Universiteit Leiden

Adaptive Data Profiling with ML-Based Quality Checks for ETL Pipeline Quality and Cost Optimization

Koorosh Komeili Zadeh (s3893995)

Supervisors:
Joost Visser & Jan van Rijn

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 8, 2026

Abstract

Background. Organisations processing large volumes of data on a continuous basis rely on Extract-Transform-Load (ETL) pipelines to meet non-functional requirements such as scalability, high throughput, and reliability. These automated systems extract data from external sources, transform and clean it, and load it into storage for analysis and business decisions. Contemporary ETL pipelines use rule-based quality checks to catch data errors, but such checks often fail to detect subtler problems, such as values that appear valid yet sit far outside the historical norm, before bad data reaches the systems that depend on it.

Aim. This thesis investigates whether machine learning can automatically learn what normal data looks like and reliably flag deviations, serving as a low-cost complement to rule-based checks at the point where data first enters the pipeline.

Method. We built and deployed a complete ETL pipeline processing hourly weather and electricity data, with artificial anomalies injected at known positions to measure detection accuracy and computational cost. We evaluated two automated approaches: classical machine learning model selection (AutoML) and Neural Hyperparameter Optimization (Neural HPO).

Results. AutoML reliably detected anomalies on columns with predictable distributions at low computational cost. Compared to Neural Hyperparameter Optimization, it ran 9.13× faster and achieved a 4.1× higher mean F2-score. Deep learning models added complexity without improving detection quality.

Conclusion. AutoML-based anomaly detection is a practical complement to rule-based checks in data pipelines. Its effectiveness and cost depend strongly on the nature of the incoming data, making it well-suited for columns with stable, predictable distributions and less effective for noisy or irregular ones.

Contents

1 Introduction

In data-driven organizations, data is collected from multiple distributed sources and must be processed, transformed, and stored before it can support dashboards, analytics, and business decisions [30]. Engineers manage this flow using automated systems called data pipelines, which define where data is collected, how it is processed, and where it is stored.

The most common type is the ETL pipeline (Extract, Transform, Load): data is pulled from a source system, processed and cleaned, and then written to a central data warehouse, a shared repository where analysts and business applications can query it.

The problem is that traditional quality checks inside ETL pipelines are rule-based. A data engineer writes checks like "temperature must not be null" or "pressure must be an integer type between 870 and 1085 hPa." These rules work well for obvious errors, but they only catch problems the engineer already knew about. Gradual sensor drift, unusual patterns, or values far outside the seasonal norm can pass through rule checks without triggering any alert [9, 29].

This thesis investigates whether machine learning can address this limitation. Instead of fixed rules, a model learns what normal data looks like from historical data and flags anything that deviates from that learned pattern, for example, a temperature reading that passes a valid-range check but sits far outside the recent seasonal norm. The goal is to make this automatic, requiring no manual rule-writing and no specialist knowledge to define what counts as normal for each column.

Figure 1 gives the intuition. Both checks run at ingestion, inside the Extract stage of the ETL pipeline, before any transformation or loading, so bad data is caught at the earliest possible point (the shift-left principle). The rule-based check accepts any value of the correct type and inside a fixed valid range, so a reading that is technically valid but far from the seasonal norm flows straight through. The learning-based check instead compares each value to a narrow normal range learned from recent history, which follows the seasonal shape of the data, and flags the values that fall outside it. The same anomalies that the rule-based check lets through are caught by the learning-based check.

1.1 Motivation and Practical Relevance

Rule-based quality checks exist, but subtle anomalies still pass through pipelines unnoticed. Quality checks are often applied after data has already been processed, so problems reach analytics and models before anyone spots them.

Recent industry work [10] shows that AI-driven ETL systems can automatically learn what normal data looks like and detect deviations before they reach downstream systems. Problems get caught earlier, before they propagate.

One main motivation is the principle of *shift-left* data quality: validating data as early as possible in the pipeline, ideally at the point where it first enters [11]. The same idea has long been part of software engineering practice, where defects are generally cheaper to address the earlier in the lifecycle they are caught. If a bad record enters the pipeline, every transformation that follows

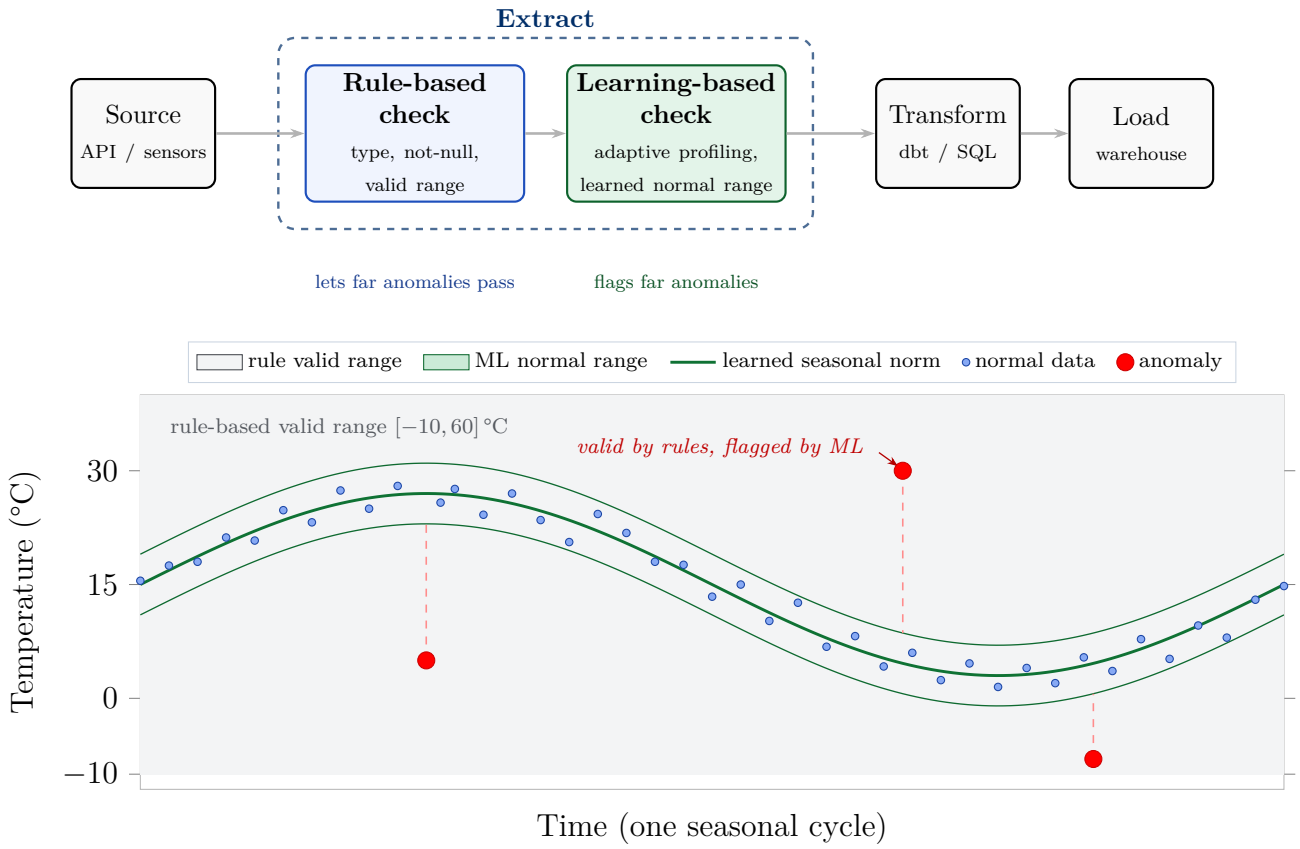

Figure 1: Where learning-based checks help. **Top:** both quality checks run inside the Extract (ingestion) stage of the ETL pipeline, before any Transform (dbt) or Load step, so anomalies are caught at the earliest point (the shift-left principle). Incoming data passes the rule-based check and then the learning-based (ML) check. **Bottom:** a zoom into the two checks on one seasonal column. The wide grey band is the fixed rule-based valid range, which accepts almost every value; the narrow green band is the normal range the model learns from recent history and tracks the season. The three red anomalies all sit inside the rule-based range (so rule checks pass them) but outside the learned normal range (so the ML check flags them).

inherits the defect. Worse, once a transformation aggregates or summarizes the data, the bad value is permanently baked into the stored result and can no longer be removed downstream, and by the time it reaches an aggregated report or a trained model it can be impossible to trace which source record caused the problem. Recent work argues that *when* validation happens, not only which method is used, determines both the final error rate and the cost of correction: Kothari et al. [24] model validation timing as a measurable design variable and show that catching errors before they propagate is substantially cheaper than catching them after downstream stages have consumed them, while Karlas et al. [20] show that data errors compound as they move through the stages of a machine learning pipeline, so an error's impact depends strongly on where it is caught. This is consistent with the broader data-centric view that data quality should be treated as a first-class concern across the pipeline rather than patched downstream [37].

