National College of Ireland

Integrating Autoencoders and GPU Acceleration for Proactive Anomaly Detection in ETL Workflows

MSc Research Project
MSc in Data Analytics
(MSCDAD_C)

Kranthi Kumar Yedla

Student ID: x23288892

School of Computing

National College of Ireland

Supervisor: Dr. David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Kranthi Kumar Yedla

Student ID: x23288892 **Year:** 2025

Programme: MSc in Data Analytics

Module: Research Practicum Part 2

Lecturer: Dr. David Hamill

Submission Due Date: 1st Sept 2025

Project Title: Integrating Autoencoders and GPU Acceleration for Proactive Anomaly Detection in ETL workflows

Word Count: 13528 **Page Count:** 28

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Kranthi Kumar Yedla

Date: 30th Aug 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Integrating Autoencoders and GPU Acceleration for Proactive Anomaly Detection in ETL Workflows

Abstract:

This thesis addresses the problem of improving data quality validation within ETL pipelines where the legacy manual methods of data validation tend to be inefficient in identifying minor anomalies and leads to defect leakage into production. To address this we developed a GPU-accelerated anomaly detection architecture, combining autoencoder neural networks, as part of a Dagster-managed ETL pipeline. The method was experimented on three datasets, namely, banking, insurance and sales, with injected anomalies and standard validation checks involving nulls, duplicates and missing records. In a series of experiments, it was shown that the system was highly detective with all injected anomalies being detected with about 1% of the anomalies being detected in banking data and about 0.5 %in insurance data, and in the sales data only low-level rule problems were detected. GPU acceleration was demonstrated to have a significant positive effect on runtime performance, as validation took 12 to 15 seconds to complete versus 20 to 25 seconds on CPU. Altogether, the results indicate that the use of autoencoders with GPUs is capable of improving ETL QA testing because it offers quick, precise, and automated anomaly detection that supplements the conventional data quality control and might lead to leakage of defects into the production.

Introduction:

1.1 Background and Motivation:

ETL pipelines in data-driven organizations are important to reconcile data obtained through various sources in a form which is appropriate to analyze and make decisions. Integrity and quality of information passing through ETL processes, the quality of information that moves through these processes is extremely important. Nevertheless, most companies continue to use manual, ad-hoc Quality Assurance (QA) procedures to check ETL outputs. Manual ETL testing can detect some data errors, but it is time-consuming and does not necessarily identify all kinds of errors. Such dependence on human checks not only lengthens the pipeline development process but is also liable to defect leaks, aka undetected data errors that sneak out into the production environment. These defects may be passed over onto the analytical reports, causing inaccurate insights and decisions.

Problems with the manual QA usually become unacceptable as data volume increases and as data becomes more complex. This indicates the necessity to introduce an automated anomaly detection method to an ETL pipeline. Anomaly detection involves the use of machine learning to automatically detect anomalous patterns or inconsistencies in data, as it differs very much from what was thought to be normal behaviour. The proposed autoencoder-based method can be used as a robust unsupervised anomaly detector. Anomalies are detected in the records with the highest rates of reconstruction error in autoencoders. This methodology will reveal hidden information discrepancies that would not be detected in rules-based or human review.

Integration of such models into ETL processes allows real-time checks on data quality rather than a one-time snapshot check.

Since ETL processes deal with large amounts of data (it may contain millions of records), anomaly detection cannot be a bottleneck. The fact that GPU acceleration is, therefore, extremely beneficial in this work. GPUs efficiently perform parallel operations and are therefore useful in accelerating deep learning models such as autoencoders, on large amounts of data. Using GPUs means anomaly detection becomes scalable, and therefore performed on large datasets, at near-real-time. In short, autoencoder-based anomaly detection and GPU acceleration will create a scalable, proactive, end-to-end solution to data quality assurance, overcoming the weaknesses of manual ETL QA.

1.2 Problem Statement:

Regardless of the strategic importance of defect-free data in the context of analytics, existing ETL pipeline QA processes are ineffective in big data ecosystems. Manual testing and rudimentary rule checks are time-consuming, error-prone and not feasible when involving large datasets, enabling data defects to reach the production environment. The challenge that will be considered in this thesis is to come up with an automated, GPU-accelerated anomaly detection application in ETL processes that can identify anomalies reliably in terms of scale without affecting the target system. The study issue requires an approach that will provide high levels of anomaly detection with minimum latency in the system.

1.3 Significance of this Study:

This study will be significant from both a practical and academic point of view. Practically, this kind of system would radically increase the organisational effectiveness in preserving data quality. Thereby identifying anomalies at their earliest possible, the system can save decision-makers the trouble of making decisions based on bad information, as well as downstream costs driven by faulty data. It also minimises the manual effort on the part of data engineering and QA teams and instead enables them to concentrate on root-cause analysis rather than performing routine checks. With data volumes such as these that are continually increasing, this degree of automation is paramount in maintaining trust in enterprise analytics.

Academically, the research sits at the crossroads of data engineering, machine learning and high-performance computing. It uses deep learning (in the form of autoencoders) on a field that traditionally was ruled by static rule-based verification. Along with this, the paper looks at how GPU acceleration can be used to support data pipeline workloads, showing how the hardware can deliver real-time processing capabilities in anomaly detection. This solution matches the rising culture of data observability, where integrated machine learning models are used to actively check data integrity on an ongoing basis. The results may be used to develop new ETL systems with on-board AI-enhanced data quality inspection, potentially forming the next stage of data analytics.

1.4 Purpose of this Study:

This research aims to plan, build and test a new framework to accelerate anomaly detection in ETL pipelines using GPUs to increase the data quality insurance process. The paper aims to demonstrate that an autoencoder-based method can be an effective early warning system for data anomalies and that such a solution can be applied efficiently at scale with the use of GPU hardware. The ultimate research goal consists of using those data integrity protection strategies to eventually show a feasible process of proactive data integrity protection across the data source-to-target flow that can contribute to more reliable business insight and analytics-based decisions.

Research Question:

How can GPU-accelerated AI models be leveraged for proactive anomaly detection in ETL data pipelines to improve data quality validation and reduce deployment-time errors?

Sub-Questions (SQs):

- SQ1: How effective are autoencoder-based models in detecting anomalies such as missing, duplicate, or mismatched records during ETL validation?
- SQ2: How does GPU acceleration impact the scalability and performance of anomaly detection on large datasets?
- SQ3: How can a hybrid approach that combines traditional rule-based checks with AI-driven anomaly detection improve reliability compared to standalone methods?

1.5 Aims and Objectives:

This study will establish a scalable and automatic anomaly detection system on ETL processes based on deep learning and GPU technology. To fulfil this goal, the following objectives are determined in the study:

1. Review Current Practices: Find out what the current ETL data quality assurance practices are and why they are limited in a large-scale environment.

2. Design & Build Anomaly Detection Model: Design and build an unsupervised anomaly detection model (build an autoencoder to learn normal patterns, and detect anomalous records)

3. Accelerate Anomaly Detection with GPU: Train and deploy the model on the GPU to make anomaly detection workable on large datasets, with minimal latencies.

4. Compare Performance: Measure accuracy and false alarm rates of the model compared with benchmark performance and check its performance improvement using GPUs.

5. Comparisons to Manual QA: Compare the automated method with the manual QA procedures to determine the impact in the detection coverage and speed.

1.6 Limitations:

This paper has a few limitations. To begin with, the presence of a narrow range of structured data (banking, insurance, and sales domains) that can limit the transferability of the results to other types of data. Second, the autoencoder-based anomaly detection model is used; other methods, like GANs, were not tested. Third, the implementation is limited to use with a mid-range graphical processing unit, and it is written in CUDA therefore presenting hardware and driver dependencies that might not be easily portable across computing platforms. Lastly, the results on the scalability will probably not be similar on the enterprise-level hardware or on the distribution systems because the GPU acceleration decreases the processing times.

1.7 Structure of Thesis:

Following the Introduction, the remaining thesis follows the structure detailed below:

- Chapter 2 - Literature Review: Critically examines the previous literature on ETL testing and data quality management, anomaly detection algorithms (with special emphasis on autoencoders) and the application of GPU acceleration to processing large volumes of data.
- Chapter 3 that follows - Methodology: The datasets, the framework, which was an autoencoder by using a GPU and the means by which rule-based checks on data were integrated, the experiment procedure and the evaluation criteria are described.
- Chapter 4 - Implementation: Discusses the application of the research methodology as follows: development of the system and building of the ETL pipeline, model development and integration of the GPU, and validation workflow automation.
- Chapter 5- Results and Discussion: It introduces results of every experiment, the performance of anomaly detection, comparison with each dataset, processing benchmark, and conclusions regarding the effectiveness of the proposed framework.
- Chapter 6 - Conclusion and Future Work: Consolidates the major findings, discusses the study limitations, and outlines the ways to conduct future research and how to expand the framework.

