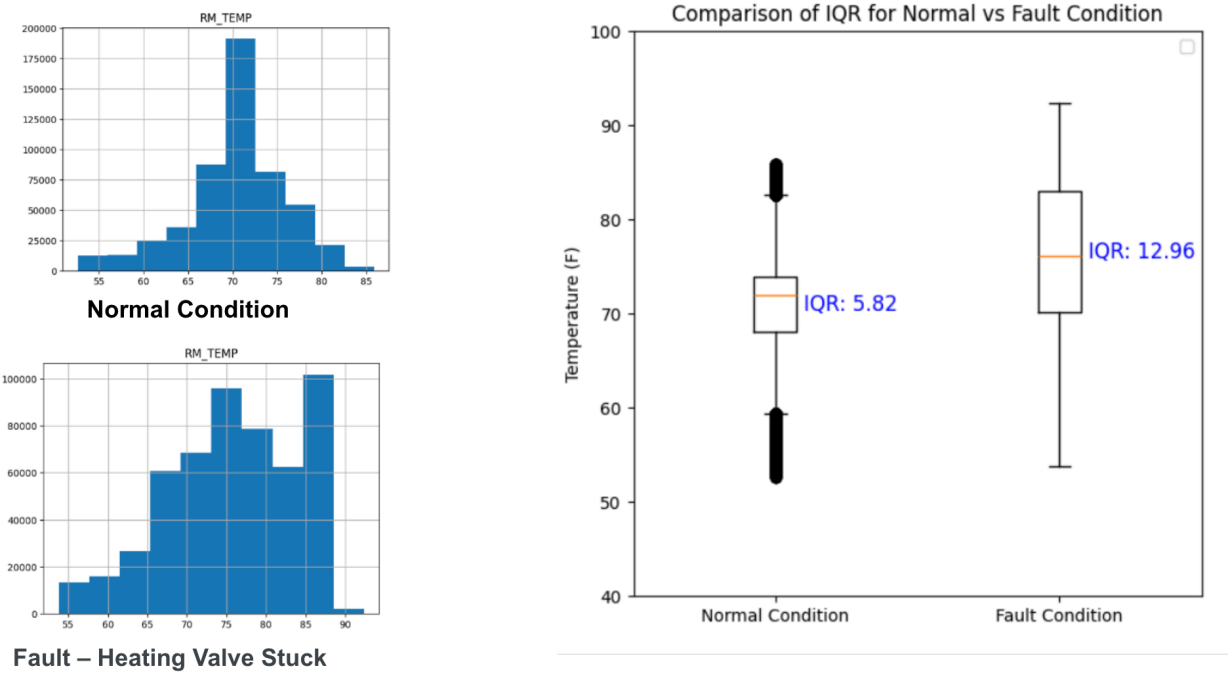


Figure 4.6: Distribution of HVAC Fault Scenario

Table 4.2: Comparison of Workflow Orchestration Tools Across Various Dimensions

Category	Prefect	Apache Airflow	Snowpark	AWS Data Pipeline	MWAA	AWS Step Functions
Ease of Setup [1]	Easy	Moderate	Easy	Easy	Complex	Easy
Scalability[2]	High	Manual	High	High	High	High
Flexibility [3]	Flexible, integrates with data sources	Flexible for batch	Flexible for batch and analytics	Moderate flexibility	Flexible, supports third-party integration	Less flexible, AWS-focused
Monitoring and Logging [4]	Built-in Prefect Cloud, real-time	Custom, less real-time	Built-in, requires tools for real-time	Built-in CloudWatch	Built-in CloudWatch, requires configuration	Built-in CloudWatch
Community and Support [5]	Small, growing	Large	Small, growing	AWS-only	Large	AWS-only
Version Control [6]	Built-in	Not built-in	Not built-in, external handling	Not built-in, external handling	Not built-in, external handling	Not built-in, external handling
Code Complexity [7]	Low	High	Moderate	Moderate	High	Moderate
UI/UX [8]	Built-in Prefect Cloud	Built-in web interface	Built-in for SQL and job orchestration	Built-in user interface	Not built-in, requires Airflow UI	Built-in visual development interface
Handling Streaming Data [9]	Yes, event-driven, supports real-time	No, batch-oriented	No, batch-oriented	No, batch-oriented	Yes, batch and streaming	Yes, task-based workflows
Ease of Cloud Deployment [10]	Easy	Hard	Easy	Moderate	Moderate	Easy

[1] How easy or difficult it is to set up and configure the workflow orchestration tool.[2] The ability of the tool to handle increasing workloads and efficiently manage distributed execution.[3] The extent to which the tool supports various data sources, third-party integrations, and different workload types. [4] Availability of built-in monitoring, real-time tracking, and logging capabilities. [5] The level of community adoption, availability of documentation, and official support. [6] Whether the tool has built-in version control. [7] The level of technical expertise required to implement and maintain workflows. [8] The quality and accessibility of the tool's user interface for designing and managing workflows. [9] The tool's capability to handle both batch processing and real-time (streaming) data workflows.[10]How well the tool supports cloud-based deployment and integrates with cloud-native services.

4.3 Results and Analysis

This section discusses the outcome of the pipeline implementation using PREFECT and includes results on anomaly detection, the data provenance chain, alert framework, sensitivity analysis functionality, pipeline resource characterization

4.3.1 Data Pipeline Implementation using Prefect

As explained in Section 2.1.2, a workflow orchestration tool is needed to establish and maintain the pipeline. To determine the best option for the most appropriate tool for this work,a number of available options were reviewed and compared in Table 4.2 [3]. It was determined that PREFECT was the best option because of its characteristics across all the categories.

The pipeline was implemented using PREFECT and visualized as a dependency graph showing all the flows and the respective tasks they are comprised of. It also includes the progression/order of flows and provides a good understanding of their relative execution lengths which helps identify resource intensive flows. The developed flows include alert, sensitivity analysis, train model, scheduled read flow, and batch processing. The main pipeline flows include scheduled read flow and batch processing with the other listed flows attached as

augmentary flows to materialize the proposed solution.

1. Scheduled Read Flow: This encompasses the ingestion of data into the pipeline. It involves selecting a subset of the data within a desired time window of a number of hours to run the pipeline. It is made up of 2 main tasks: Line-by-line extraction of the data and accumulation into batches. The tasks included in the flow are illustrated in Figure 4.7 detailed as follows:

    - Line-by-line data extraction: Extracts selected data subset line by line to mimic real-time data sensor readings. A read interval of 1 second was chosen to facilitate rapid testing.
    - Data accumulation: Combines data into manageable chunks for processing. An accommodating batch resolution of 30 was chosen. This can be included in future studies as a malleable/tunable parameter dependent of tradeoff between processing efficiency and resource availability.