The other motivation is to reduce the cost of computation for high-quality data delivery by applying outlier detection once, rather than repeatedly by different downstream users. The benefit of acting early is not only lower cost: the ActiveClean project [25] shows that dirty data systematically biases downstream model parameters, and that cleaning earlier in the pipeline gives up to 2.5 times better model accuracy for the same cleaning budget, while HoloClean [32] shows the broader trend away from pure rule-based quality toward learning-based approaches. In practice, many data teams skip anomaly detection entirely due to the engineering effort and domain expertise required [35].

1.2 Research Questions

The main research question is:

How effective and practical is learning-based anomaly detection as an addition to rule-based quality checks at the point where data first enters an ETL pipeline?

This is explored through three sub-questions:

- In terms of *effectiveness*:

  How accurately can learning-based anomaly detection identify outlier data quality issues compared to rule-based checks?

- In terms of *practicality*:

  What is the computational overhead of integrating learning-based anomaly detection at the point where data enters the pipeline, and how does it scale with data size?

- In terms of *learning model*:

  Does classical machine learning outperform deep learning architecture search in the trade-off between detection accuracy and computational cost for automated anomaly detection?

Throughout this thesis, the term **adaptive data profiling** refers to a quality check layer that trains a separate anomaly detection model on each individual data column. It is adaptive in three ways: the algorithm chosen varies based on the column's own data characteristics; each data source trains its own model; and models are retrained on a schedule as data patterns change over time.

2 Background

Several concepts underpin the rest of the thesis: how data pipelines work in practice, what data quality means in ETL pipelines, the case for validating data early at the point of ingestion (shift-left), unsupervised anomaly detection, and automated machine learning with hyperparameter optimization. Each is covered in turn below.

2.1 Data Pipelines in Practice

A data pipeline is the system that handles this automatically, governing how data is collected, processed, and delivered to the systems that depend on it [30]. There are two main types. Real-time (streaming) pipelines process data as it arrives, suitable for fraud detection or live dashboards. ETL pipelines process data in batches on a schedule, suitable for analytics, reporting, and training machine learning models [36].

2.1.1 Business Use of ETL Pipelines

ETL pipelines are central to how data-driven companies operate. A travel platform such as Skyscanner might collect different types of data that affect flight pricing, for example:

- Data from booking platforms such as Booking.com to identify demand based on hotel reservations.
- Data from airlines such as KLM and Lufthansa to adjust pricing and identify travel seasons.
- Data from weather services to predict when people are more likely to travel to destinations with optimal weather.

This data can be used to adjust prices based on demand, helping the company remain profitable, stay competitive in the market, and offer customer-friendly prices. This process can be automated through data pipelines.

In each case, the pipeline extracts raw data from sources the company does not control, transforms it into a consistent and usable format, and loads it into a warehouse that supports products and business decisions. The quality of everything downstream depends directly on the quality of the data that flows through the pipeline.

2.1.2 ETL: Extract, Transform, Load

Extract-Transform-Load is the long-established process for populating a data warehouse from operational sources, and it remains the dominant integration pattern behind analytics and decision-support systems [36, 8].

Extract pulls data from a source: a REST API, a database like PostgreSQL, data lakes, or a stream. In this project, data is extracted from the Open-Meteo weather API [40] to simulate a modern ETL system.

Transform reshapes and cleans the data: fixing types, removing duplicates, joining tables and running quality checks. In this pipeline, transformation is handled by dbt, which runs SQL on a DuckDB warehouse.

Load writes the processed data to its destination. Here the data is stored as Parquet files on Amazon S3, and transformed results are materialized as tables in DuckDB/MotherDuck.

2.2 Data Quality in ETL Pipelines

Data quality problems in ETL pipelines are common. Sources change their schemas, sensors send wrong readings, APIs return corrupted records, and values drift outside expected ranges over time. When these problems go undetected, they propagate through the pipeline into every downstream system.

Chu et al. [9] describe two types of data quality problems. Qualitative problems are violations of integrity constraints such as wrong types, missing values, or uniqueness failures. Quantitative problems are statistical anomalies that are technically valid values but unusual compared to what the data normally looks like. Rule-based checks address the first type well, but cannot address the second.

Sculley et al. [34] showed that hidden technical debt in ML systems often comes from the data pipeline. When data quality is not monitored at the source, every downstream system inherits the problem.

2.3 The Case for Shift-Left Data Quality

"Shift-left" means validating data as early as possible in the pipeline, ideally at the point of ingestion. The principle has long been part of software engineering practice, where defects are generally cheaper to address the earlier in the lifecycle they are caught [11]. The same logic transfers to data pipelines: a defect that enters at ingestion passes through every transformation after that, and by the time it reaches an aggregated report, it may be impossible to identify which source record caused the problem.

This argument becomes decisive when the Transform step is irreversible. Many transformations aggregate or summarize records, for example by computing a daily average, a running total, or a downsampled rollup. Once a bad value has been folded into such an aggregate, it is permanently incorporated into the stored data and can no longer be isolated or removed downstream: the only remedies are to discard the entire aggregate or to recompute it from the raw records, assuming those raw records are even still available. Catching the error before this step is therefore not merely cheaper, it is the last point at which the error can be removed at all.

Crucially, recent work argues that the timing of validation, not only the method used, determines both the final error rate and the cost of correction. Kothari et al. [24] formalise validation timing as a measurable design variable through an error-propagation model and show, drawing on the software-engineering evidence, that catching errors before they propagate is substantially cheaper than catching them after downstream stages have already consumed them. Karlas et al. [20] similarly emphasise that data errors compound as they move through the stages of a modern ML pipeline, so the impact of an error depends strongly on where in the pipeline it is caught. This is consistent with the broader data-centric view that data quality must be treated as a first-class concern across the pipeline rather than patched downstream [37].

Detecting the problem at the point of ingestion stops this propagation immediately. The flagged record can be reported upstream for validation, excluded from aggregations, or filtered before reaching downstream consumers. Doing it once at ingestion also avoids each downstream consumer having to independently detect and handle the same defect. This motivates placing data-quality checks as early as possible in the pipeline, including at ingestion when the relevant error signals are available. Industry discussions of shift-left data quality make this argument explicitly, while recent research on annotation QA and ML-pipeline errors supports the broader point that validation timing and error propagation matter [11, 24, 20].

2.4 Unsupervised Anomaly Detection

Unsupervised anomaly detection identifies unusual data points without labeled training data [7]. The idea is to train a model on historical data to learn what normal looks like, then score new records based on how well they fit that learned distribution. Records that deviate significantly get a high anomaly score.

We selected PyOD [38], an open-source Python library that implements a wide range of outlier-detection algorithms behind a single API, over the broader AutoML libraries H2O AutoML, AutoGluon, and PyCaret for its targeted unsupervised detection models and simple integration into the Adaptive Profiling library.

Four algorithms from PyOD were selected by Optuna, a hyperparameter optimization framework that automatically searches for the best-performing model and configuration (detailed in Section 2.5), across all trained models in this project:

Local Outlier Factor (LOF) [5] measures the local density of a point compared to its neighbors. Points in less crowded areas than their neighbors get high anomaly scores. It is effective for contextual anomalies and scales super-linearly with data size.