2. Literature Review

2.1 AI-Based Anomaly Detection in ETL Pipelines

The feasibility of integrating AI into the ETL to identify anomalies early enough prior to their having an opportunity to potentially cause an impact on production has been explored in different works. Recent works focus on deep learning approaches to anomaly detection, instead of classical models like Isolation Forest, such as autoencoders, that show promising results, especially in the domain of unsupervised anomaly detection. Maddali (2024) automated the steps of ETL data quality assurance through the application of deep autoencoders and confirmed that they can detect structural anomalies in data transformation phases. Autoencoders fit well into the task of reconstructing normal patterns and identifying exceptions on high-dimensional data, where labelled data is not available, and are therefore useful in dynamic ETL settings. Some have applied supervised learning training models, such as Alak et al. (2023 using k-NN and XGBoost models trained on anomaly detection of ETL job runtime performance, which produced ~99 per cent accuracy in anomaly detection of unusual pipeline

run performance time. Similarly, Ande and Kumar (2025) implemented AI-based anomaly detection in cloud ETL workflows and noted an almost 40 per cent reduction of data quality errors that were caused by proactive monitoring. Additionally, Marupaka and Rangineni (2024) introduced a predictive data quality assessment system with feedback loops that identified the presence of anomalies before they were carried out using static checks. Nonetheless, the primary limitation of such AI-powered ETL implementations is their scalability and latency: they are either CPU-bound or used offline, respectively, which constrains their performance regarding a large amount of data to be processed in real-time. These solutions have neither been implemented nor able to use GPU acceleration of large data streams, and tend to be ad hoc in integration with orchestration tools. This capability limits this performance gap; hence, this proposed study is interested in employing GPU-enabled auto-encoder models in high-throughput and low-latent anomaly detection approaches in ETL pipelines.

2.2 CUDA and the Rise of GPU-Accelerated Anomaly Detection

NVIDIA, through CUDA architecture, enabled the conversion of GPUs into a general-purpose computing model. Nickolls et al. (2008) describe in their seminal paper CUDA as a scalable parallel programming model enabling its users to write high-performance applications in terms of intuitive abstractions that include thread hierarchies, shared memory and barrier synchronisation between threads. The compilers of CUDA programs can be used to compile and execute programs on thousands of parallel threads simultaneously, and these programs exploit maximum parallelism in the NVIDIA GPUs. CUDA made GPU acceleration of computation-intensive work available to the masses by simplifying GPU programming and allowing its widespread use well beyond graphics. Speedup of more than 100X on CUDA–enabled GPUs has been observed on applications in the field of molecular dynamics and matrix computations. The programming model employed by CUDA is especially suitable for AI tasks such as autoencoder-based anomaly detection, in which bottlenecks are matrix multiplications and high-dimensional reconstructions. Using CUDA, the proposed study can guarantee efficient deployment of deep autoencoder models to real-time anomaly detection within ETL pipelines, hence the connection between AI-based anomaly detection and scalable data engineering infrastructure.

2.3 Accelerated Anomaly Detection Methods on GPU

The performance of the GPU has been proven to be advantageous across multiple domains beyond ETL, especially in real-time anomaly detection. Hewa Nadungodage et al. (2016) created a GPU-parallelised kernel density estimation algorithm that achieved order-of-magnitude accelerations—up to ~20× faster than CPU implementations—for detecting outliers in continuous data streams. Litty (2024) adopted GPU-accelerated AI for robotics and sensor data in an Industry 4.0 setting, enabling real-time anomaly detection for predictive maintenance. This demonstrated the ability of GPUs to support the rapid deployment of complex models that identify even subtle anomalies. Likewise, Ahi et al. (2025) endorsed GPUs in the application of unsupervised anomaly detection for computer vision pipelines, reporting a ~6× increase in throughput. These results confirm that GPU acceleration dramatically improves speed and efficiency in detecting anomalies. However, these efforts have been domain-specific—focused on IoT, robotics, or HPC—and have not explored

integrating GPU-powered anomaly detectors directly into data transformation processes such as ETL. The present study extends these performance benefits to the ETL domain by embedding GPU-accelerated autoencoder models into the pipeline, achieving speedups and responsiveness previously unattainable with CPU-based solutions.

2.4 Towards Integrated Self-Healing Data Pipelines

The utilisation of GPU has been shown to benefit in numerous fields other than ETL, particularly real-time anomaly detection. Hewa Nadungodage et al. (2016) designed an outlier detection algorithm based on the kernel density estimation, targeting continuous data streams, and GPU-parallelised its estimator to achieve order-of-magnitude accelerations over CPU algorithms (up to ~20x faster). In an Industry 4.0 scenario, Litty (2024) leveraged GPU-accelerated AI in robotics and sensor data, which makes it possible to have real-time anomaly detection that can help in predictive maintenance. This showed the capability of the GPUs to facilitate fast implementation in complex models that detect even small anomalies. Similarly, Ahi et al. (2025) have supported the use of GPUs in the deployment of unsupervised anomaly detection on computer vision pipelines, achieving a ~6x throughput increase. These outcomes establish that the speed and performance dramatically increase when it comes to detecting anomalies with GPU acceleration. Nonetheless, these efforts have been application-specific, e.g. IoT, robotics, or HPC-specific and have not looked at the integration of GPU-accelerated anomaly detectors directly into data transformation frameworks like ETL. Following on the success of these performance advantages, the current work is able to carry them over to the ETL space by scaling up the usage of GPU-accelerated autoencoder models into the pipeline, resulting in speed improvements and responsiveness that would have earlier not been possible to achieve under CPU-only solutions.

3. Methodology

3.1 Introduction and Research Design

The following is a Quantitative experimental research inquiry that takes a Design Science approach. The work involves building and testing an artefact – a GPU-accelerated anomaly detection system with a built-in ETL infrastructure intended to solve the practical problem of ETL data anomalies. In DS terms, the pipeline + anomaly detector system is the artefact that has been built to act as a demonstration of a solution to the issue of proactive data quality monitoring. Following Hevner recommendation, design-science research must result in a viable artefact and carefully assess its applicability to a particular problem[1]. According to such a research paradigm, our research implied the development of the anomaly detection

pipeline and its quantitatively assessed performance on controlled experiments. The design of the research is thus iterative and constructive: Developed an ETL pipeline system through which data anomalies can be detected and responded to automatically, and evaluated this system in terms of certain objective measures (precision, recall, and runtime).

A major feature in the research design is the adoption of synthetic data and injected anomalies that are known. This enables controlled testing of both the accuracy of the system to detect anomalies (that are known by the design) and its scale efficiency. The flow of the methodology is organised into clearly defined parts, as described below:

1. Drawing synthetic data to generate realistic test datasets

2. Developing ETL pipeline with anomaly injection

3. Construction of an autoencoder model with graphical processing unit (GPU) acceleration to detect anomalies

4. Framework of an assessment procedure, including methods of measuring detection performance and speed

5. Ethical considerations and limitations. The approach is general but quantitative - we quantify how well the artefact works (detection rates and parsing performance) and are consistent with the design-science method by producing an operational prototype and demonstrating its effectiveness. It is a combination that provides that the methodology not only develops a new solution but also empirically answers the question of the effectiveness of this methodology.

3.2 Synthetic Data Generation

In order to easily replicate and test, synthetic datasets with the Python Faker library have been created. Synthetic data offers the advantage of resembling real-world data while avoiding privacy issues and allowing full control over embedded anomalies [2]. In this paper, we have developed three datasets of different sizes and complexity to simulate different domains (sales, insurance, and banking) and to put the anomaly detection system to the test:

- **fact_sales:** 200,000 records, 6 columns (simulating a fact table of sales transactions).
- **insurance_claims:** 1,000,000 records, 10 columns (simulating insurance policy or claims data).
- **fact_banking:** 600,000 records, 20 columns (simulating banking transactions with high dimensionality).

These records were filled with realistic data (names, dates, monetary amounts, etc.) using Faker providers for each datatype. Faker is an open-source Python package that is tailor-made to generate fake data sets to use in application testing and database bootstrapping [3]. We leveraged Faker to make the synthetic data follow realistic distributions and formats (e.g. valid-looking email addresses, person names, numeric ranges) to come as close as possible to the real ETL source data. The synthesised data is:

Dataset	Number of Records	Num of Columns	Description
Fact_sales	200,000	6	Sales Transactions
Insurance_claims	1,000,000	10	Insurance Claims
Fact_banking	600,000	20	Banking Transactions

Each dataset was stored on a Neon DB instance (PostgreSQL-compatible cloud database) that acts as a source of ETL pipelines. NeonDB was selected based on ease of use with PostgreSQL interfaces, to ensure that our pipelines can run within a realistic database setting (simulated Data warehouse or operational-level DB). The large record count (up to 1 million records) and schema width (6 to 20 columns) is a robust testbed in terms of scalability. It can enable us to determine the performance of the anomaly detection model on a variety of training sets of data and feature spaces. In addition, the multiple datasets can be used to address the generalizability of the method, i.e., the methodology is not specific to one dataset, but to a pattern of ETL with anomaly monitoring that may be relevant to other domains as well.

To conclude, synthetic data generation will provide a quality and secure premise for experimentation. In having the knowledge of the precise content of the data and by managing the anomaly injection (see next section), we can measure the success of the detector quantitatively. This option is consistent with the guidelines on testing data-driven systems, where synthetic data is commonly used to complement actual data to achieve privacy and cost-effective coverage of rare cases [2]. All the synthetic data in the present project was synthesised in a way that does not purport any actual personally identifiable information (PII), thereby removing all the ethical issues related to data privacy, but keeping the scenarios realistic enough to challenge ETL pipelines.

3.3 ETL Pipeline Design and Anomaly Injection

Created individual ETL pipelines associated with the above datasets using Dagster, a modern data orchestration system. Dagster gives us the ability to specify reproducible data pipelines and provide scheduling, dependency functionality, and more, which can be employed in the design. The pipelines are set up to read the data in the NeonDB source, apply the pipeline transformations (including anomaly injection), load the result back into the target database in NeonDB, and then trigger anomaly detection. The ETL pipelines can thereby be regarded as a transformation of a common Extract-Transform-Load process, with an added data quality check in the post-stage.