2. Batch Processing Flow: This encompasses the remaining pipeline components including Filtering, Cleaning, Aggregation, Integration, and Fault Detection. The details of these tasks are detailed in Table 4.1. This flow also includes data provenance and UQ computation tasks as well as other augmentary subflows including sensitivity analysis and the alert flow. The tasks included in the flow are illustrated in Figure 4.9 detailed as follows:

    - UQ computation: The UQ statistics includes IQR, mean, variance, standard deviation, and entropy are computed for all stages in each batch process.
    - Data Provenance: A global keyed dictionary is constantly updated with the UQ computed for each stage during every batch process. The data provenance tasks within the batch process flow are highlighted in yellow in Figure 4.9

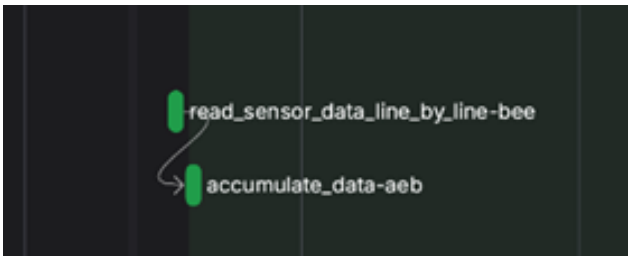


Figure 4.7: Dependency Graph for Scheduled Reading Flow

The subflows included in this flow are detailed in Section 4.1.

- Sensitivity Analysis: Perform Sobol analysis on desired parameter on given function.
- Alert Framework: Triggers alerts based on manual of dynamic thresholds. If set for dynamic thresholds, it invokes the task which computes the dynamic threshold based on historical UQ trends observed from data provenance chain. The alert flow within the batch process flow is highlighted in red in Figure 4.9. The "getDynamicThresholds" tasks is only used when using dynamic thresholds.

3. Model Training: Tasks include combining faulty and fault-free data, extracting the temperature features, training the RF model, calculating ML metrics, and saving the trained model to be utilized for fault detection in batch processing flow. The flow is illustrated in Figure 4.8.

4.3.2 Sensitivity Analysis

Local sensitivity analysis was performed on the window size parameter in the smoothing data function within the batch processing flow. The goal was to see which range of values for window size, the parameter of interest, influences the output the most. The test was conducted for 5 sample ranges spanning 4 units each to determine the 1st and total order

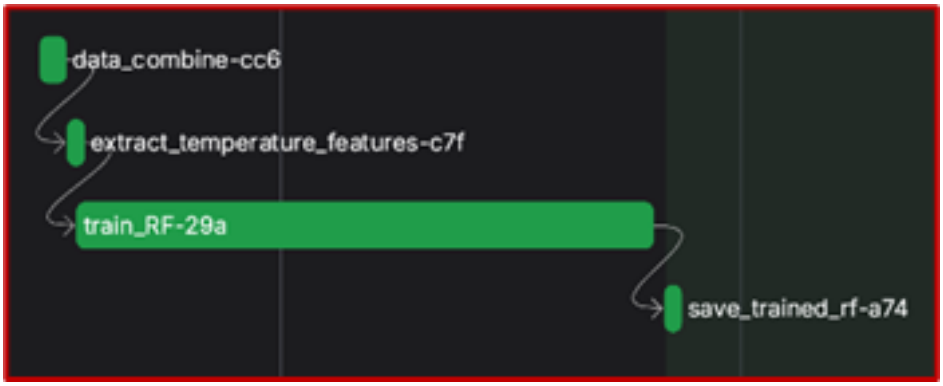


Figure 4.8: Dependency Graph for Model Training Workflow

Figure 4.9: Dependency Graph for Batch Processing Flow

Table 4.3: Sensitivity Analysis Results for Smoothing Data Function with Window Size as Parameter of Interest

Bound	First Order Sensitivity Index	Total Order Sensitivity Index
(1,4)	0.66800533	0.22401795
(5,8)	0.67667689	0.232412
(9,12)	0.67543253	0.23113216
(13,16)	0.67536851	0.23105555
(17,20)	0.67508652	0.23076825

sensitivity index. The first and total order sensitivity index is reported in Table 4.3. As a reminder, the first order index checks the influence of the value ranges independently while total order index check for influence relative to other ranges. The high the index, the more the influence. After ranking them based on both indexes, the most influential range of the window size parameter is (5,8).

4.3.3 Data Provenance and Alert Framework

Figure 4.10 shows the dependency graph for a batch processing workflow highlighting the consistent update of the data provenance chain at every stage (indicated in yellow) with the alert framework engaged at the end of the flow (indicated in red). In this flow, the thresholds were set dynamically, resulting in the activation of the ”getDynamicThresholds” task. The task would be absent in a use case that utilizes manual thresholds.

Figure 4.10: Data Provenance and Alert Framework in PREFECT Flow


```
"Batch 1": {
    "Filtering Stage Statistics": {
        "FCU_OAT": {
            "IQR": 0.07499999999999929,
            "Mean": 25.911,
            "Standard Deviation": 0.06331911516143936,
            "Variance": 0.00400931034482762,
            "Entropy": 2.635976399632171
        },
        "Label": {
            "IQR": 1.0,
            "Mean": 0.5333333333333333,
            "Standard Deviation": 0.5074162634049248,
            "Variance": 0.2574712643678161,
            "Entropy": 0.690923309313818
        }
    },
    "Cleaning Stage Statistics": {...
    },
    "Aggregation Stage Statistics": {...
    },
    "Integration Stage Statistics": {...
    }
},
"Batch 2": {
    "Filtering Stage Statistics": {...
    },
    "Cleaning Stage Statistics": {...
    },
    "Aggregation Stage Statistics": {...
    },
    "Integration Stage Statistics": {...
    }
},
"Batch 3": {
    "Filtering Stage Statistics": {...
    },
    "Cleaning Stage Statistics": {...
    },
    "Aggregation Stage Statistics": {...
```