ECOD (Empirical Cumulative Distribution-based Outlier Detection) [27] finds unusual values by checking how far each value is from the normal range of a feature. It is fast, easy to explain, and requires minimal hyperparameter configuration. This makes it an effective baseline for columns where most values are close together.

Histogram-based Outlier Score (HBOS) [14] estimates the density of each feature independently using histograms. It is fast and effective for low-dimensional data where features are

approximately independent.

COPOD (Copula-based Outlier Detection) [26] finds unusual rows by looking at how feature values behave together. It gives higher scores to rows with rare combinations of values. It is fast and requires minimal hyperparameter configuration.

Campos et al. [6] show that no single algorithm dominates across all datasets. The right choice depends on the data, which is why automated model selection is important.

2.5 AutoML and Hyperparameter Optimization

AutoML refers to automating the selection of algorithms and their hyperparameters so that building a model requires little manual tuning or specialist intervention [18]. Instead of manually choosing among Isolation Forest [28], LOF, ECOD, HBOS, and COPOD, and tuning each configuration by hand, an optimization framework searches the space of possible configurations automatically. Such automated search has been shown to find good configurations far more efficiently than manual or grid search [3].

We use Optuna [1] for this purpose, choosing it over Scikit-Optimize (skopt), Hyperopt, and Ray Tune because it is simple to use, compatible with PyOD, and well suited for searching across model types and key hyperparameters. Optuna's default sampler is the Tree-structured Parzen Estimator [2], a sequential model-based method that concentrates trials in the promising regions of the search space. Optuna is also used for the neural comparison experiment (Section 6.3), where the search space switches from PyOD classical detectors to PyTorch neural models.

3 Related Work

Prior work relevant to this thesis spans four areas: machine-learning approaches to data quality in ETL pipelines, the limitations of existing rule-based quality tools, AutoML for anomaly detection, and the comparison between classical and neural detectors for tabular data.

3.1 ML-Based Quality in ETL Pipelines

Several recent studies have explored machine learning for ETL data quality. Maddali [29] proposes a general ML-based framework for automated data quality assurance in ETL pipelines, but does not focus on AutoML, automated model selection, or per-column model tuning. Gopa [16] investigates the integration of machine learning models into ETL pipelines for anomaly detection, data quality assurance, and automated data governance, including data classification, audit logging, and compliance enforcement.

In contrast, this thesis integrates AutoML on historical data, automating model tuning and retraining as data patterns change. This reduces the need for continuous engineering effort and lowers the cost of maintaining quality checks. In many teams, data engineers may not have the required

machine learning domain knowledge, while close collaboration with data scientists can be difficult and costly. As a result, companies often rely on simple rule-based checks and can miss important data quality issues.

3.2 Limitations of Similar Tools

The SIGMOD 2016 paper by Chu et al. [9] draws a clear distinction between qualitative and quantitative data quality problems and shows that modern systems need both approaches. Rule-based systems such as Deequ [33] are effective at checking rules that users define, such as missing values, allowed ranges, duplicates, and data types. Deequ also supports anomaly detection, but mainly on data-quality metrics over time, such as changes in row count or completeness. It does not mainly focus on finding unusual values inside individual records. Therefore, values that follow the written rules but are still statistically unusual may pass through unless extra anomaly detection is added.

3.3 AutoML for Anomaly Detection

Koh et al. [21] demonstrate AutoML applied end-to-end in an industrial domain, showing that an automated pipeline generalizes across different dataset configurations without manual tuning. Applying AutoML to anomaly detection is harder than to supervised learning: without labels there is no hold-out validation signal against which to compare candidate models, so model and hyperparameter selection cannot rely on a conventional objective. Zhao et al. [39] tackle this unsupervised outlier model selection problem with MetaOD, a meta-learning approach that picks a detector and its hyperparameters for a new dataset by transferring performance knowledge from a large collection of historical benchmark datasets, and report that principled selection significantly outperforms always using a single popular detector or an ensemble of many.

This thesis takes a complementary and more lightweight route. Rather than transferring meta-knowledge across datasets, it runs a per-column Optuna search on each source's own recent history and uses the injected anomalies purely for evaluation, never for training, so the search itself remains unsupervised. Emmott et al. [13] provide a methodology for constructing ground-truth anomaly detection benchmarks with controlled anomaly properties, which motivates the controlled-anomaly evaluation approach used to measure that search in this project.

3.4 Classical ML vs. Neural Networks for Tabular Anomaly Detection

For high-dimensional inputs like images or audio, neural autoencoders are a natural fit for anomaly detection. However, this logic does not carry over as cleanly to low-dimensional tabular data, where a single-column input gives the autoencoder little structure to compress. A further, well-documented difficulty is that autoencoders often *generalize* well enough to reconstruct anomalies almost as accurately as normal points, so a large reconstruction error is not a reliable indicator of an anomaly. Gong et al. [15] observe exactly this failure, that the reconstruction assumption

"does not always hold in practice", and propose memory-augmented autoencoders specifically to counteract it.

In the most comprehensive tabular anomaly detection benchmark to date, Han et al. [17] evaluate 30 detectors over 57 datasets and report that, in the unsupervised setting, no algorithm is statistically better than the rest and that several deep-learning detectors perform worse than classical shallow methods, which are also easier to train and tune. Campos et al. [6] reach a consistent conclusion in a focused study of twelve nearest-neighbor-based detectors: performance depends strongly on the dataset and on parameter settings, with no universal winner, which motivates automated per-column model selection. The PyOD library [38] exposes both classical detectors such as LOF and neural ones such as autoencoders behind a single API, which makes a like-for-like comparison between the two model families straightforward.

Speed is also a practical concern. Training a neural network for tens to hundreds of epochs per Optuna trial is much more expensive than fitting a tree or density model. Section 6.3 reports a direct comparison: classical AutoML achieves 4.1× higher mean F2-score at 9.13× lower search cost per column.

4 System Design

This work follows a build-and-evaluate approach: this section describes the system that was built, and Section 5 describes how it was evaluated. A complete ETL pipeline was designed and deployed on AWS, using Apache Airflow for orchestration, dbt for SQL transformations, and DuckDB as a data warehouse. A custom Python library called the Adaptive Profiler was developed and integrated at the ingestion stage. It adds two quality check layers on each incoming batch: a rule-based layer for data contract checks, and an ML-based layer that trains an anomaly detection model per column using PyOD and Optuna. Both layers are configured through a single YAML file.

4.1 System Architecture

The system adds ML-based anomaly detection to an ETL pipeline using open-source tools.[1]

- AWS EC2 (a Linux server) as the compute host, running the pipeline inside Docker containers
- Apache Airflow for pipeline orchestration and scheduling
- External weather APIs as the data source for the Extract stage
- dbt for data transformation and schema metadata
- Amazon S3 as the data lake where processed data is loaded

[1]Full source code, system design, and experiment scripts are available at https://github.com/kooroshkz/adaptive-data-profiling-etl [22].