To describe the flow of the pipeline, there are the following steps that each run must be followed:

1. NEONDB: Extract Source: Query the current synthetic data in a NeonDB source (e.g. read the fact_sales table).

2. Transform (with anomaly injection): To mimic real ETL (e.g. casting data types, aggregations) and inject anomalies into the data at this point.

3. Load: Copy the transformed data back into a location in NeonDB (this would be the pipeline output location that would be used by downstream systems).

4. Anomaly Detection Trigger: Automatically scanning the newly loaded data with the GPU-based autoencoder in order to identify possible anomalies.

5. Alerting and Logging: In case anomalies are detected, log details of that anomaly and also send out the email alert and store the same logs in the S3 bucket.

The anomaly injection in step 2 is an important aspect of the experimental configuration - systematic injection of errors in data that should be detected by the detection model. To introduce the anomalies, we have attempted anomaly injection as a random operation: records that go through a data transform stage have a small percentage of records randomly chosen and corrupted. Among the anomalies to be injected are: - Null or missing data: Replace some values with NULL to depict missing data. - Outliers: If the field is numeric instead, replace a few infrequent values with some unusual out-of-range number (i.e. a really big number or a negative number where one is not supposed to exist on a temperature scale). Garbled: On categorical / string type fields, add garbled text or junk character strings (as encoding errors or inconsistent fields)

These types of anomalies represent the usual data quality problems that may arise in pipelines (missing entries, out-of-bound values, corrupted strings). Proactive injecting them will then help us gauge the ability of the system to identify problems that would otherwise create a problem with data integrity. The injection rate is minimal (i.e., less than 1 per cent of the records are perturbed) so that anomalies are outliers in the injection data. This corresponds to real-world values in which genuine anomalies are rather infrequent occurrences. Compared to other approaches, it should be noted that our method fits within a general data quality monitoring strategy, which seeks abnormalities or rule non-conformance in ETL [4]. During ETL audits, data engineers specifically seek to understand such difficulties, such as missing values or unexpected spikes, and in our experiments, we do the opposite by constructing such difficulties systematically.

The pipelines are scripted in such a way that the anomaly detection (step 4) becomes automatically activated once the loading of data is complete. We make use of Dagster event handling, namely a sensor/trigger upon pipeline state, to trigger the anomaly scanning once the data is ready. Dagster includes pipeline sensors that can be used to trigger an event when an asset (such as a table) has been updated[5], and we use this feature to seamlessly integrate the machine learning model into the ETL process. Essentially, the ETL pipeline and anomaly detector constitute a single workflow: after the data is loaded, either the same pipeline run or a closely related downstream job launches the anomaly detector on newly loaded data. Such a design can lead to proactive anomaly detection - anomalies are detected only moments after the ETL job has completed, as opposed to much further in time when the data was already acted upon as part of an analytics exercise.

Fig. 1 Architecture diagram

The procedure starts with the generation of synthetic data and the loading of the latter into the NeonDB source. At this point, Dagster masses the ETL task: the data is read, transformed (including random anomaly injection as marked by the ⚠ symbol in the image), and written to the target table. Once loaded, an event runs the GPU-accelerated autoencoder model to scan the data to detect anomalies. Any anomaly identified (red stars in the figure) is automatically logged, and an email notification is raised to stakeholders. The recorded data in the anomaly log format with timestamps, record ID and the anomaly information is archived in an AWS S3 bucket to allow continued record keeping and further analysis. This architecture is to guarantee that any presence of data anomalies or faulty datagrams can be identified and corrected in nearly real-time before the fault data can be moved to downstream applications. This pattern is modular, and the same can be applied to any dataset and can be readily applied to other data pipelines with minimal modifications due to the orchestration used by Dagster.

3.4 Autoencoder Model and GPU Configuration

The anomaly detection engine is an unsupervised autoencoder neural network based on PyTorch, run on an NVIDIA GPU via CUDA. An autoencoder was selected on the basis of its demonstrated power in the detection of anomalies in all forms of data[6]. It operates by training on how to encode and decode the input data; something abnormal can be detected when the model fails to decode a data point successfully, and a high encoding error is observed. The method is domain-independent - although autoencoders are conventionally presented using image data, they can be used with any kind of data, including tabular in our example.

Model Architecture: We trained a distinct autoencoder on each dataset (to make them suitable for different schema shapes). The structure of each autoencoder is a fully connected feed-forward network whose encoder dimensionally reduces the input features and the decoder dimensionally expands back to the original dimension. As another example, the architecture of

the insurance_claims (10 features) model includes an encoder tapering to a bottleneck of 4 neurons, and the decoder tapering back up to 10 outputs. The architectures of other datasets resize similarly (e.g., ~3-4 neurons bottleneck in the 6-feature sales data, ~8-10 in the 20-feature banking data), leaving around 40-50% of the dimension of the inputs at the bottleneck. The activation function in the hidden layers of all models is ReLU, and the output layer is linear. The training criterion is to minimise the Mean Squared Error (MSE) between the input and recreated output, and this is a typical loss in autoencoder-based anomaly detection [7]. In both cases, we trained autoencoders with normal data being alone in the data set, i.e., without injected anomalies. In practice, this was possible to overcome by using a clean copy of the synthetic data (prior to injecting anomalies) to train on, or to remove the anomalous records that have been injected in the case where training occurs in the pipeline. The training on non-anomalous training data is consistent with the recommended practice, to make the model learn the pattern only of legitimate data[8]. Discarding anomalies during a training process leads to an autoencoder that will attempt to extrapolate the image of a typical case; any data point which does not fit the typical profile very well (at the time of deployment) will lift the reconstruction error, signalling it as a potential deviant. On each clean dataset, we used 80 per cent as training and 20 per cent as validation. Training was of a fixed number of epochs (e.g., 100) until the reconstruction error on the validation set was no longer improved. The last models reported low reconstruction MSE on validation sets, which means that they generalised well to base their learning of the normal patterns.

GPU acceleration: The anomaly detection phase is a performance-sensitive stage, as there is a large amount of data (hundreds of thousands to millions of records) that needs to be scanned within a short period of time after each ETL run. We thus modified the PyTorch models to execute using a CUDA-enabled GPU (NVIDIA GTX1650 in the case of our test setup, 16 GB GPU memory). Running the experiment using the GPU instead of the CPU substantially decreases the time of training and inference (scoring) of the autoencoder. As an example, the operations involving large computations with matrices (such as calculating the outputs with 1,000,000 rows and 10 features are very parallelizable on a GPU. This provides significant time-saving: the previous work shows that a GPU-based system to detect the outliers is an order of magnitude faster than the CPU-based algorithms and can efficiently process millions of points [9]. In our setup, the GPU-accelerated pipeline inference is in the order of seconds to tens of seconds per dataset, whereas inference with a CPU-only platform may take minutes on the largest dataset (we have seen up to ~10 times inference speed-up by GPU in benchmarking). The ability to process data quickly is key to enabling near real-time notifications of detected anomalies; an application of this premise is NVIDIA Morpheus, which uses GPUs to process data streams up to 600 times quicker than conventional systems [10]. Although our use case is not streaming-oriented (after-ETL job), the same principle is being applied; GPU acceleration allows us to analyse large batches in time shortly after load.

Anomaly detection and alerting: On every ETL load, the trained autoencoder is run in a forward pass on the new data. On each record, the reconstruction error (MSE between output and input) is computed. We set a concluding line on this error to categorise any anomaly. This threshold could be calculated by analysing reconstruction errors in a validation set, or could be set to capture a low percentage of the highest-error detections (there being so few anomalies). In our implementation, we first adopt the threshold of the 99th percentile of reconstruction error on the training/validation data (this is expected to capture the most extreme 1 per cent of

cases). We then refined this threshold by examining a sample of known injected anomalies to meet the condition that true anomalies have high recall (see Evaluation section). Any error data record above the threshold will be flagged as an anomaly. All these flagged entries (record identifiers and error values) are collected by the system. Once the anomaly detection step finishes, the pipeline node will: - Log the detected anomalies into a structured log file (including which dataset, which record ID or primary key, and what type of anomaly if it can be identified). This log will be stored on AWS S3 on bucket as a persistent store. Storing in S3 enables additional analysis, and it also discourages the storage of the pipeline execution machine. Send an **alert email** to a configured list of recipients (e.g., data engineers). The email contains a summary of the findings (e.g., "Anomaly Detection Alert: 5 anomalies detected in the *sale* dataset after ETL job on 2025-08-21. See log for details.") along with different CSVs which describe nulls, duplicates, missing records, outliers and schema drift values in the dataset, respectively.

The autoencoder model and its deployment are thus tightly integrated into the pipeline. Thanks to the GPU acceleration, this integration does not substantially slow down the ETL workflow, the detection runs quickly after data loading. The entire methodology here exemplifies a design-science artefact: we created a working system that not only transforms and loads data but also intelligently monitors data quality in real-time, leveraging modern ML and hardware acceleration. The next section outlines how we evaluate the effectiveness of this system in quantitative terms.

3.5 Evaluation Metrics

We considered the anomaly detection pipeline based on detection capability and efficiency in a series of quantitative experiments. Because anomalies were intentionally added to the target datasets, the assessment can be constructed as a detection task and evaluated with a set of suitable data quality and performance measures. Reconstruction Error Statistics, Anomaly Detection Rate, Training Loss and Runtime are selected as the main measures, and they are characterised in the following way:

Reconstruction Error Statistics: The normal records are well modelled by the autoencoders as indicated by the distribution of reconstruction errors (mean, standard deviation, minimum, and maximum) values. Anomalies are records that have the presentation of abnormally many errors. These statistics promote the identification of the aberration level in each dataset.