Figure 4.11: Structure of Data Provenance Chain

An example recorded provenance chain is provided in Figure 4.11 a structure that accounts for pipeline stages within every batch flow that is run. The chain is then analyzed by the alert framework to create alerts. As referenced in the alert notification sample provided in Figure 4.12, the alerts provide the user with information about which pipeline stage had thresholds exceeded, the affected feature, the UQ statistic which exceeded an upper or lower limit, and by how much. With manual threshold limits, the user decides the sensitivity of alerts manually and with dynamic thresholds, the user tunes the sensitivity by using a specific buffer value. The buffer value determines how much of the historical UQ trends affects subsequent thresholds

```
[
    {
        "Pipeline Stage":"Filtering Stage Statistics",
        "Feature":"RM_TEMP",
        "UQ Measure":"Entropy",
        "Value":2.8928894493,
        "Lower Threshold":2.6486285755,
        "Upper Threshold":2.8881000204,
        "Threshold Type Exceeded":"Upper Threshold",
        "Exceeded by":0.0047894289
    },
    {
        "Pipeline Stage":"Filtering Stage Statistics",
        "Feature":"FCU_DAT",
        "UQ Measure":"Mean",
        "Value":119.91,
        "Lower Threshold":119.14772,
        "Upper Threshold":119.8950533333,
        "Threshold Type Exceeded":"Upper Threshold",
        "Exceeded by":0.0149466667
    },
    {
        "Pipeline Stage":"Filtering Stage Statistics",
        "Feature":"FCU_RAT",
        "UQ Measure":"Entropy",
        "Value":2.8928894493,
        "Lower Threshold":2.6486285755,
        "Upper Threshold":2.8881000204,
        "Threshold Type Exceeded":"Upper Threshold",
        "Exceeded by":0.0047894289
    },
    {
```


Figure 4.12: Sample of Alert Notifications for Dynamic Thresholds

The dynamic threshold setting was tested using buffer values: 0.02, 02, 1, 2 and the plots showing the number of alerts for each batch shown in Figure 4.14. Owing to the fact that the provenance chain is not established initially, the first batch has no reference to any historical trend and thus produces no alerts.4.14. Once the first batch is processed and the provenance chain updated, thresholds for the subsequent batch are set conditioned by the buffer value and the range of UQ values derived from the first batch. As the buffer value is increased

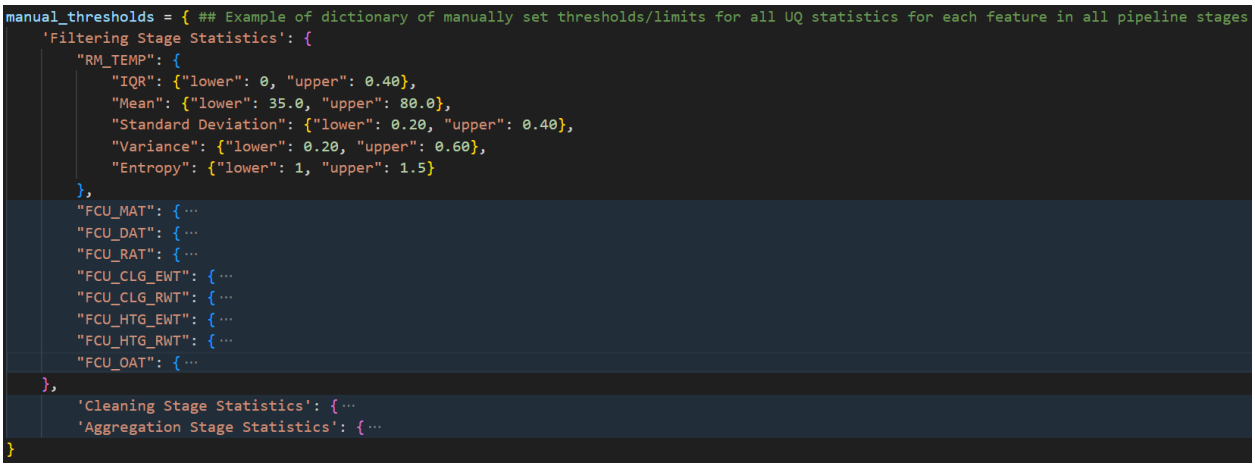


Figure 4.13: Example of User Specified Manual Threshold Specifications for Alert Framework

from 0.02 to 2, it is evident that the alert sensitivity increasing, indicated by the rise in the number of alerts. Constant batchwise revision of threshold values continue and thus the plots are not linear.

The manual threshold setting was also tested (Example of manual thresholds in Figure **??**) and results provided in the solid plot in Figure 4.15. Compared with the plots of the two extremes from the dynamic threshold setting with buffer values 0.02 and 2 (shown in dashed lines in Figure 4.15), the manual plot shows an approximately constant number of alerts due to the fixed thresholds that are not constantly revised with every batch.