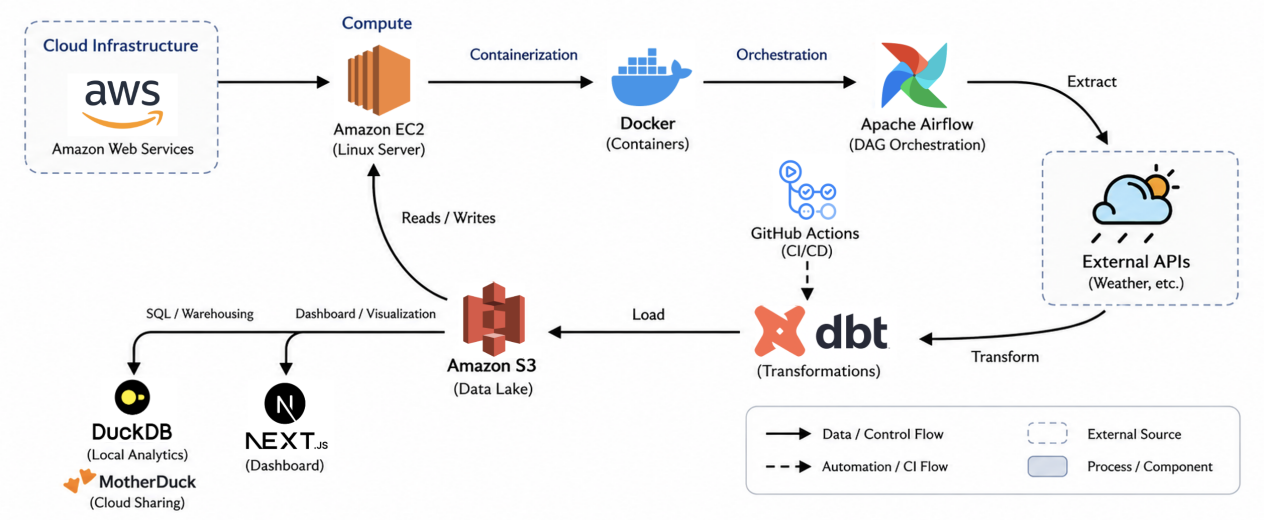

Figure 2: System architecture and tool integration. Apache Airflow runs in Docker on an Amazon EC2 server (on AWS) and orchestrates the Extract (external weather APIs), Transform (dbt), and Load (Amazon S3 data lake) stages. From S3, DuckDB/MotherDuck serves SQL analytics and a Next.js dashboard provides visualization; GitHub Actions handles CI/CD. Solid arrows are data and control flow, dashed arrows are automation and CI flow, and dashed boxes mark external sources.

- DuckDB/MotherDuck as a SQL warehouse for local analytics and cloud sharing
- A Next.js dashboard for visualizing the results
- GitHub Actions for continuous integration and deployment
- A custom Python library for ML-based anomaly detection using PyOD and Optuna [23]

Figure 2 shows how these tools fit together. The pipeline runs on an Amazon EC2 Linux server, where Apache Airflow executes inside Docker containers. Airflow orchestrates the three ETL stages: it extracts data from external weather APIs (Extract), dbt transforms it (Transform), and the results are loaded into an Amazon S3 data lake (Load). From S3, DuckDB/MotherDuck provides SQL analytics and a Next.js dashboard provides visualization, while EC2 reads from and writes to S3. GitHub Actions automates continuous integration and deployment. In the figure, solid arrows show data and control flow, dashed arrows show automation and CI flow, and dashed boxes mark external sources. The AutoML anomaly detection that this thesis adds is integrated into the Airflow workflow and is shown in detail in the pipeline DAG of Figure 4.

4.2 Extending the Pipeline

Figure 3 shows the baseline Airflow DAG. Five ingestion tasks run in parallel, one per city, and when they finish, dbt is triggered. Figure 4 shows the extended DAG with AutoML detection tasks (in green) inserted after each ingestion step.

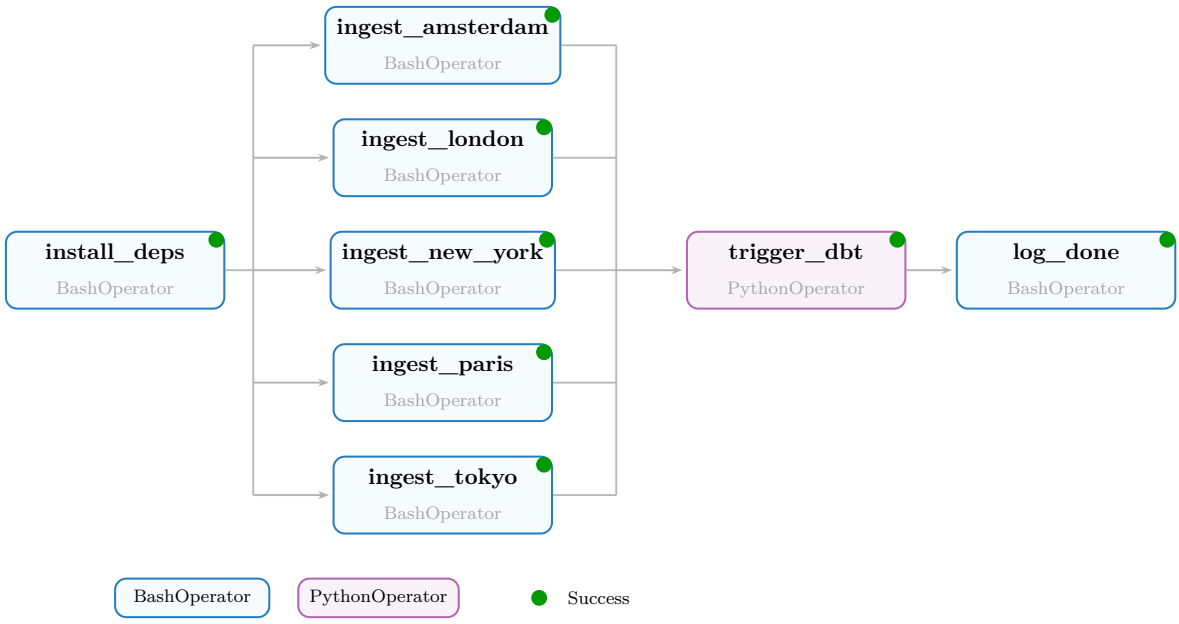

Figure 3: Baseline Airflow DAG for weather ingestion. Five ingestion tasks run in parallel, one per city. Once all complete, `trigger_dbt` runs SQL transformations and `log_done` records the result.

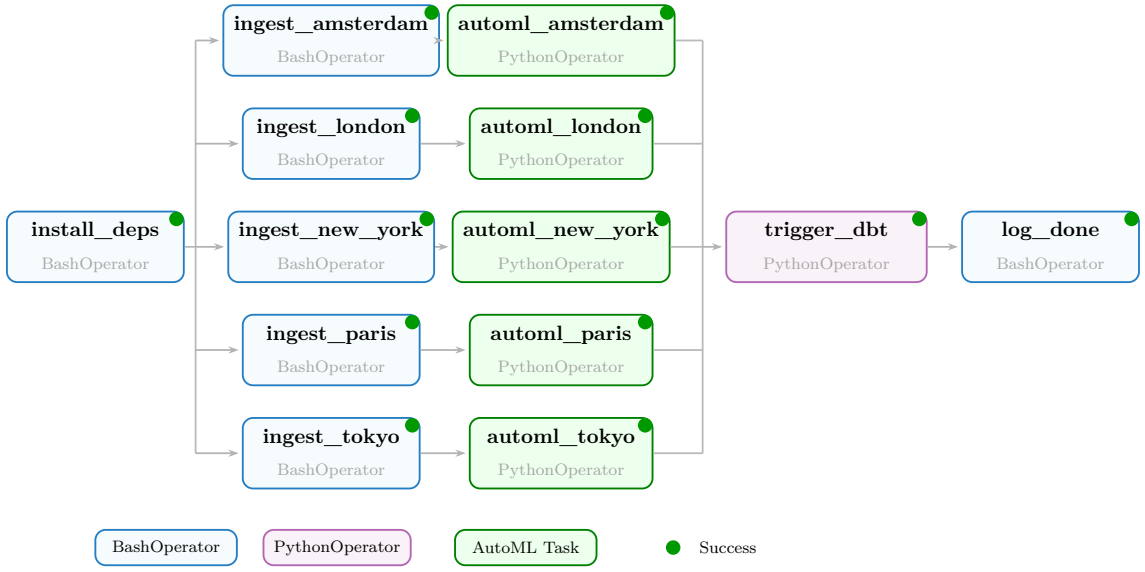

Figure 4: Extended Airflow DAG with AutoML detection tasks (green) inserted after each city's ingestion. Each `automl_*` task loads the city's pre-trained anomaly detection model, scores the newly ingested batch, and writes anomaly flags back to the warehouse. All original tasks are unchanged.