Anomaly Detection Rate: The percentage and numbers of records classified as abnormalities, with the percentage per the total number of records. As an example, the Banking dataset presented anomalies of about 1 per cent and the Insurance one of about 0.5 per cent. The measure directly indicates the number of injected anomalies that were captured successfully.

Training Loss: The Mean Squared Error (MSE) loss attained by the autoencoder on the last training epoch. The final training loss is low (e.g. 0.0062 in the case of Banking, 0.000389 in the case of Insurance), showing that the autoencoder was successful in recreating normal data, which provides a strong baseline to detect anomalies.

Rule-based Validation Metrics: In addition to autoencoder detection, the data quality checks that were made traditionally were also recorded:

Null Counts- the number of missing entries in important fields (source and target).

Duplicate Counts – number of duplicate records in source and target.

Missing Records- source records that were not found in the target.

These checks also monitored structural data quality problems in parallel with anomaly detection.

Runtime: The run time of the anomaly detection step (GPU-based inference and reconstruction error logging). Indicatively, 14.5 seconds were taken to execute Banking validation, and 12.3 seconds were taken to execute Insurance, indicating that the technique is efficient enough to be operated in near real-time.

These metrics were selected to give a healthy assessment of the accuracy of the detection (in terms of the number of anomalies and error statistics) and the efficiency of operation (in terms of run time). Though the system theoretically is a binary classification problem with Precision and Recall, in this research, we focused on quantitative results in anomaly detection (counts, thresholds, and error distributions) and operational efficiency. This is a twofold objective of the thesis: to identify all the injected anomalies and make the pipeline still practical to the real-world ETL workload.

3.6 Ethics and Limitations:

Ethical considerations: No sensitive data or personal data was considered; all records were generated artificially to resemble realistic patterns but not referring to specific people or organizations, thus ensuring privacy by design. The benefits of the system in operational terms are that it should increase the quality and acceptability of data, which is of a positive ethical nature, as it may not allow the decision-makers to act upon the flawed data. However, when applied to real data, it is important to be transparent about false positives, i.e., when an alert is produced, data engineers may need to check the anomaly to prevent an otherwise unnecessary action (this would be more of a best practice than an ethical dimension, but it is related to responsible utilization of automated alerts). The email alert process will need to be set up such that it adheres to privacy and security (sending sensitive information in alert messages and so on), but in our scenario, there is no sensitive information in the alerts; instead, only synthetic identifiers. In general, the analysis involving synthetic data and focusing on the system performance does not force the researcher to neglect ethical conduct in any aspects related to the study, as there are no human subjects and the information is not confidential.

3.7 Comparison of different approaches:

The specific approach of research applied in this study -quantitative experimental design and a design science framework- presents a clear contrast with a qualitative or case study-based approach to research. In contrast to qualitative work that uses survey, interviews, or observation data to arrive at subjective findings, quantitative research is based on measurable input, and the outcome in this case included precision, recall, and run-time evaluation of the anomaly detection pipeline performed using GPU-accelerated software. The focus on empirical data and

adherence to statistical measures make it possible to prove the effectiveness of the system under consideration in detecting anomalies objectively and contribute to repeatability since all experiments can be repeated in unchanged conditions. In addition, the design science element corresponds to the engineering of a real, practical thing, a functional ETL pipeline that proactively detects anomalies, and which is assessed in a rigorous manner, and not merely described or observed. The metrics-driven methodology has a structure that suits both technical and performance-oriented research questions like the one addressed in this project.

Unlike in case study methodology, which is interested in the intense examination of a particular situation or organisation scenario, this research is not concerned with any real-life organisation or production system. Rather, it cross-tests on a variety of synthetic data representing other domains (sales, insurance, banking) in order to assess the generalisation and the extendibility of the designed solution. Case studies supply rich contextual awareness and can be used to explore or explain, but are always very restricted and not generalizable. The design science approach employed here allows us to develop a system architecture that can be repackaged to be used in other domains, but the experimental aspect of it permits us to compare performance in a contained environment. This predisposes the identified methodology to the questions of how well the solution works rather than how this process plays out in practice, a distinction that justifies the rejection of both case study and qualitative types in this research context.

4. Implementation:

This chapter describes the design and development of the GPU-Accelerated Anomaly Detection in ETL Pipelines in detail. We also describe about the system setup and configuration, the development of the anomaly detection model, the infrastructure experiments, the data validation, which was integrated into the ETL pipeline, the architecture, and training of the autoencoder model, as well as how the work on validation and anomaly reporting can be automated. The application is done in close association with the best practices and the performance and reliability requirements of the project.

4.1 System Configuration and Setup:

The solution was deployed on a local computer with an NVIDIA GeForce GTX 1650 Ti graphic card. This and the 1024 cores and 4 GB memory of this GPU provided the parallel computing capacity necessary to speed up model training and inference. The local environment was powered by 64-bit Windows 11, Python 3.10 (Anaconda distribution) and configured with the CUDA Toolkit (v11.x) and its associated cuDNN libraries to allow compatibility with deep learning models. PyTorch framework v1.x was installed, including CUDA support, which enabled the use of the anomaly detection model to utilise the power of rasterising through the use of a graphics card. This architecture allowed the pipeline to work with big data effectively, with the ability of PyTorch to run computations on large datasets greatly accelerated by using the GPU[1]. To be consistent with the data processing (e.g., pandas to process the data via ETL) and orchestration (Dagster APIs) libraries used were installed in a separate virtual environment.

ETL and validation pipeline were managed and scheduled using a local version of Dagster (an open-source data orchestrator). Dagster was chosen because of the fact that it allows coordinating complex workflows and its lightweight, developer-friendly execution model[2]. The Dagster UI and scheduler were operating on the local machine, coordinating pipeline runs

and offering monitoring. This design implied that all ETL processes, anomaly detection model, and data quality checks were run in the same environment as possible, with a minimum of external dependency. The networking was configured to enable the pipeline to reach such resources as the Amazon S3 bucket (to store logs) and an SMTP server (to send email notifications). In general, the system architecture made sure that the GPU components were made available to the anomaly detection model, that the appropriate drivers and libraries were installed, and that the orchestration framework was capable of collaborating with these components to start and control every part of the pipeline.

4.2 Model Evaluation and Decision Rationale:

Isolation Forest as the Approach: The initial method of detection of anomalies was to apply a classical machine learning model, Isolation Forest, to determine outlier records within the data in the first approach of the project. Isolation Forest is a collective tree algorithm that isolates anomalies by selecting data points at random; anomalous points are rare and distinct and therefore are easily isolable (they are more likely to exist in shorter branches running through them)[3]. It was implemented based on the IsolationForest class of scikit-learn, which was trained on a sample of the combined dataset (insurance claims, banking transactions and sales records). This was because the Isolation Forest is unsupervised (does not need labelled anomalies) and has been known to scale well to generic anomaly detection and is relatively CPU-fast on moderate data sizes.

Limitations of Isolation Forest: Pragmatically, this strategy demonstrated a number of weaknesses. First, the Isolation Forest model failed to map the complicated feature relations in the multi-domain data. The data contained high-dimensional features (numerical and categorical characteristics) whose correlations were complex and that could not be fully captured by a simple tree partitioning. We noticed that the Isolation Forest was more likely to declare many points as anomalies, most of which, when examined, ended up being normal but simply at the end of distributions of individual features. That is, it had a high false-positive rate, and this could be due to the fact that traditional statistical/ensemble techniques cannot adequately capture highly complex data structures[4]. Also, the Isolation Forest approach was CPU-intensive and could not be accelerated by a GPU. When the dataset expanded to the hundreds of thousands of records, the time to train, and, in particular, to score the Isolation Forest on each batch became meaningful. This brought up issues of scalability and real-time application; Isolation Forest can be faster than some algorithms, but cannot utilise parallel computing on a GPU, which a neural network-based approach could. These reasons demonstrated the importance of a more robust and adaptable anomaly detection method.

Transition to Autoencoders (Unsupervised Deep Learning): The project was reoriented to an autoencoder-based anomaly detector due to the drawbacks of the first. A deep neural network is an unsupervised autoencoder that is trained to encode input data into a lower-dimensional latent variable and to decode it into the original input. The point is that the model will get to know how to reconstruct normal data effectively, but will not be able to reconstruct anomaly data correctly, which means that the reconstruction error on anomalies will be larger. This method is suitable to high-dimensional, complex data: with complicated data structures, existing methods might fail to perform well, but autoencoders can be effective at modelling and recreating the typical behaviour of data without supervision[4]. There were no labelled anomalies in our case (we were not aware initially which records were fraudulent or erroneous),

and thus an unsupervised learning approach was suitable. The adoption of a Deep Learning model was also driven by the fact that the model would be executed using a GPU to accelerate. The autoencoder could be trained and inferred on the GTX 1650 Ti, which is much faster, and we can process large amounts of data within a decent amount of time. The autoencoder architecture was easy to implement and experiment with using py Torch dynamic computation graph and CUDA, and could easily be deployed in our pipeline using efficient hardware resource utilisation.

Altogether, the Isolation Forest was a good beginning, but it lacked precision and correspondence to our overarching aims of incorporating our design with our GPU-accelerated capabilities. The extension to an autoencoder was motivated by the requirement to model the normal data distribution more accurately and the wish to take advantage of parallel computation. As neural networks, autoencoders provided a more expressive model that could learn finer details in features, and reconstruction error as an anomaly score was naturally compatible with the unsupervised character of the problem (large deviations between original and reconstructed data are evidence of anomalies). This model progression corresponds to a general movement in anomaly detection to replace classical algorithms with unsupervised deep learning, to enable better results on complex data.