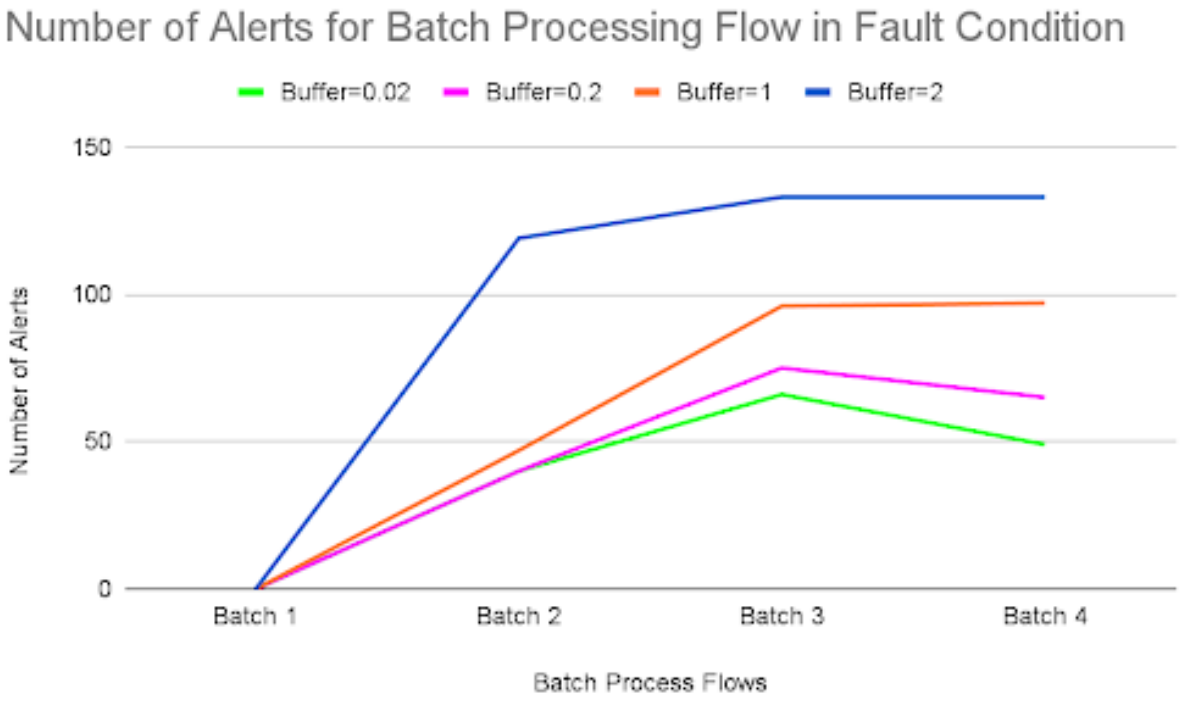


Figure 4.14: Number of Alerts for Dynamic Thresholds in Fault Scenario

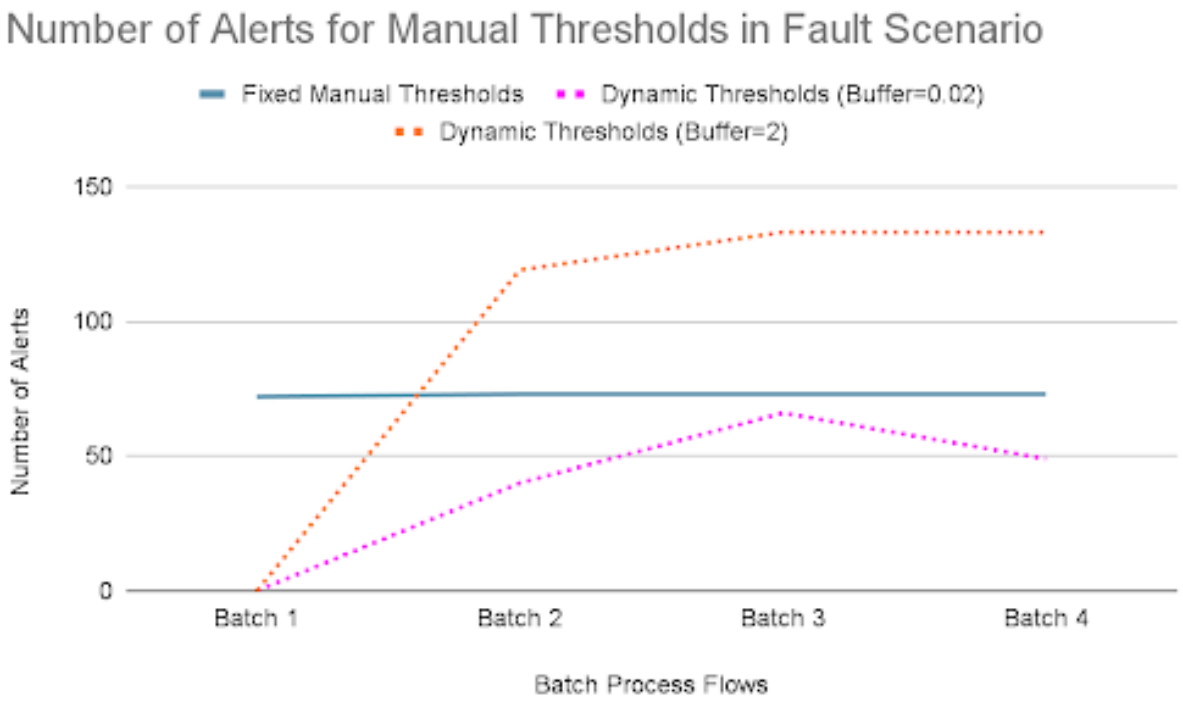


Figure 4.15: Number of Alerts for Manual vs Dynamic Thresholds in Fault Scenario

4.3.4 Anomaly Detection Experimentation

The pipeline was tested to determine distributional shifts in UQ statistics for anomaly detection in events of attacks and results published in [3]. Two experiments are conducted for evaluation. The first scenario involves an attacker targeting the computation in a specific pipeline stage The second scenario involves attack on the data ingestion where the attacker passes corrupt data through the pipeline. For each scenario, we record and analyze shifts in batches of 30 mins recorded data. The experiment was conducted for a 3 hour observation window, resulting in 6 batches.

We used the HVAC data from the FCU setup discussed Section 4.2 and also repeated experiments with UAV data under normal operations and with cyberattacks [44] with results

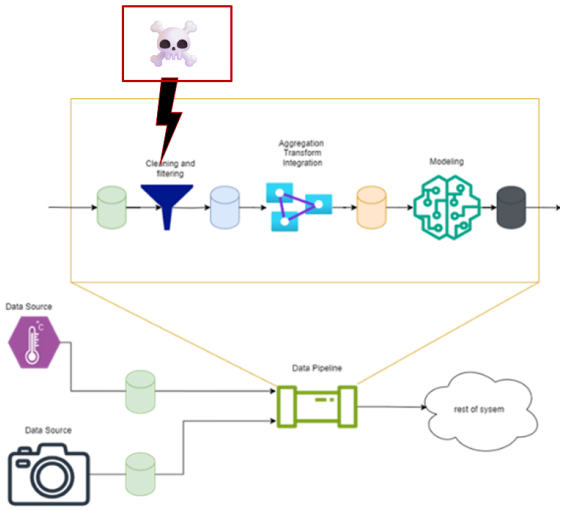


Figure 4.16: Threat Model for Pipeline Stage Corruption Showing Attacker Targeting Cleaning Stage

detailed in Github.

Case 1: Single-Stage Computational Corruption

In this experiment, we aim to detect computational anomalies occurring in a specific stage of the pipeline The threat model, illustrated in Figure **??** portrays an attacker targeting the data cleaning stage where missing data is handled by forward filling, outliers are detected and removed, data is smoothened, and normalized. To simulate pipeline stage corruption, controlled corruption in introduced by changing the method of normalization. for a duration during processing, specifically in the third Batch. The remaining batches remain untouched by this attack. The key research question here is "Can the proposed architecture detect reliably detect a localized, stage-specific deviation with other stages functioning normally?"

To determine if the corrupted stage will show abnormal shifts in UQ statistics during the duration of the attack, we observe batch-wise statistical graphs for the observation widow.