4.3 The Adaptive Profiler Library

Quality checking in this pipeline has two layers.[2]

The first layer is **rule-based**. Every column has an expected type, a valid range, and a not-null constraint. These rules form a data contract: a written agreement about what the data should look like. dbt enforces a similar contract at the transformation layer through schema tests on the warehouse models. Both rule sets run on every incoming batch and flag any record that violates a constraint.

Rule-based checks are fast and transparent, but limited. A rule like `range: [-10, 60]` for temperature in Celsius will not catch a sensor reading of 0°C when every other reading in the past three days was above 25°C.

The second layer is **ML-based**. For any column marked `automl: true` in the configuration, a statistical model is trained on recent historical data. The model learns what normal looks like for that column and flags values that fall outside its learned distribution, even when they pass all rule-based checks. This is what the library calls adaptive profiling.

In **training mode**, Optuna searches for the best model and hyperparameters for each configured column. Trained models are saved to S3 under a path that includes the column name and city, so Amsterdam gets its own model trained on Amsterdam's weather patterns.

In **scoring mode**, the profiler loads the saved model for each column and scores new incoming rows. The output is a long-format DataFrame with one row per (timestamp, column) containing the anomaly score, a binary flag, and any rule-based violations. Failures are reported in the output rather than raised as exceptions so the pipeline keeps running.

Both layers are declared in a single YAML configuration file. An example is shown below.

[2]The Adaptive Profiler is published as a standalone Python package and can be installed via `pip install adaptive-profiler`. The package is available at https://pypi.org/project/adaptive-profiler/ and the source at https://github.com/kooroshkz/adaptive-profiler [23].

```yaml
# Where trained models are saved
model_store:
  backend: s3
  bucket: weather-etl-bucket

# Optuna search settings
training:
  n_trials: 25
  window_size: 5000      # Train on the most recent 5 000 rows only

# Data model configuration
columns:
  temperature_2m:
    type: float
    automl: true         # AutoML anomaly detection enabled
    checks:
      not_null: true
      range: [-10, 60]

  surface_pressure:
    type: float
    automl: true
    model: LOF           # Pin algorithm & skips Optuna search
    hyperparameters:
      n_neighbors: 20
    flag_threshold: 0.42  # Tune sensitivity and Confidence Threshold
    checks:
      not_null: true
      range: [870, 1085]

  precipitation:
    type: float
    automl: false         # Rule checks only for this column
    checks:
      not_null: true
      range: [0, 300]
```

Listing 1: Example `profiling_schema.yml`. Each column declares a type and rule-based checks (data contract layer) and optionally enables an AutoML model (`automl: true`). Optional per-column keys (`model`, `hyperparameters`, `flag_threshold`) and a global `window_size` are also shown.

5 Experimental Setup

To evaluate the system, hourly weather data from five cities was ingested from the Open-Meteo API with controlled anomalies injected as ground-truth labels. Three experiments were run, one per research sub-question: detection effectiveness, computational overhead and scaling, and a comparison between classical AutoML and neural detector search.

5.1 Dataset

The experiments use weather data from the Open-Meteo API [40], benchmarked across five different cities: Amsterdam, New York, London, Paris, and Tokyo. Each hourly record contains temperature, apparent temperature, precipitation, surface pressure, soil temperature, and soil moisture. The data is stored in Parquet format on Amazon S3. Each city dataset contains 20,352 hourly records.

We selected weather data for this project because it is widely used by businesses to improve planning, prediction, and service optimization. For example:

- **TomTom/Google Maps:** Weather conditions can be compared with historical traffic patterns to improve traffic estimation systems.
- **UPS/FedEx/Amazon:** Delivery delays can be compared with weather conditions to improve route planning, ETA prediction, and driver scheduling.
- **Uber/Bolt:** Weather effects on ride demand, cancellations, pickup delays, and surge pricing can be analyzed.
- Many other businesses ingest weather data to improve pricing, operations, and service optimization.

A second dataset is introduced in Section 6.1.6 to assess whether the approach generalizes beyond weather data. We sourced GB national electricity demand from the Elexon BMRS Insights API [12], a publicly accessible data service for Great Britain's electricity market.

5.2 Experiment Design

The three research sub-questions map to three experiments.

Experiment 1: Detection effectiveness. To answer how accurately ML-based detection finds data quality issues compared to rule-based checks, we inject controlled anomalies into the data and compare the system's detections against the known injection labels. This directly measures detection quality. Section 6.1.6 repeats the same procedure on electricity demand data to assess cross-domain generalization.

Experiment 2: Practical overhead. To answer the practical impact question, a built-in projection tool in the profiling library benchmarks training time across different data sizes, column counts, and trial budgets, and fits an empirical scaling law to the results.

Experiment 3: Classical vs. neural detectors. To answer the learning-model question, the chosen classical AutoML approach is compared against an automated neural (autoencoder) detector search on the same data, preprocessing, and unsupervised objective, measuring both detection quality and search cost.

5.2.1 Experiment 1: Anomaly Injection and Detection

We shifted one column's value by a controlled amount in a randomly selected subset of records (approximately 0.36%, around 74 records per city). The injected records serve as ground-truth labels.

Model performance is measured with precision, recall, F1-score, and F2-score. F2 weights recall more heavily than precision. Missing a real anomaly is considered worse than a false alarm.

5.2.2 Experiment 2: Scaling and Cost Projection

The projection tool runs the full training pipeline across approximately 90 benchmark configurations, varying data size, column count, and Optuna trial budget. Results are fitted to an empirical power-law scaling model:

$$T(n, m, k) = \alpha \cdot n^{\beta} \cdot m^{\delta} \cdot k^{\gamma}$$

where n is the number of rows, m the number of columns, and k the number of Optuna trials. Once fitted, the formula predicts training time for configurations within and near the benchmarked range. Run it on a small sample, get an estimated time for the full dataset, and decide whether the overhead is acceptable before setting up production.

The library's built-in benchmarking feature produces these parameters, so users can project training time to larger datasets or apply the tool to other metrics and pipelines.

5.2.3 Experiment 3: AutoML vs. Neural Detector Search

To validate the choice of classical PyOD algorithms over neural network detectors, we ran both approaches on the same five-city dataset, same preprocessing, and same unsupervised objective. Neither sees anomaly labels at any point. Both are driven by the same Optuna search; the only difference is the search space.

- **AutoML:** Optuna selects among Isolation Forest, LOF, ECOD, HBOS, and COPOD with 25 trials per column, the same classical PyOD detectors used in the main pipeline.

- **Neural Hyperparameter Optimization (Neural HPO):** Optuna searches over two PyTorch autoencoder detectors, the Autoencoder (AE) and Variational Autoencoder (VAE), with 25 trials per column to match AutoML.

We implemented the neural detectors in PyTorch rather than a higher-level wrapper such as AutoKeras or Auto-PyTorch because these wrappers do not support the Autoencoder (AE) and Variational Autoencoder (VAE) architectures. We restricted the search space to autoencoder-style models, the standard neural approach to unsupervised anomaly detection [31]: an autoencoder is trained only to reconstruct normal-looking data and flags the points it reconstructs poorly, so, like the PyOD detectors, it needs no anomaly labels. A One-Class SVM was deliberately left out of the search space because it is not a neural network; keeping the search purely neural ensures a like-for-like comparison against the classical baseline. The two variants differ in how they model the normal distribution.

Autoencoder (AE) compresses each input through a narrow hidden bottleneck and reconstructs it, minimizing reconstruction error on historical data. It learns to rebuild normal values accurately, so a record it reconstructs poorly receives a high anomaly score. It is the most widely used neural baseline for anomaly detection.

Variational Autoencoder (VAE) extends the autoencoder with a probabilistic latent space and is trained on the evidence lower bound, a reconstruction term plus a KL-divergence regularizer. The latent regularization is intended to give a smoother model of the normal distribution and a better-calibrated reconstruction score, at the cost of a harder optimization.

For both models Optuna searches the same architecture space: hidden layer count, layer width, activation function, learning rate, epoch count, and batch size; Scikit-learn handles the preprocessing. The detection and timing results of this comparison are reported in Section 6.3.