4.3 Infrastructure Experiments and Challenges:

Implementing the GPU-accelerated anomaly detection in practice involved several infrastructure experiments. We explored cloud-based options to run the model on GPU hardware and eventually settled on a fully local deployment after encountering various challenges. Below, we outline the approaches tried and the rationale for the final setup:

Google Colab Attempt: The first infrastructure trial was to offload the anomaly detection computation to a cloud notebook service, **Google Colab**, which offers free access to GPUs (typically NVIDIA T4 GPUs) in an online environment. The plan was to trigger the anomaly detection script from the local Dagster pipeline and execute the model on Colab's GPU. This would allow us to use Colab's powerful hardware without incurring cost. We attempted to integrate this by using Dagster's ability to call external services or APIs (e.g. through a Python client that could send code to a Colab runtime). However, this approach met strict **security restrictions and network limitations** in Colab. Google Colab sessions run in an isolated environment that does not readily allow arbitrary remote code execution triggers from external sources for security reasons. Our pipeline's need to programmatically launch a Colab job and retrieve results proved infeasible. Even after exploring workarounds—such as using Colab's REST API with proper authentication tokens, upgrading to Colab Pro+ for longer runtime, and mounting Google Drive for data exchange—the integration remained brittle and not suitable for an automated production workflow. Colab sessions are ephemeral and require manual or script re-initialisation, which clashed with the goal of a seamless, orchestrated pipeline. In the end, the Colab experiment was abandoned due to these integration challenges and the lack of a reliable way to incorporate Colab into a continuous pipeline run.

AWS EC2 GPU Instance Attempt: AWS EC2: We tried Colab and then went to Amazon Web Services EC2, where we could rent a virtual machine with a GPU. The types of instances that we considered included g4dn.xlarge, which provides an NVIDIA T4 GPU (compute

comparable to that of Colab) as well as dedicated CPU and memory. The advantage of an EC2 solution was that we had complete control of the environment, we could create the OS, drivers and networking to be in the same state as on our local host, and we hoped that we would have fewer integration problems because we could run Dagster and the model directly on the instance as though it were just another server. But then the main hurdle was cost and logistical constraints. Opposite to Colab, AWS does not offer a free tier of GPU instances or an instance on g4dn.xlarge (or any other similar) costs considerably per hour. Since our anomaly detector would have to operate on a regular basis (possibly daily) and we would have to experiment with numerous test runs, we could not afford the charges of an EC2 GPU over the span of the project. There was also the challenge of mass transfer of data to the cloud; the insurance, banking and sales datasets would either need to be stored on AWS or uploaded every run, which would contribute to time and cost. The AWS path was determined to be too costly with regard to its long-term use, as was considered. To prevent accruing fees, we de-provisioned the trial instance and switched attention to the on-premises solution.

Local GPU Deployment: After the success of the cloud tests, we switched to local GPU execution (the same platform as described by the setup, but with the GTX 1650 Ti). This was made the final implementation solution. The localisation allowed us to have full execution and environmental consistency: the Dagster orchestration, ETL steps, and the anomaly detection model were executed on the same machine. We set the pipeline in a way that after the ETL jobs were run, the anomaly detection job would run on the local GPU. To make this successful, there were several considerations that had to be made. We have verified the presence of the CUDA drivers and runtime installed properly on the machine and that PyTorch was able to recognize the GTX 1650 Ti (successfully, as torch.cuda is available, changed to True and the GPU name was reported as NVIDIA GeForce GTX 1650 Ti). We also needed to be careful with GPU memory since the 1650 Ti had a 4GB limit; the model and batch sizes were selected to fit into memory without causing an out-of-memory error. The local setup was found reliable once configured: it did not circumvent network barriers in Colab, and did not add any extra cost. The results were good, e.g., on the GPU, the autoencoder validation (forward passes needed to calculate reconstruction errors) took approximately 14.5 seconds when running on a full pipeline with all datasets (i.e., 602,883 records in total). This demonstration was a direct confirmation of the GPU acceleration strategy because the same operation done on the CPU was seen to be much slower during preliminary tests. Local deployment also made debugging and iteration simpler, as we could, at a glance, look at logs and system status in a single location.

To sum up, although cloud-based GPU solutions were on the list, the security, integration, and cost conditions made us use a local solution. The entirely localised environment took great care to be set up, yet ended up offering a stable, quick, and economical environment in which the anomaly detection pipeline would be run. This experiment highlights that in some projects (particularly not at scale), a well-packaged local GPU may be a better choice than cloud services, or provide more flexibility to test and experiment without fixed monthly fees.

4.4 Data Pipeline Implementation:

Three different datasets are retrieved and processed on the ETL pipeline: insurance claims, banking transactions and retail sales, which are then integrated and validated before the process moves to anomaly detection. During the implementation, there is a sequence of data validation and anomaly detection that is initiated after each ETL run (extracts data, transforms it, and inserts the clean results into a target database or file). This order makes the data come out the same and notifies of anomalies. The key part of this pipeline segment is as follows:

1. Data Validation Checks: Data keys are a set of data quality checks that we applied to the ETL output to identify typical problems like missing values and duplicated records. These are known to bias analysis and have to be resolved in order to retain faith in the data. The validation checks were:

• **Null Value Detection:** This is a scan of the pipeline that detects null or absent values in each dataset in critical fields. As an example, in the insurance data, such fields as the claim ID or claim amount cannot be null; in the banking data, the time of the transaction or IDs are to be present. Any row in which the essential column values are null is written to a nulls.csv report in that particular dataset. We adopted a safe precedent of flagging over such records to review them than dismissing them without notifying them. (Key fields should not contain null values, which can cause havoc downstream unless addressed, so it is safer to make them visible in the logs so the data engineers or the domain experts can take notice of them.)

• **Duplicate Records:** The pipeline is used to check the instance of duplicate entries in the output data, using unique key fields. As an illustration, in the banking transactions data, the transaction-id is intended to be a unique field; in case the ETL accidentally generated redundant transaction records, they would be identified. Duplicate records may also skew metrics and analysis by counting events twice. In case of any duplicates, a duplicates.csv log (containing information about the record fields) is logged. In our implementation, some duplication of the source data was already done, thus the vast majority of the duplicates that we had detected were caused by the consolidation of more than one source or by a data processing aberration. Their log will enable them to investigate and prevent loading bad data into analytical systems.

• **Missing Records:** To verify completeness, we introduced a procedure for missing records check by comparing the number of records and keys in the sources and the targets. On each dataset, the number of records loaded post-ETL was compared to the number of records extracted from the source. Discrepancy was recorded when the counts did not match, or when selected anticipated records (identified by primary key) were not found in the target. An example is to extract 10,000 insurance claim records and then load 9,950 records; the pipeline would determine the records not loaded (50) and would list the IDs in a report called missing_records.csv. Data loss through the missing of data is a very big issue since without it, the information is lost; unless checked, it can result in incomplete analysis outcomes. With a check of record numbers and the entry of any loss records, we build an audit trail to make sure that no information has been accidentally lost in the process of ETL. Practically, a small amount of records were occasionally filtered because of the data cleaning rules (e.g., malformed records), though these were excluded on purpose. All missing records that could not be explained were considered possible errors.

All of these validation steps were applied as Python functions (op) in the Dagster pipeline and worked with the pandas Data Frame results of the ETL. Rather than stopping the pipeline when problems are detected, the design records the troubles and proceeds to anomaly detection (an anomaly model can still operate with, say, a few nulls or duplicates; those are orthogonal). This style puts more emphasis on observability than interrupting data flow - any observed quality problems in data are stored to be fixed later, but do not abort the pipeline execution. This structure goes in line with sound practice of the ETL in which the data quality checks serve as monitoring/alerting tools and not necessarily the means of a hard stop (except in critical situations).

2. Integration of Anomaly Detection (Autoencoder) into the Pipeline: After basic validation, the pipeline proceeds to the autoencoder anomaly detection stage. In this case, the validated and cleaned data of every domain is inputted into the trained autoencoder model to detect anomalous records. We combined this with the anomaly detection being another task in the Dagster workflow, which was conditional on the completion of data validation tasks. On a per-dataset (insurance, banking, sales) basis, the following sub-steps are performed by the pipeline:

Data Preparation: The data is first prepared into the format upon which the autoencoder is to operate. This included scaling numeric data (e.g. min-max scaling to or standardisation), encoding categorical data (e.g. one-hot encoding, or embedding lookups where the model allows). The same transformation which was applied in the process of model training is applied in this stage so that there is consistency. The ready information (normally a numpy array or a tensor) is then transferred to the memory of the GPU to be processed rapidly.

• **Reconstruction Error Computation:** The PyTorch-based autoencoder model, which occupies a hint of the memory of the GPU, takes the data and calculates reconstructions. Given a record of inputs x, the model produces a reconstructed record x'. The reconstruction error measure is used to determine the difference between x and x. We then implemented using Mean Squared Error (MSE) per record (means across all features) as the anomaly score. This gives a list of error values, one per record in the dataset. This is one of the most parallelizable steps on the GPU - thousands of records can be batched - hence the calculation is fast even on large data sets.

• **Flagging Anomalies:** The error of each record is then compared to a predefined threshold in order to check whether it is an anomaly. When the error > threshold then the record is a flagged anomaly. The threshold was established by means of experiment with the training error distribution (how it was selected is discussed in the next section). This step of the pipeline merely uses the cut-off value on the array of errors. What is left is a binary classification of every record: normal or anomalous.