Two features, room temperature (RM_TEMP) and Fan Coil Unit Discharge Air Temperature (FCU_DAT) were selected and plots showing the trends in the UQ statistical indicators at the target cleaning stage were developed as shown in Figure 4.17.

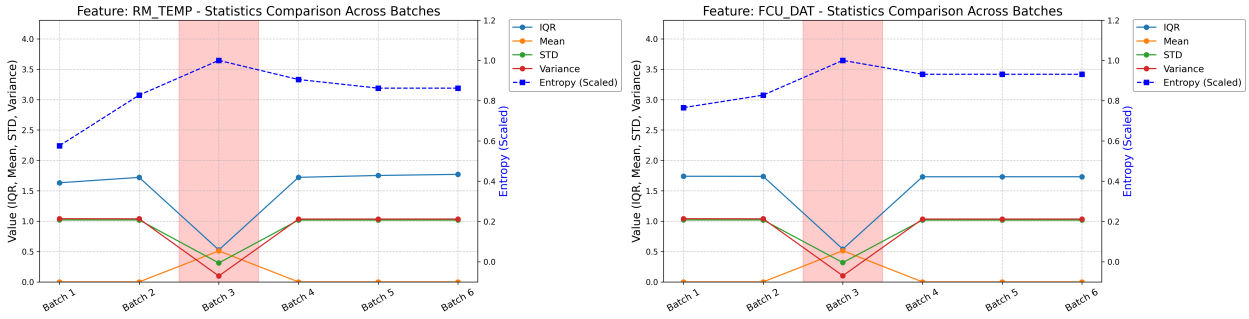


Figure 4.17: Batch-wise Statistical Feature Comparison illustrating changes in IQR, Mean, and STD, Variance, and scaled Entropy Across Different Batches for RM_TEMP and FCU_DAT at the cleaning stage. A significant deviation is observed in Batch 3 for majority of UQ statistics.

Batch 3, highlighted in light red, is where corruption has been introduced by changing the normalization method. If majority of UQ statistics show significant shifts at this point, there is an indication of anomalous behavior. A clear shift in statistics is observed in Batch 3 with a sharp recovery in Batch 4, reflecting a discontinuity in the data caused by the manipulation. There are instances in other features (available on Github) have instances where only one UQ statistic differs from this trend where sharp shifts are shown at multiple batch period besides the target batch for only that statistic. This could be an indication of the influence of ripple effects of the shifts after the attack in subsequent batches. Nevertheless, all features have a majority shift at the target location, indicating an anomaly that should prompt a response. These results confirm that our framework can detect when computations in a pipeline stage have been corrupted, making it especially useful for prompt fault response in real-world data pipelines where the location and timing of the anomaly is not known in advance.

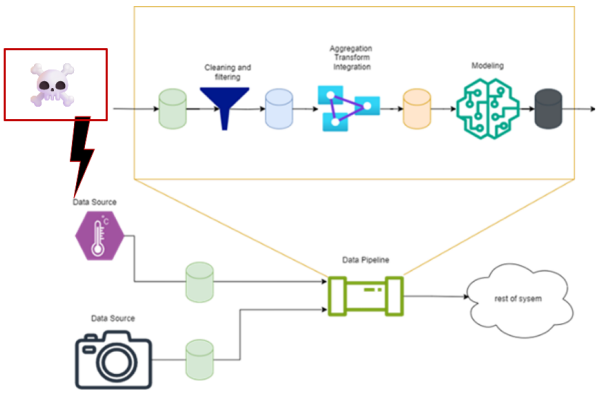


Figure 4.18: Threat Model for Inline Data Corruption Showing Attacker Targeting Ingestion

Case 2: Inline Data Corruption

In this case, we explore a different type of threat model where the attacker targets the data source at the point of ingestion, essentially corrupting the raw input before it even enters the pipeline. The attacker's goal is to manipulate the incoming data stream, such that it triggers false alerts or incorrect decisions downstream despite the pipeline functioning as expected. As shown in Figure 4.18, the adversary intervenes at the very first stage, before the data enters preprocessing or modeling. This can cause a chain reaction where even valid pipeline logic produces corrupted model outputs. We model this by feeding a train of fault-free data followed by faulty data. The statistics are averaged across all stages to create a unified impression of data changes across the pipeline as a single unit.

The key research question becomes "Can our pipeline architecture detect data corruption as an influence throughout the pipeline as a whole?"

Figure 4.19 illustrates the results of an attack introduced at the data ingestion point, simulating corruption of the pipeline as a whole.

Each plot shows statistical trends for RM_TEMP. The fault type introduced here is a sensor

bias in air temperature measurements making the RM_TEMP feature a very likely target. Other features that are not directly affected by this fault type would not follow the same observed trends. On the left, fault-free data is fed for 1 hour followed by 2 hrs of faulty data (meaning corrupt data enters the pipeline in Batch 3). On the right, fault-free data is fed for 2 hrs hour followed by 1 hr of faulty data. (meaning corrupt data enters the pipeline in Batch 5). The UQ statistics across all stages are averaged for each batch process to show trends in the whole pipeline.

The highlighted regions represent the corrupted batches, where the attacker introduced manipulation.

What is observed are spikes in the scaled mean (orange dotted line), while other statistics like variance and entropy also deviate from their typical patterns in those intervals. These visual deviations highlight how even early-stage corruptions can propagate through the pipeline and distort the overall statistical signature of the data.

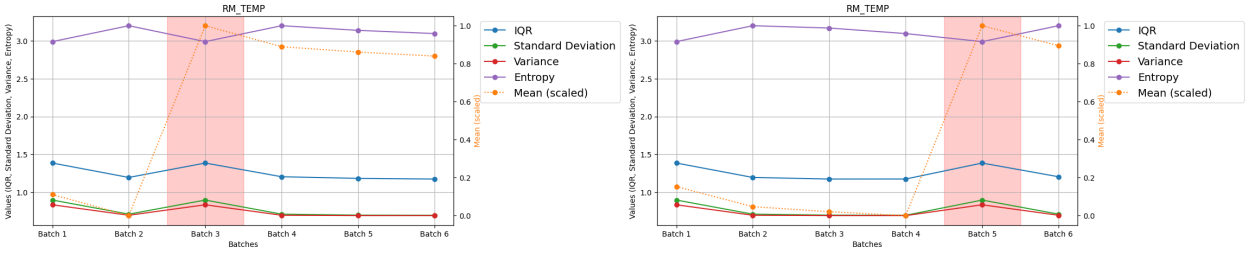


Figure 4.19: Batch-wise Statistical Feature Comparison illustrating changes in IQR, Mean, and STD, Variance, and scaled Entropy Across Different Batches for RM_TEMP with Fault-free Data Followed by Faulty Data for 1hour-2hours and 2hours-1hour Respectively. A significant deviation is observed in Batch 3 and Batch 5.