6 Results

The pipeline targets obvious data quality failures, not every subtle statistical deviation. The focus is on anomalies from upstream system failures: a sensor stuck at zero, a data feed that goes silent, or a value so far outside historical range that no operator would accept it. High sensitivity is avoided on purpose. Noisy signals like precipitation and temperature would produce too many false alarms at low thresholds.

The injected anomalies in the experiments below are moderate shifts, not extreme spikes, which makes the evaluation harder than a typical real-world failure scenario.

6.1 Experiment 1: Detection Effectiveness

This experiment measures how accurately ML-based detection finds injected anomalies compared to rule-based checks. We first describe the injected anomalies, then report detection results for

Amsterdam, the algorithm Optuna selected across all cities, the multi-city comparison, the effect of shift magnitude, and finally a generalization test on electricity demand data.

6.1.1 Injected Anomalies

Figure 5 shows a close-up of Amsterdam temperature, where pink markers indicate synthetic anomalies injected into the data. Blue points represent normal readings. Out of 20,352 hourly records, 74 anomalies were injected at a rate of 0.36%, with varying positive or negative shifts distributed randomly to measure model accuracy across different shift percentages.

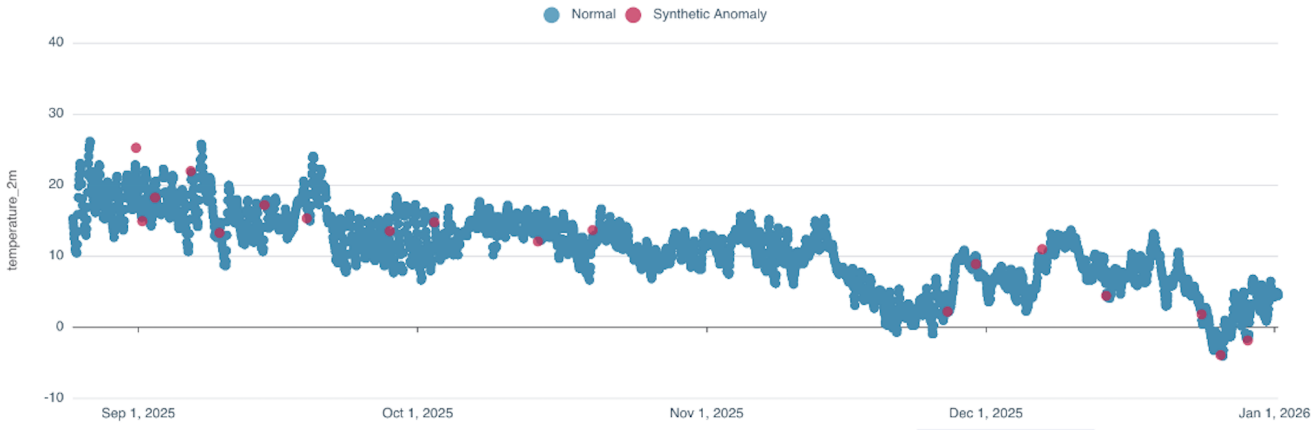

Figure 5: A snapshot of Amsterdam temperature data from September 2025 to January 2026. A total of 74 anomalies were injected, corresponding to a 0.36% anomaly rate.

6.1.2 Amsterdam Detection Results

For comparison, the rule-based temperature check defined a valid range of $[-10, 60]\,°\text{C}$. All 74 injected anomalies fell within this range, whereas the ML-based quality checks detected 53 of the 74 anomalies, corresponding to a recall of 71.6%.

Figure 6 shows precision, recall, and F1 for all six univariate columns in Amsterdam. Each bar represents the metric for the best model selected by Optuna for that column.

Recall matters more here: a missed anomaly silently corrupts downstream data, while a false alarm costs only a manual review.

6.1.3 Model Selection Across All Cities

Figure 7 shows which algorithm Optuna selected across all 30 trained models (5 cities $\times$ 6 columns). LOF was chosen in 80% of cases and ECOD was selected for surface pressure in 4 out of 5 cities (Paris used HBOS instead).

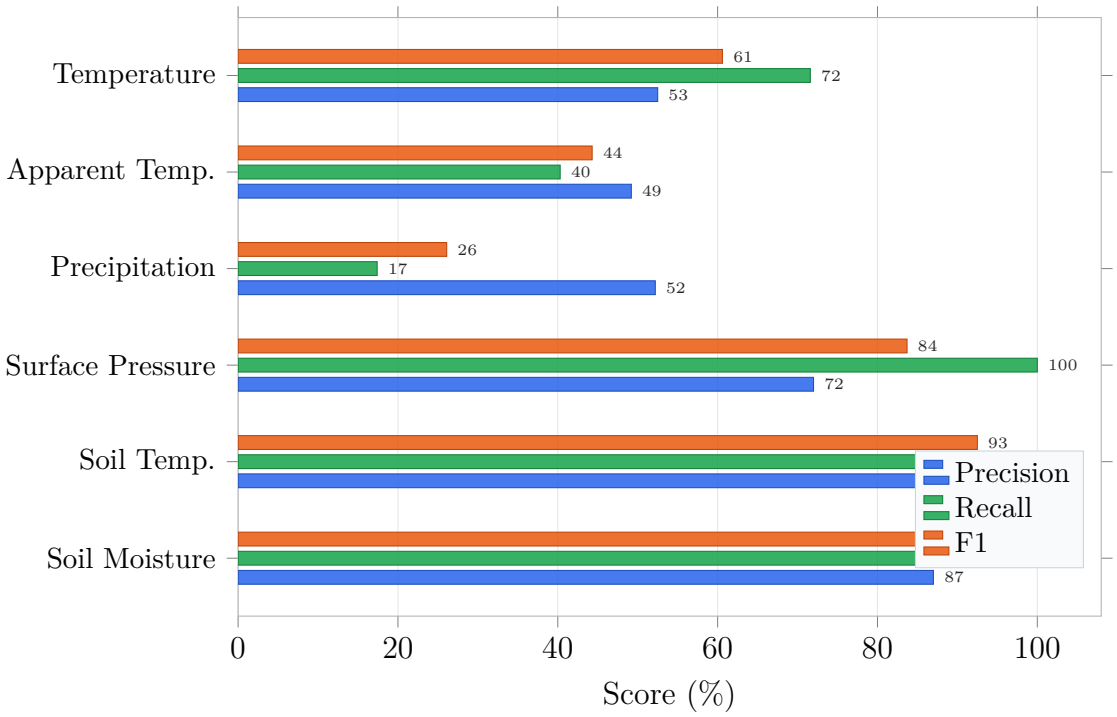

Figure 6: Precision, recall, and F1 scores for Amsterdam detection across all six columns. Precision measures the share of raised flags that are real anomalies, recall measures the share of real anomalies that are detected, and F1 summarizes the balance between precision and recall.

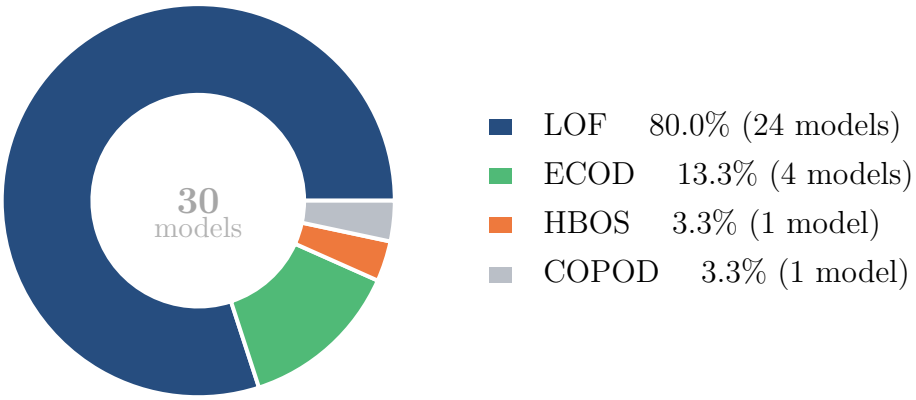

Figure 7: Algorithm selected by Optuna across all 30 trained models (5 cities × 6 columns). LOF was selected in 80% of cases. ECOD was selected for the surface pressure column in 4 out of 5 cities (Paris used HBOS).