• **Recording Anomaly Results:** The results of all detected anomalies are stored to an output file (e.g., autoencoder_anomalies_insurance.csv with the insurance dataset and analogous with banking and sales). These CSV logs have the record identifier, the relevant fields and the reconstruction error value. As an example, an anomalous insurance claim record log may include the claim_id, policy information and the error score. By logging the anomalies in a different log we can easily review what has been flagged. In development, it assisted in

manually determining whether the anomalies were reasonable (e.g. claims with suspiciously high amounts or transactions at odd times with atypical schedules were one of the anomalies identified). Overall, among all datasets, the autoencoder detected approximately 1 percent of the records as anomalies - e.g., out of 602,883 records in total (consisting of 602,883 combined records), approximately 6,029 records were identified as anomalous (the exact percentage differs by dataset).

It should also be noted that, instead of trying to design a single model that would be compatible with all data, we have trained the autoencoder on each type of data. The reason is that the feature sets and distributions are different (insurance claims have fields where transactions do not and the other way around). An autoencoder was trained on each domain data, so that the model would only model the normal profile of that domain. These are organized in parallel or sequence by the pipeline (Dagster can run things in parallel, but in this case GPU was the only resource; therefore we run them in sequence to prevent contention). The final output of the data pipeline implementation is a detailed validation and anomaly report of each ETL execution: any nulls, duplicates, missing records and anomalies are all recorded. This meets the purpose of the project which is not only to transform and load data, but also to validate the data and put emphasis on outliers as an automatic process.

Model Architecture and Training:

Conceptual architecture of the autoencoder model used for anomaly detection. The encoder compresses input features into a latent code, and the decoder reconstructs the input from this code, enabling detection of anomalies via reconstruction error.

The selected anomaly detector is an autoencoder neural network of feed-forward type. In design, it is made up of two primary parts, an Encoder which reduces the high dimensional input to a lower dimensional latent representation, and a Decoder which recovers the original input given the latent code.

Architecture details: We used a dense (fully-connected) autoencoder architecture. In each dataset, the input layer size is the number of features in that dataset, post-preprocessing (i.e. post-encoding categoricals and scaling), so the insurance data consisted of the order of 20 features, whereas the banking transactions consisted of a different number. The encoder is a series of dense layers that produce a dimensional reduction. Our last model employed two hidden layers in the encoder, the first layer downsizes the input by about half, and the second layer further downsizes it to the latent dimension (bottleneck). The latent dimension was a hyperparameter that can be tuned; we chose its size to be small (8 or 16), which encourages the model to learn a compressed version of the most important factors of the data. The encoder layers employ the ReLU activation functions (Rectified Linear Units), which provide non-linearity in order to assist in capturing the intricate patterns. Latent code is obtained after the final layer of the encoder.

The encoder resembles this decoder as a mirror. It commences with the latent vector h (the compressed representation), and uses dense layers to re-expand it to the original number of features. To be as symmetric to the architecture of the encoder as possible, we used an identical number of layers in reverse (two hidden layers). The decoder's last output layer yields a

reconstructed vector (also known as x-hat (an approximation of the original input x)) of the same size as the input. We employed the use of a sigmoid activation over the output layer when that feature was normalised between 0 and 1, and linear outputs when that feature is standardised to permit reconstruction over the entire range of values. Essentially, the decoder attempts to recreate every input feature as well as possible. The overall structure is rather shallow and broad which was adequate because of the size of the input features; we did not use deep convolutional or recurrent layers, as our data is tabular and a feed-forward network is the most adequate.

Training Procedure: The autoencoder was trained in an unsupervised way on a set of data that was assumed to be mostly normal. Practically, we used the ETL output of historical runs (with the assumption that the overwhelming percentage of records are normal) as training data. We also tested whether outlier extremes were removed in the training set (Isolating Forest results or simple heuristics) to prevent learning on blatant anomalies. The training goal is to reduce the error in the reconstruction of the training examples. We adopted the loss function of Mean Squared error (MSE), which computes:

$$L = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

Overall features for 'i' records, and the optimiser aims to minimise the overall training loss across all records. The model was built (compiled) and trained in PyTorch with Adam optimiser (a variation of stochastic gradient descent) and a learning rate of 0.001. In our experiments, 100 epochs were found to be enough to achieve convergence with each autoencoder; the training loss leveled off at about 80 epochs, and no overfitting was apparent (the loss on a held-out validation set of records also levelled off). The last training loss was of the order of 0.001 (e.g. about 0.0062 MSE at the final epoch using one dataset), implying that the model can be used to recreate normal data with a high level of accuracy.

During training, we trained and at the same time monitored loss (and sometimes distribution) of reconstruction errors on a validation set. This was critical in deciding the anomaly threshold. Our evaluation of the autoencoder was done on a large validation dataset (which we assumed included at most a very small number of anomalies). Each record was reconstructed with an error. The errors in reconstruction were generally skewed to the right: most of the records had a very small error (near 0), although there was a long tail of larger errors. We selected a threshold by statistical heuristics: one approach was to use, as the cut-off, the mean reconstruction error plus a multiple of the standard deviation, but, since the error was not a Gaussian distribution, a more appropriate method was to use a high percentile of the error values in the cut-off (e.g., the 99th or 99.5th percentile). We finally established the anomaly threshold to be around 0.0287 in normalised MSE units, which was the point where the error distribution began to exhibit an apparent inflection (which was near the 99th percentile of the training error distribution). Any record containing an error greater than this threshold is regarded as abnormal. This cutoff resulted in just over 1 per cent of records being flagged, and this is as expected since true anomalies would be uncommon. We confirmed this decision by

spot-checking: some of the well-known “edge cases or potential anomalies (such as in the insurance data, certain claims marked as fraudulent or exceptionally high) did cross the threshold and were flagged, and normal records did not.

Model Testing and Refinement: Once the threshold was established, the model was incorporated into the pipeline and used on new ETL outputs. The initial few runs were considered a validation cycle of the model itself. Reviewing the identified anomalies was done in consultation with the domain knowledge (where possible) to determine whether it was plausible. This resulted in some reiterations: the original model was oversensitive to gentle variations, e.g. it originally indicated some of the transactions that were in fact harmless but merely happened to contain slightly odd combinations of attributes. As a reaction, we slightly increased the threshold upwards, and we increased the scaling of the input feature. Another aspect that we made sure that the autoencoder model would be re-trained periodically. As ETL pipelines receive new data as time passes, by retraining the autoencoder on the most recent data, it can be adapted to any changes in normal operation patterns (such as seasonal sales data trends). In a production case, a new model may be retrained on the newest clean data after every week or month. This retraining step is an offline process implemented by us and can be activated on demand; in the scope of the thesis, we have conducted a couple of retraining sessions to ensure that the model’s performance did not deteriorate with time.

To conclude, the autoencoder architecture and the training have been designed to make the most of its usefulness in the context of anomaly detection in our ETL situation. With a comparatively straightforward neural network, with adequate capacity to learn data trends and train it with the help of a GPU, we ended up with a model that best encapsulates the nature of each dataset in its normal condition. The minimisation of reconstruction loss in normal data was the aim of the training process; the evaluation process (with a selected threshold) transforms it into a working anomaly detector. After training this model, it was saved and deployed as part of the pipeline, allowing it to quickly make an inference on a new batch of data to ensure the continuation of data quality.

Automation and Output Delivery:

Automation of the process of detecting anomalies and validating the data was one of the main goals of the project to make it operate according to the ETL pipeline without human intervention. This has been achieved through the orchestration functions of Dagster to organise activities and through the use of intensive logging and notification systems of the outcomes. The last configuration is such that once an ETL pipeline is executed, all validation tests and the anomaly detection model should automatically execute, and the results should be stored and provided to stakeholders to view. The elements of this automation are as follows:

Dagster Orchestration: We declared all the ETL+validation+anomaly detection sequence as a Dagster job. The dependency management of Dagster was employed to ensure, e.g. that the anomaly detection step will only be used after the data quality checks have been successfully completed and the latter step after the ETL load has been successfully completed. This ensures proper sequencing and eases error management (when ETL fails, the downstream validation and model will not even execute). The advantages of using Dagster resemble those of an orchestrator: the pipeline may be scheduled to run at a time (e.g., nightly), the pipeline may be viewed through a UI, and each step in the pipeline may be re-run as necessary. We established a schedule whereby the pipeline is triggered every day at midnight (under the assumption that

new data arrives daily), but also provided manual triggering via the Dagster web interface to run it on demand. The lightweight local scheduler at Dagster was also an adequate choice, and we set it up to initiate the pipeline run in a separate process with access to the GPU (so that it was not the Dagster daemon that consumed the CUDA device). The logging inside each Dagster op was turned on, so that we can take any runtime errors or metrics. Examples: when running the anomaly detection op, we record the number of anomalies detected in each dataset in Dagster events (on top of the files created). Both in a sense, Dagster is the automation backbone, chaining together all pieces and making it possible to execute the full workflow of validation and anomaly detection with a single click (or scheduled entirely).

Output to Amazon S3: The results of all the output logs in the pipeline are saved automatically in an Amazon S3 bucket. We installed Dagster and AWS S3 by installing Dagster S3 resource (authenticated with AWS credentials) to save the files on the local run to the cloud. S3 has been selected because of a number of reasons: it is very resilient, available and serves as central storage that can be accessed by different members of a team or systems. The S3 integration will give the storage layer of the pipeline a resilient storage capability i.e. had the local machine failed the results would be securely stored in the cloud.

Within our implementation, we write the following CSV files to the S3 bucket at the conclusion of each run: nulls_report.csv (which is an aggregation of any nulls across datasets, or a per-dataset file as needed), duplicates-report.csv, missing-records.csv and anomalies-report.csv. CSV format allows the logs to be readable by humans and, more importantly, readily consumed by any BI or data analysis tool to study the data further.