4.3.5 Resource Availability Analysis

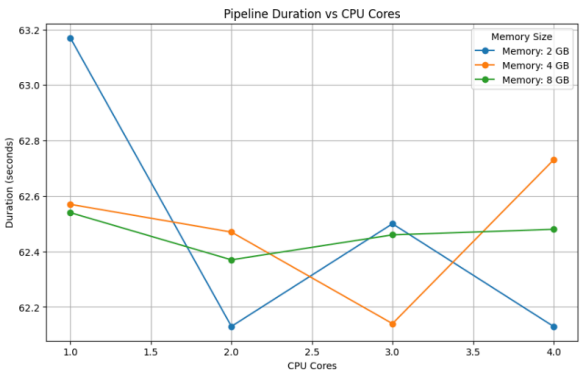
For the purpose of identifying potential performance bottlenecks to to optimize the allocation of resources, the resource consumption of the pipeline is characterized. A linear

regression model is used to establish a quantified relationship between these three metrics of interest. The equation representing this is expressed in Equation 4.8, indicating an inverse relationship between processing time (in seconds) and the other metrics; CPU and memory utilization. The -0.0049 coefficient implies that processing time reduces by approximately 0.0049 seconds with every 1% increase in CPU utilization. More notably, processing time reduces significantly by about 1.4578 seconds with every 1% increase in memory utilization. Thus, processing time is more strongly influenced by memory utilization.

$$\text{Duration (seconds)} = -0.0049 \times \text{CPU (\%)} - 1.4578 \times \text{Memory (\%)} + 93.6 \tag{4.8}$$

Further resource availability analysis is accomplished by investigating the changes in processing time with specific combinations of CPU cores and memory sizes. In simple terms, it involves running the pipeline with the same load multiple times observe execution time versus the number of CPU cores and memory usage. A Linux system equipped with 4 CPU cores and 7.7GB available memory was used in experiments where the pipeline was executed with configurations involving varying combinations of CPU cores (1 - 4) and memory sizes (2GB, 4GB, and 8GB) and the execution time recorded and illustrated in Figure 4.20. From the graph of number of CPU cores,versus pipeline duration the resultant change in duration throughout the progression from 1 core to an increased 4 cores is an overall decrease in pipeline duration.

Similarly, the graph of memory size versus pipeline duration shows a fair decrease in processing time as memory size increases with the overall resultant trend considering all cpu cores plots.



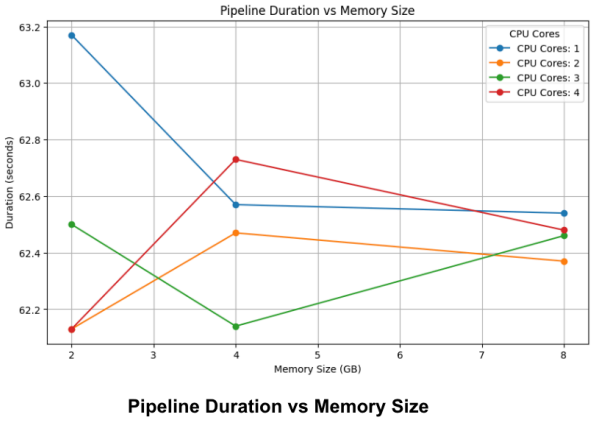


Figure 4.20: Processing Time vs Available Resources

Chapter 5

Discussion

The goal of this thesis is to accessorize the generic data pipeline framework with security-specific features to maintain integrity through attentive and efficient monitoring. This is done by implementing a combination of augmentation frameworks consisting of UQ estimations, sensitivity analysis, data provenance, and anomaly alerts. As detailed in the methodology, the procedure for realizing this entails (1) Sampling inputs for a pre-determined window, (2) Computing entropic UQ, statistical moments, and IQR for the output distribution, (3) Optionally performing sensitivity analysis of parameters, (4) Attaching measured UQ statistics to the incrementally updated provenance chain, (5) Observing distributional shifts in captured UQ statistics, and (6) Raising alerts during undesired shifts from historical trends or user-defined thresholds. Proof of concept is illustrated with experiments to address two cases of security analysis; single-stage computational corruption detection and inline data corruption detection. The added monitoring frameworks serve their purposed objectives (Table **??**) to identify shifts within the window of attack or corruption for targeted stages in the pipeline.

5.0.1 Future Work

The concepts described in this implementation can be extended to hardware proposed as "Uncertainty Quantification in RTL Information Flow Analysis for Security-Aware Hardware

Data Processing". RTL descriptions of hardware designs are provided manually by expert engineers or automatically by rapid generators. The enormity of the code implementations result in exploitable vulnerabilities causing the design to fail to meet security standards or specifications. Testing frameworks have since been deduced to hold hardware systems to a security standard. Some common concerns with the state of the art verification approaches are the computational expense, the complexity and reusability of manual testing, and the extent of coverage for large systems such as SoCs. A framework capable of isolating the most vulnerable components in the design for prioritized testing is required.

To resolve this, future work can facilitate the integration of uncertainty quantification mechanisms to assist designers during the design phase to prioritize certain parts of the hardware design which carry very sensitive information. By analyzing the information flow through major paths with all system components, the proposed framework observes input and output data shifts, detecting potential malicious information flows, and providing a measure of confidence (UQ) to determine the extent of vulnerability to certain categories of attack. This is visually summarized in Figure 5.1. The RTL description is first converted into an abstract tree graph [45] and the data flow paths are isolated as stand-alone pipelines from the graph. For the chosen testbench attack data, the UQ is framework is applied to estimate the uncertainty and distributional shifts in statistics to determine vulnerability.