6.1.4 Multi-City Comparison

Figure 8 highlights three representative columns across all five cities: surface pressure (best performing), soil moisture (consistently high), and temperature (most variable). Figure 9 extends this to all six columns, ordered by average F2 from lowest to highest.

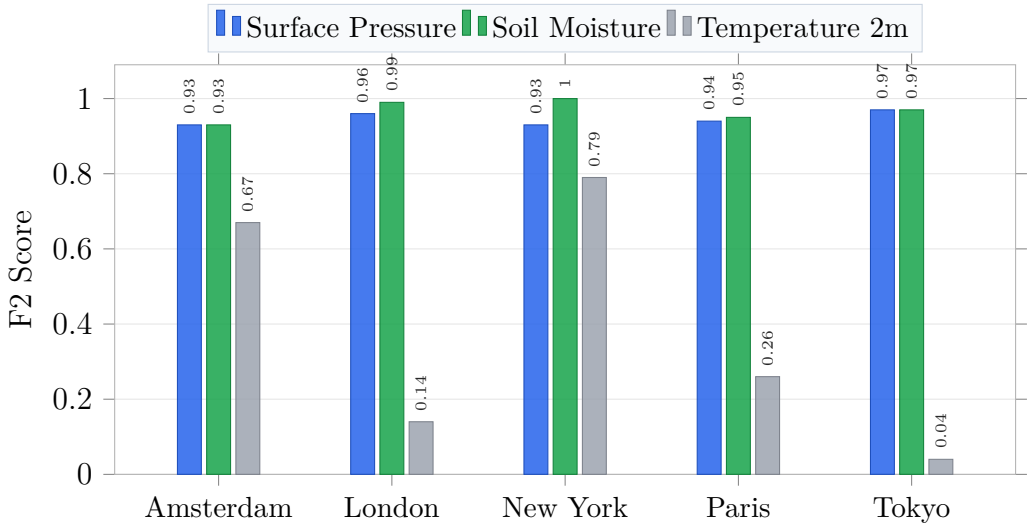

Figure 8: F2 scores across all five cities for three representative columns. Surface pressure and soil moisture are consistently high across all cities. Temperature detection varies significantly, suggesting that local climate variability and the column's noise level strongly influence difficulty.

6.1.5 Shift Magnitude vs. Detection Rate

Figure 10 shows how detection rate changes with the size of the injected shift, measured as a percentage of the column's natural value range.

6.1.6 Generalization to Electricity Data

To verify that the approach generalizes beyond weather data, we applied the Adaptive Profiler to GB national electricity demand records from the Elexon BMRS Insights API [12]. The dataset contains two columns: `initialDemandOutturn` (INDO) and `initialTransmissionSystemDemandOutturn` (ITSDO). We injected 182 synthetic anomalies at a rate of approximately 2.5%, with shifts of 50–100% of the original value distributed randomly in sign and target column.

This experiment was conducted by running the profiler twice on the same data. The untuned run is equivalent to `automl: true` with default hyperparameters in `profiling_schema.yml` selected by Optuna. The tuned run has customized parameters found by the script to fit better based on data characteristics and ground truth, simulating a data engineer setup for increasing detector accuracy. Table 1 shows the impact of tuning.

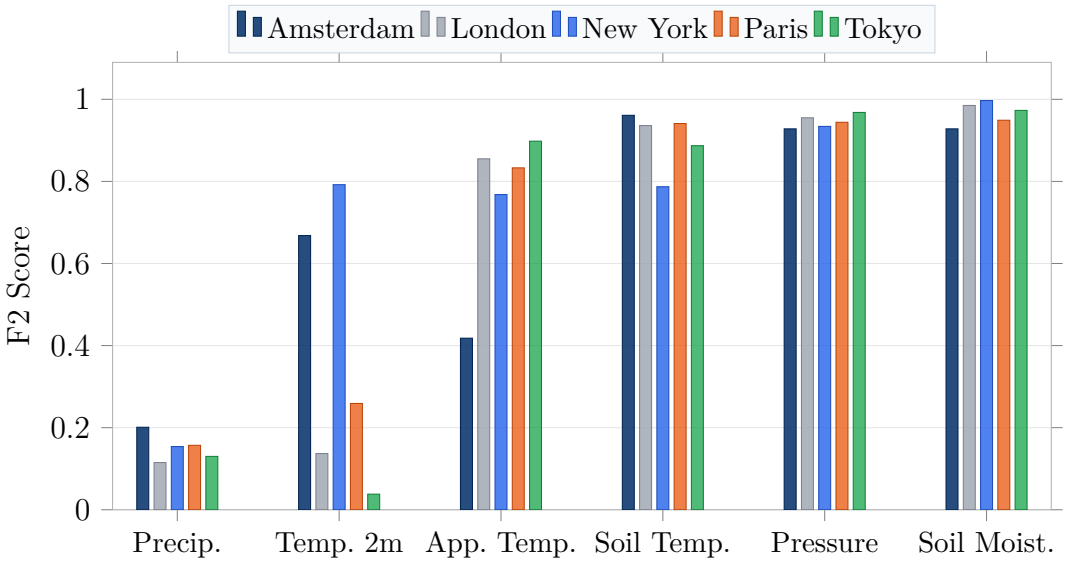

Figure 9: F2 scores across all five cities and all six columns, ordered by average F2 from left to right. Precipitation is consistently low across all cities. Apparent temperature and temperature vary by city. Soil and pressure columns are reliably high everywhere, making them the strongest candidates for adaptive profiling.

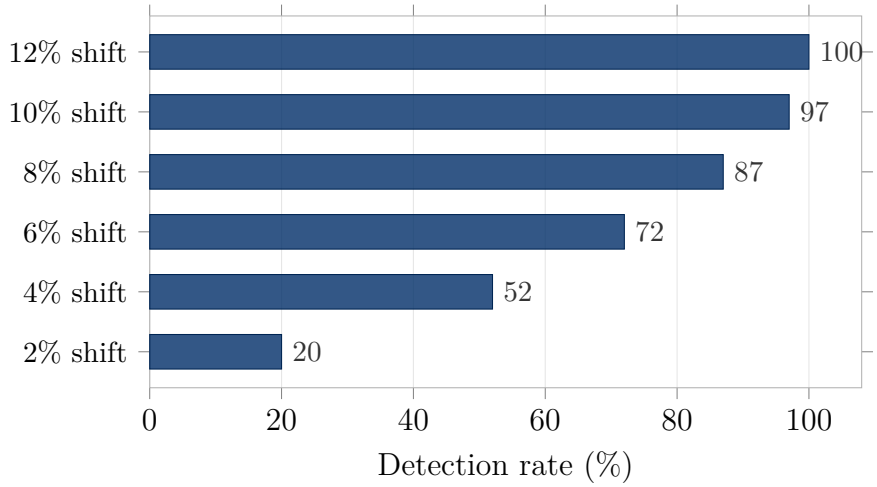

Figure 10: Detection rate by injected shift size for Amsterdam temperature (univariate LOF). The shift is expressed as a percentage of the column's natural value range. Very small anomalies (2%) are hard to distinguish from normal variation. Anomalies of 10% or more are caught reliably.

Column	Untuned baseline			Tuned AutoML		
	Model	F2	Recall	Model	F2	Recall
INDO	IForest	0.769	0.835	HBOS	0.929	0.961
ITSDO	IForest	0.788	0.924	ECOD	0.902	0.911

Table 1: F2 and recall before and after AutoML tuning on GB electricity demand.

Table 2 shows the detection breakdown for the tuned run. Missed rates stay below 9% for both columns.

Column	Synthetic	Caught	Missed rate	Extra flags	FP rate
INDO	103	99	3.9%	22	0.30%
ITSDO	79	72	8.9%	11	0.15%

Table 2: Detection breakdown for the tuned AutoML run on GB electricity demand. Extra flags are non-synthetic rows scored as anomalous by the model.