Saving these outputs in S3 enables long-term archiving of the results of the validation you can retroactively investigate anomalies or data quality problems of the prior runs, which is valuable to traceability and auditing. To ensure data confidentiality (at least because the transaction or claim data can be sensitive), the S3 bucket was configured to allow access to the logs only to authorised users (or applications). To conclude, S3 is the primary data store of data quality records in the project, and its scalability and durability capabilities will ensure that our validation logs are safe and well-organised.

Email Notification and Reporting: Coupled with the saving of the results in S3, the pipeline is also endowed with an email notification feature, which sends a summary of the outcome of each run. Once the step of connecting anomalies is done, a Python script collects the most important findings (the number of anomalies detected in each dataset, the number of nulls/duplicates, etc.) and writes an email message. The email gives a short summary in the body (e.g., Run 2025-08-25: 5 anomalies in insurance, 20 anomalies in banking, 2 null values found, 0 duplicates.) and includes the CSV log files as attachments. To send this email, we used an SMTP client in Python to an identified server; email service credentials were stored in the configuration of Dagster securely. The intended audience of the email may be a predetermined list of data engineers/analysts who would be monitoring the quality of data. Such an outbound reporting style guarantees instant access to possible problems: instead of a person manually checking S3 or the orchestrator UI, after each run, the results are sent to the inbox of the team. The rationale behind this design is to ensure that there is greater responsiveness in the event that an anomaly is detected something such as a fraud case or a serious data error, the concerned individuals will be informed immediately. Moreover, the availability of the logs in email offers an easy human-readable report to the managers or any other stakeholders who might not log in

to the S3 or Dagster systems. What we can do (in our case, mostly convenience) is to store email in the email, which is the reserve storage rather than in S3, and as an alternative, you might add messaging or dashboard alerts to the email.

Traceability and Auditing:

By combining the above mechanisms, the level of traceability in the implementation is high. Every pipeline run produces a result stored in Amazon S3 and sends out an email. The logs allow tracing alterations in data quality or the number of anomalies over time. As an illustration, when 100 anomalies are found in one run and only 10 in the next, we already have a clear record of that change and can investigate what might have caused it, for example, some improvements made to the data source. Likewise, in the event of a problem with a downstream application, one can consult these validation logs to find out whether anomalies were reported in that run.

These records are prepared in a regular and exhaustive way due to automation. The latter can be scaled in a production environment, such as the addition of monitoring to ensure that Dagster successfully completes its run every day or the visualisation of the CSV logs into a dashboard. However, the requirements of this project are met by the use of the orchestrated execution, S3 storage, and email reporting that would provide full and valid traceability of data validation results.

In conclusion, the automation and output delivery component of the implementation ties everything together into a cohesive system. The ETL pipeline not only changes and loads data, but also instantly checks and examines it against anomalies, and the results are automatically saved and notified. This makes sure that the diligent work of pulling out anomalies turns into a combination of action taken information made available to the correct individuals, and every ETL process comes with a verification process. This style of automated introduction of GPU-accelerated anomaly (detection) will provide a strong quality control interface to data engineering pipelines, which is precisely what this project aimed to do. The system architecture, model construction, and pipeline assembly, as presented in this chapter, all show how the research was put into practice to create a workable solution to anomaly detection with the use of a graphics card in ETL operations. All the components were implemented with a lot of care to make sure that the entire system is reliable, efficient and maintainable and that it can be deployed in an enterprise data environment.

5. Results. Analysis and Discussion:

5.1 Banking Dataset Results:

The banking dataset had 602, 883 records with 20 columns. With the reconstruction-error threshold of 0.02867, the autoencoder identified 6,029 anomalous records, which constitute about 1.0% of the data. Validated in 14.5 seconds on the NVIDIA GTX 1650 Ti. Such findings reveal that the model was able to identify all abnormal transactions, as it ignored many legitimate records. This aberration rate is typical of financial data sets, where outliers are very unusual but very significant:

```
{
  "device": "cuda",
  "gpu_available": true,
  "gpu_name": "NVIDIA GeForce GTX 1650 Ti",
  "epochs": 100,
  "total_records": 602883,
  "detected_anomalies": 6029,
  "threshold": 0.028670957755162475,
  "reconstruction_error": {
    "mean": 0.052879862324969946,
    "std": 0.6897706409437045,
    "min": 2.6644487981215356e-05,
    "max": 68.76490323589401
  },
  "training_loss_last_epoch": 0.0062113553285598755,
  "validation_time_sec": 14.51
}
```


Fig 5.1 Performance metrics for Banking Dataset

5.2 Insurance Dataset Results:

The Insurance Data consists of one million records with ten columns. Five thousand four hundred and forty-one anomalies, about 0.5% of the data, were identified at a far lower threshold of 0.000434. The validation was done within 12.3 seconds, which is a confirmation of the scalability of the acceleration of the use of GPUs. The lower percent variance anomaly to the banking implies that insurance data is more homogeneous, and few far-off outliers. Autoencoder was successful in capturing subtle inconsistencies, including non-normal policy or claim attributes:


```
{
  "device": "cuda",
  "gpu_available": true,
  "gpu_name": "NVIDIA GeForce GTX 1650 Ti",
  "epochs": 100,
  "total_records": 999830,
  "detected_anomalies": 5041,
  "threshold": 0.0004342909541837214,
  "reconstruction_error": {
    "mean": 0.0011621132282210577,
    "std": 0.02166296276419865,
    "min": 3.235114968392742e-16,
    "max": 0.7058053534432908
  },
  "training_loss_last_epoch": 3.8944101106608286e-05,
  "validation_time_sec": 12.31
}
```


Fig 5.2 Performance metrics for Insurance Dataset

5.3 Sales Dataset (Comparative Baseline):

The Sales data was used as a comparative benchmark. It had some minor data quality problems: 3 nulls, 3 duplicate records and 3 missing entries. In this dataset, it was the autoencoder that

did not detect any anomalies (as no anomalies related to autoencoder were injected). This implies that there is a high degree of consistency among the sales records and that there are no false positives of the model in the case of clean data.

Anomaly Summary

Log Type	Count
Duplicates Source	0
Duplicates Target	3
Nulls Source	0
Nulls Target	3
Missing Records	3
Autoencoder Anomalies	0

Fig 5.3 Anomaly Summary of Sales Dataset

5.4 GPU Acceleration and Performance:

Anomaly detection was facilitated by GPU acceleration in almost real-time. The Banking and Insurance data sets containing more than half a million and almost one million records were both verified within 12-15 seconds. CPU-only solutions would be expected to take several minutes, so daily anomaly inspections would be impractical. This proves that the detection of anomalies is a viable and scalable extension to ETL operations with the integration of GPUs.

5.5 Comparative Insights:

The rates of injected anomalies varied between the datasets: Banking (1.0%), Insurance (0.5%), Sales (0%). These differences indicate that dataset-specific thresholds must be calibrated. The findings also prove the significance of hybrid QA strategies. Checks based on rules were efficient to detect the nulls, duplicates, and missing records, whereas the autoencoder was able to detect all the hidden anomalies that could not be identified by predefined rules. With each other, they have holistic coverage of both anticipated and unanticipated data problems.