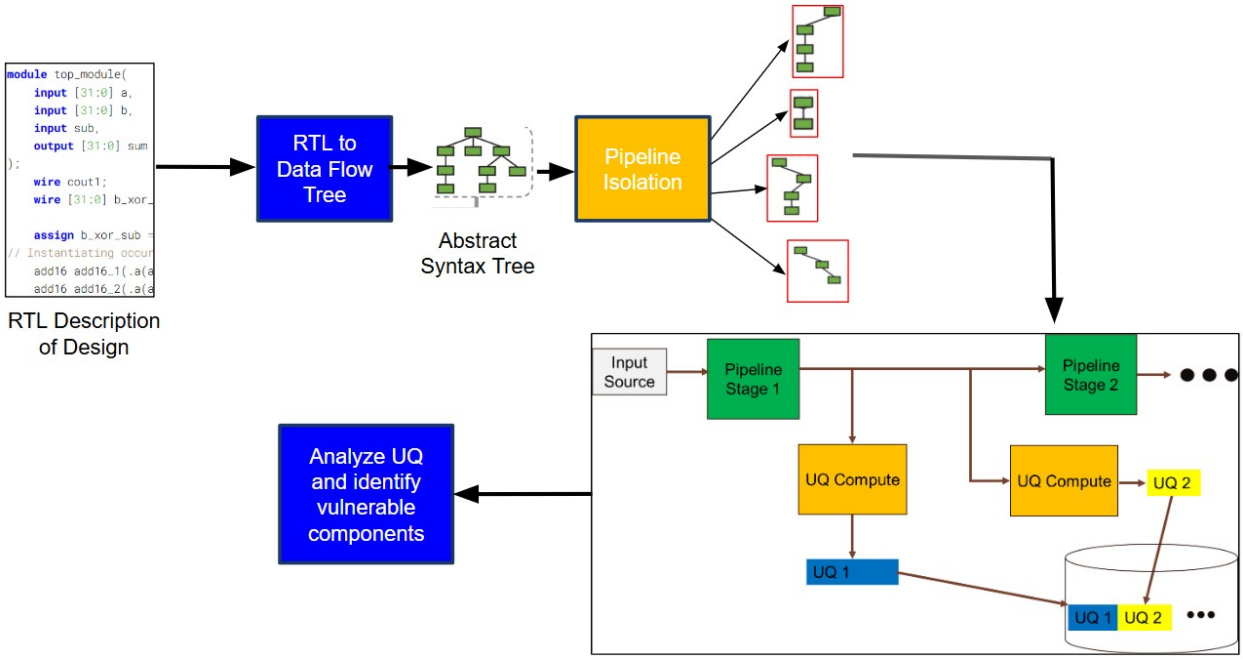


Figure 5.1: Uncertainty Quantification Framework for RTL Information Flow Analysis

Chapter 6

Conclusion

The central reflections on the impact of this work include:

- **Fault-sensitive Execution:** The recurrent calculation of uncertainty coupled with an intuitive alert framework create a pipeline environment that detects and creates notifications that prompts operators to attend to possible faults or attacks targeting specific components in any cyber-physical system.
- **Full Fault Traceability:** Data provenance provides accurate recordings of statistical trends over multiple batch computations to allow for ease of audits when tracking the source of fault or attack instances.
- **Tuneable Alert Frequency**: In order to encourage alert fatigue alleviation, this work introduces an alert triage mechanism where operators can specify which specific UQ statistics should be referenced for triggering alerts. Additionally, the manual threshold option in the alert flow enables automatic triage based on restrictions placed by operators. Finally, the introduction of the buffer parameter in the dynamic threshold option gives assigned security analysts the ability to control the buffer parameter in the alert pipeline flow which determines how much dependence is placed on historical trends to set alert thresholds. The direct proportionality relationship between this value and the number of alerts also determines the alert sensitivity .
- **Parameter Analysis for Strategic Sampling Readjustment:** Sensitivity anal-

ysis provides users with information on the extent of influence of input parameters on function outputs. This helps inform the user on how to effectively control sampling hyperparamters such as rate of sampling or window size to improve computation results.

- **Pipeline Resource Characterization:** The quantification of the relationship between CPU, memory utilization, and processing time provides ample information on the influence of certain resource combinations on pipeline execution efficiency.

- **Data Visualization** Line-by-line analysis of the data provenance chain can be exhausting. A more practical approach would be to provide analysts with a collective visual of trends in all UQ statistics to make examining of the data easier. This work provides batchwise graphs with scaled values for joint analysis all statistics at the same time.

References

[1] Morning Consult. *Global Security Operations Center study results.* March 2023.

[2] KPMG. *KPMG Cybersecurity Survey.* 2024.

[3] Alberta Dadeboe, Farzaneh Mansourifard, and Shridatt Sugrim. Uncertainty quantification and data provenance for data pipeline security analysis. In *Proceedings of the 7th Workshop on Design Automation for CPS and IoT*, page 1–6, Irvine CA USA, May 2025. ACM.

[4] J. Densmore. *Data Pipelines Pocket Reference.* O'Reilly Media, 2021.

[5] Ralph Kimball and Margy Ross. *The data warehouse toolkit: The definitive guide to dimensional modeling.* John Wiley & Sons, 2013.

[6] Sebastian Pulkka. The modernization process of a data pipeline. 2023.

[7] Aiswarya Raj Munappy, Jan Bosch, and Helena Homström Olsson. Data pipeline management in practice: Challenges and opportunities. In Maurizio Morisio, Marco Torchiano, and Andreas Jedlitschka, editors, *Product-Focused Software Process Improvement*, pages 168–184, Cham, 2020. Springer International Publishing.

[8] Tatiana Von Landesberger, Dieter W. Fellner, and Roy A. Ruddle. Visualization system requirements for data processing pipeline design and optimization. *IEEE Transactions on Visualization and Computer Graphics*, 23(8):2028–2041, aug 2017.

[9] Harald Foidl, Valentina Golendukhina, Rudolf Ramler, and Michael Felderer. Data pipeline quality: Influencing factors, root causes of data-related issues, and processing

problem areas for developers. *Journal of Systems and Software*, 207:111855, January 2024.

[10] Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001.

[11] Panagiotis D. Diamantoulakis, Vasileios M. Kapinas, and George K. Karagiannidis. Big data analytics for dynamic energy management in smart grids. *Big Data Research*, 2(3):94–101, 2015. Big Data, Analytics, and High-Performance Computing.

[12] Mohamed Mokbel. A systematic review of big data integration challenges and solutions for heterogeneous data sources. *ACADEMIC JOURNAL ON BUSINESS ADMINISTRATION, INNOVATION SUSTAINABILITY*, 4(4):1–18, October 2024.

[13] Pavan Kumar Narayanan. *Getting Started with Workflow Management and Orchestration*, pages 361–382. Apress, Berkeley, CA, 2024.

[14] CodeSeoul. Sh: Let's build a data pipeline with prefect!, September 2023.

[15] Janifer Nahar, Md Atiqur Rahaman, Md Alauddin, and Farhana Zaman Rozony. Big data in credit risk management: A systematic review of transformative practices and future directions. *International Journal of Management Information Systems and Data Science*, 1(04):68–79, Sep. 2024.

[16] Peipei Sui and Xianxian Li. A privacy-preserving approach for multimodal transaction data integrated analysis. *Neurocomputing*, 253:56–64, 2017. Learning Multimodal Data.

[17] Rashid Hussain Khokhar, Farkhund Iqbal, B. Fung, and J. Bentahar. Enabling secure trustworthiness assessment and privacy protection in integrating data for trading person-specific information. *IEEE Transactions on Engineering Management*, 68:149–169, 2021.

[18] Jiaxin Zhang. Modern monte carlo methods for efficient uncertainty quantification and propagation: A survey. (arXiv:2011.00680), November 2020. arXiv:2011.00680.

[19] *Guide to the expression of uncertainty in measurement — Part 1: Introduction*$_2$023. *jan*2023.

[20]

[21] Juan Zhang, Junping Yin, and Ruili Wang. Basic framework and main methods of uncertainty quantification. *Mathematical Problems in Engineering*, 2020:1–18, August 2020.

[22] Hiromitsu Kumamoto and Ernest J. Henley. *Uncertainty Quantification*, page 535–572. IEEE, 1996.

[23] National Research Council. *Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification*. The National Academies Press, Washington, DC, 2012.

[24] Mark Russinovich. Confidential computing: Elevating cloud security and privacy: Working toward a more secure and innovative future. *Queue*, 21(4):44–48, September 2023.

[25] Peterson Yuhala. *Enhancing Security and Performance in Trusted Execution Environments*. PhD thesis, Università della Svizzera italiana, Lugano, 2024.

[26] Haotian Gao, Cong Yue, Tien Tuan Anh Dinh, Zhiyong Huang, and Beng Chin Ooi. Enabling secure and efficient data analytics pipeline evolution with trusted execution environment. *Proceedings of the VLDB Endowment*, 16(10):2485–2498, June 2023.

[27] Fan Mo, Zahra Tarkhani, and Hamed Haddadi. Machine learning with confidential computing: A systematization of knowledge. *ACM Computing Surveys*, 2022.

[28] Antongiacomo Polimeno, Paolo Mignone, Chiara Braghin, Marco Anisetti, Michelangelo Ceci, Donato Malerba, and Claudio A. Ardagna. Balancing protection and quality in big data analytics pipelines. *Big Data*, 13(2):127–143, 2025. PMID: 38603580.

[29] Julia Jentsch, Ali Burak Ünal, Şeyma Selcan Mağara, and Mete Akgün. Privacy preserving data imputation via multi-party computation for medical applications. In *2024 IEEE International Conference on E-health Networking, Application Services (HealthCom)*, pages 1–6, 2024.

[30] Yustus Eko Oktian, Shinwook Heo, and Howon Kim. Signora: a blockchain-based framework for dataflow integrity provisioning in an untrusted data pipeline. *IEEE Access*, 10:89714–89731, January 2022.

[31] Akif Quddus Khan, Nikolay Nikolov, Mihhail Matskin, Radu Prodan, Dumitru Roman, Bekir Şahin, Christoph Bussler, and Ahmet Soylu. Smart data placement using storage-as-a-service model for big data pipelines. *Sensors (Basel, Switzerland)*, 23, 2023.

[32] Bruno Sousa, Miguel Arieiro, Vasco Pereira, João Correia, Nuno Lourenço, and Tiago Cruz. ELEGANT: Security of Critical Infrastructures With Digital Twins. *IEEE Access*, 9:107574–107588, 2021.

[33] Prakash Tekchandani, Abhishek Bisht, Ashok Kumar Das, Neeraj Kumar, Marimuthu Karuppiah, Pandi Vijayakumar, and Youngho Park. Blockchain-enabled secure collaborative model learning using differential privacy for iot-based big data analytics. *IEEE Transactions on Big Data*, 11:141–156, 2025.

[34]

[35] Sanchit Sethi. Applying reservoir computing for driver behavior analysis and traffic flow prediction in intelligent transportation systems. Master's thesis, Virginia Tech, 2024.

[36] Chunxiao Lin, Muhammad Farhan Azmine, and Yang Yi. Invited paper: Accelerating next-g wireless communications with fpga-based ai accelerators. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–8, 2023.

[37] Nan Chen, Stephen Wiggins, and Marios Andreou. Taming uncertainty in a complex world: The rise of uncertainty quantification - a tutorial for beginners. *Notices of the American Mathematical Society*, 72(03):1, March 2025.

[38] Nasir Bilal. Implementation of sobol's method of global sensitivity analysis to a compressor simulation model. *International Compressor Engineering Conference*, January 2014.

[39] I.M Sobol . Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55(1):271–280, 2001. The Second IMACS Seminar on Monte Carlo Methods.

[40] Andrea Saltelli and I.M. Sobol'. Sensitivity analysis for nonlinear mathematical models: Numerical experience. *Matematicheskoe Modelirovanie*, 7, 01 1995.

[41] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[42] Jessica Granderson, Guanjing Lin, Yimin Chen, Armando Casillas, Jin Wen, Zhelun Chen, Piljae Im, Sen Huang, and Jiazhen Ling. A labeled dataset for building hvac systems operating in faulted and fault-free states. *Scientific data*, 10(1):342, 2023.

[43] Jessica Granderson, Guanjing Lin, Yimin Chen, Armando Casillas, Piljae Im, Sungkyun Jung, Kyle Benne, Jiazhen Ling, Ravi Gorthala, Jin Wen, et al. Lbnl fault detection and diagnostics datasets. Technical report, DOE Open Energy Data Initiative (OEDI); Lawrence Berkeley National …, 2022.

[44] Samuel Chase Hassler, Umair Ahmad Mughal, and Muhammad Ismail. Cyber-physical intrusion detection system for unmanned aerial vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 25(6):6106–6117, 2024.

[45] Rozhin Yasaei, Shih-Yuan Yu, Emad Kasaeyan Naeini, and Mohammad Abdullah Al Faruque. Gnn4ip: Graph neural network for hardware intellectual property piracy detection, 2021.