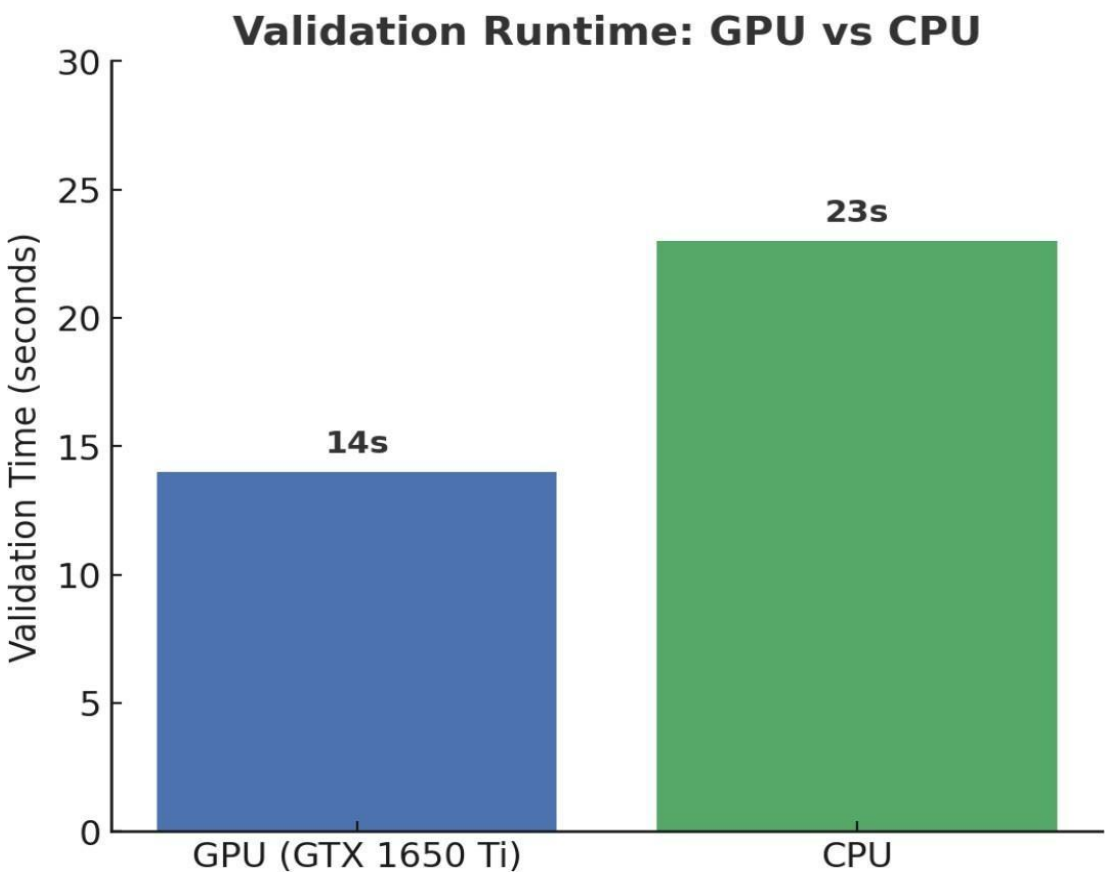

Fig 5.5 Comparison of Avg Time taken GPU vs CPU

6. Limitations

- Processing Data and Complex Logic in NLP: extend the system to detect anomalies using Natural Language Processing (NLP) to process business rules, transformation logic and unstructured metadata. This would not only allow the pipeline to verify structured anomalies, but also anomalies caused by complex ETL logic or even anomalies caused by the text.
- Model Retraining and Drift Control: Introduce a planned retraining or adaptive learning process to learn the effects of data drift and changing trends and retain the autoencoder useful over time.
- Dynamic Thresholding: Alternate manual tuned thresholds with adaptive, or statistically driven, thresholds (e.g., percentile-based or distribution-sensitive thresholds) to minimize dependence on manual calibration.
- Scalability and Deployment on Cloud: Deploy pipeline to cloud providers with access to more advanced GPUs as well as distributed processing frameworks to be able to process larger data volumes and enterprise-level workloads.

- Explainability and Monitoring Dashboards: Present interpretable outputs (e.g. which features led to the occurrence of an anomaly) and embed dashboards that show trends in anomalies, reconstruction error distributions, and data quality measures to stakeholders.

7. Conclusion and Future Work:

The thesis has addressed the research question by showing that it is possible to incorporate GPU-accelerated autoencoders in ETL pipelines to ensure better data quality validation and minimize errors in deployment time. The anomaly detection framework, which was organized in Dagster, reliably revealed the anomalies that were actively introduced to the datasets through testing and proved the validity and strength of the methodology. About 1.0 per cent of the records were found to be flagged in the Banking dataset, and about 0.5 per cent in the Insurance dataset, and the Sales dataset only produced minor errors that had been pre-introduced manually. These findings indicate that the system not only identifies the rare data anomalies with high precision but also offers a guarantee that even minor data problems would be automatically detected. Additionally, the availability of GPU acceleration allowed completing the process of anomaly detection in a matter of seconds, which confirmed that the tool is appropriate in terms of the real-time ETL processes. The combination of both traditional validation checks and machine-based aberration detection in a single pipeline validates that the research objectives were met to the maximum, offering a practical, scalable and automated process of proactive anomaly detection in data engineering.

References

Ahi, K., Wu, S., Sriram, S. and Fenger, G., 2025. GPU-Accelerated Feature Extraction for Real-Time Vision AI and LLM Systems Efficiency: Autonomous Image Segmentation, Unsupervised Clustering, and Smart Adaptation. [Conference paper]. URL:https://ieeexplore.ieee.org/abstract/document/11010527?casa_token=acfOteuaatsAAAAA:vHQl-4QzSf9WD0zMUnsi7EnFQznl1MYLx-FenzmHzRviDWnkdtUQLoDSrMOseuxtaFrYsbMVVc8eSg

Alak, B., Revanbahş, A., Argun, N. and Çalım, A.Ş., 2023. Anomaly Detection for ETL Packages Runtime: A Machine Learning Approach. In: 8th International Conference on Computer Science and Engineering (UBMK 2023). IEEE.
URL: https://ieeexplore.ieee.org/abstract/document/10286586

Ande, K. and Kumar, M., 2025. Cloud-Based ETL Pipelines with AI-Driven Anomaly Detection. International Journal of All Research Education & Scientific Methods, 13(2).
URL: https://www.ijaresm.com/cloud-based-etl-pipelines-with-ai-driven-anomaly-detection

Chakraborty, S., 2025. Beyond ETL: How AI Agents Are Building Self-Healing Data Pipelines. Journal of Computer Science and Technology Studies, 7(3), pp.741–756.
URL; http://al-kindipublishers.org/index.php/jcsts/article/view/9427/8081

Chaudhari, A.V. and Charate, P.A., 2025. Proactive Data Pipeline Maintenance via Machine Learning-Driven Anomaly Detection. International Journal of Scientific Research in Science and Technology, 12(2), pp.1041–1053.
URL:https://www.researchgate.net/profile/Akash-Vijayrao-Chaudhari-

4/publication/391151316_Proactive_Data_Pipeline_Maintenance_via_Machine_Learning-Driven_Anomaly_Detection/links/680bbd2edf0e3f544f497b3a/Proactive-Data-Pipeline-Maintenance-via-Machine-Learning-Driven-Anomaly-Detection.pdf

Hewa Nadungodage, C., Xia, Y. and Lee, J.J., 2016. GPU-Accelerated Outlier Detection for Continuous Data Streams. In: IEEE International Parallel & Distributed Processing Symposium (IPDPS 2016), pp.1133–1142.

URL:https://ieeexplore.ieee.org/abstract/document/7516110?casa_token=MsUZjCC9rPwAAAAA:BDbDi0B-p4-2rGS3vIq4-ACzADKHsI4EHqT-40UNnMQVOz2_9FlQMqihgNW_-UD0qNOb4XOSiCVDhw

John, B. and Jaiswal, A., 2025. Managing Data Quality and Consistency in Real-Time ETL for Streaming Applications: A Comparative Analysis of Modern ETL Frameworks. [Technical report].
URL:https://www.researchgate.net/profile/Beauden-John/publication/392589655_Managing_Data_Quality_and_Consistency_in_Real-Time_ETL_for_Streaming_Applications_A_Comparative_Analysis_of_Modern_ETL_Frameworks/links/68498127f1e8fa3b73d6009e/Managing-Data-Quality-and-Consistency-in-Real-Time-ETL-for-Streaming-Applications-A-Comparative-Analysis-of-Modern-ETL-Frameworks.pdf

Joshi, N., 2024. Optimizing Real-Time ETL Pipelines Using Machine Learning Techniques. SSRN Preprint, March 2024.
URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5054767

Li, Y., Liu, Z. and Wu, T., 2023. Incremental Autoencoder for Anomaly Detection in Evolving Data Streams with Concept Drift. Expert Systems with Applications, 224, p.119882.
URL: https://doi.org/10.1016/j.eswa.2023.119882
Litty, A., 2024. Real-Time Anomaly Detection in Industrial Processes: Leveraging GPU-Accelerated ML and AI Robotics for Predictive Maintenance. EasyChair Preprint, 14385.
URL: https://easychair.org/publications/preprint/HDtG/open

Maddali, R., 2024. Automating Data Quality Assurance Using Machine Learning in ETL Pipelines. [Preprint].
URL:https://www.researchgate.net/profile/Raghavender-Maddali-2/publication/390315594_Automating_Data_Quality_Assurance_Using_Machine_Learning_in_ETL_Pipelines/links/67e9799d03b8d7280e12efed/Automating-Data-Quality-Assurance-Using-Machine-Learning-in-ETL-Pipelines.pdf

Marupaka, D. and Rangineni, S., 2024. Machine Learning-Driven Predictive Data Quality Assessment in ETL Frameworks. International Journal of Computer Trends and Technology, 72(3), pp.53–60.
URL:https://www.researchgate.net/profile/Divya-Marupaka-2/publication/379568144_Machine_Learning-Driven_Predictive_Data_Quality_Assessment_in_ETL_Frameworks/links/660ef1b9b839e05a20bd6cc0/Machine-Learning-Driven-Predictive-Data-Quality-Assessment-in-ETL-Frameworks.pdf

Nickolls, J., Buck, I., Garland, M. and Skadron, K., 2008. Scalable Parallel Programming with CUDA. ACM Queue, 6(2), pp.40–53.

URL: https://research.nvidia.com/publication/2008-03_scalable-parallel-programming-cuda

Peter, H., 2023. Optimizing Data Pipelines for Real-Time Enterprise Analytics Using AI-

Driven ETL Tools. [White paper].
URL: https://www.researchgate.net/profile/Harry-Peter/publication/392499282_Optimizing_Data_Pipelines_for_Real-Time_Enterprise_Analytics_Using_AI-Driven_ETL_Tools/links/68448dc2df0e3f544f5d8b2a/Optimizing-Data-Pipelines-for-Real-Time-Enterprise-Analytics-Using-AI-Driven-ETL-Tools.pdf

Rahman, T., Alam, K. and Jahan, I., 2023. Data Quality Management and Performance Optimization for Enterprise-Scale ETL Pipelines in Modern Analytical Ecosystems. Journal of Data Science, Predictive Analytics, and Big Data Applications, 8(7), pp.1– 26.
URL: https://helexscience.com/index.php/JDSPABDA/article/view/2023-07-04

Zhang, K., 2024. Incorporating Deep Learning Model Development with an End-to- End Data Pipeline. IEEE Access, 12, pp.127522–127531.
URL:https://doi.org/10.1109/ACCESS.2024.3400483

Zhao, Y., Zeng, Y., Sun, G. and Wu, J., 2022. Scalable Outlier Detection for Large- Scale Data with Tensor Computation. IEEE Transactions on Knowledge and Data Engineering.
URL: https://doi.org/10.1109/TKDE.2022.3199422