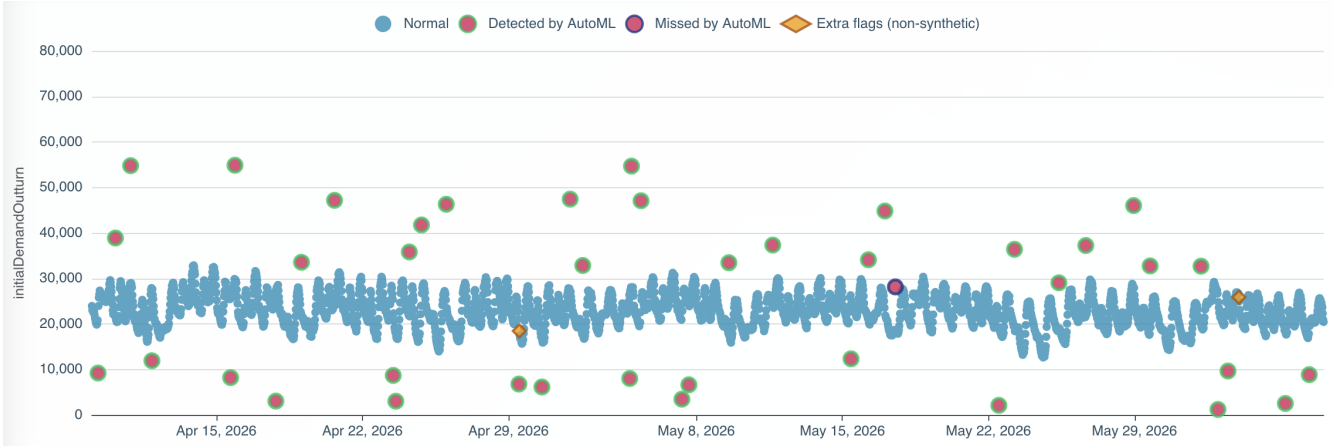

Figure 11: Detection overlay for GB initial demand outturn (INDO). Green rings are detected synthetic anomalies, purple rings are missed anomalies, and orange diamonds are extra flags on non-synthetic records (22 flags, 0.30% FP rate). The tuned HBOS model catches 99 of 103 injected anomalies.

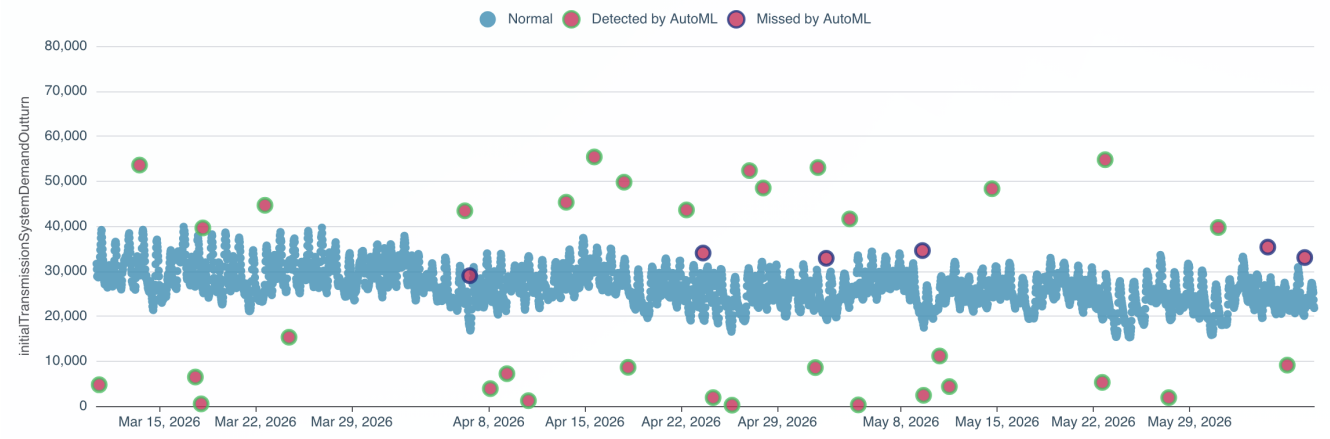

Figure 12: Detection overlay for GB transmission system demand outturn (ITSDO). The tuned ECOD model catches 72 of 79 injected anomalies (purple rings mark the 7 missed). The 7 missed anomalies occur near the period of seasonal demand decline, where the local contrast used for scoring is reduced.

Shift magnitude vs. detection rate. Figure 13 compares detection rate by shift band for INDO. At the 50–60% band the tuned model detects 95% of anomalies versus 67% for the untuned baseline; the tuned model reaches 100% at shifts of 80% or above, while the untuned baseline reaches 90–94%. The benefit of tuning is largest where the signal is weakest.

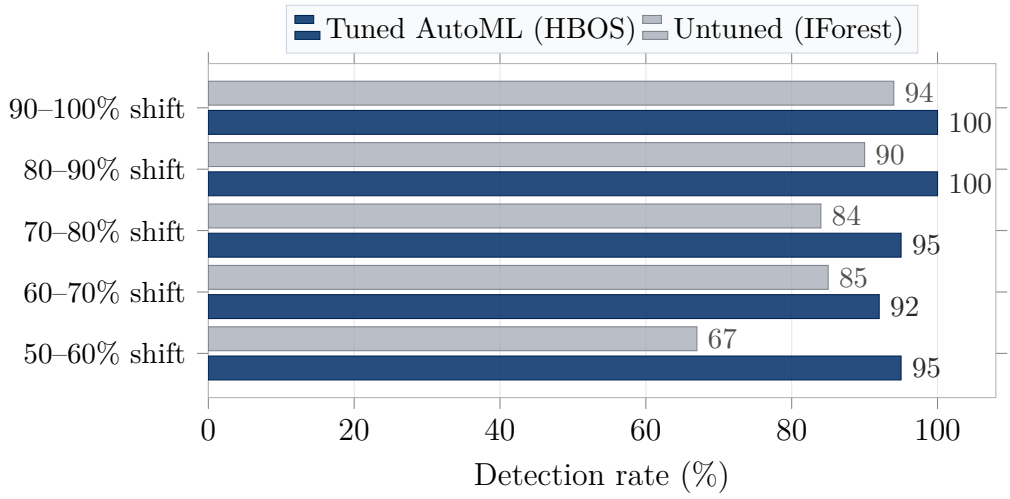

Figure 13: Detection rate by injected shift size for INDO (tuned AutoML vs. untuned IForest baseline). The tuned model catches all anomalies at 80% shift or above, while the untuned baseline reaches 90–94%. The tuned run provides the largest benefit at the 50–60% shift band, lifting detection from 67% to 95%.

6.1.7 Sensitivity to Anomaly Rate and Shift Magnitude

The electricity results so far fix a single anomaly rate (≈2.5%) and use large 50–100% shifts. To map *where* AutoML detection succeeds or fails on this domain, we sweep the two factors that govern difficulty: the *anomaly rate* (the fraction of records made anomalous) and the *shift magnitude* (how far each injected value is moved, in either direction). For every combination we inject fresh anomalies into the raw demand series and run the identical AutoML search used throughout this work (Optuna over the same PyOD detectors, F2 objective, seasonality-adjusted contextual features), changing nothing about the detector itself. We report the *capture rate*: the share of the injected anomalies the model flags, that is, recall against the known ground truth, the same quantity Figure 13 reports for one setting. The grid covers anomaly rates of $\{1, 2, 5, 10, 20\}$% and shift magnitudes of $\{10, 20, 30, 40, 50\}$% with random sign where each cell is the mean of three injection seeds (the across-seed variation is small), and results are reported separately for the two demand columns, INDO and ITSDO. Table 3 gives the capture rates and Figure 14 shows the same values as a surface.

INDO

	Shift magnitude				
Rate	10%	20%	30%	40%	50%
1%	49	63	78	84	86
2%	57	77	82	85	90
5%	66	83	92	91	94
10%	60	84	91	88	93
20%	49	65	74	78	81

ITSDO

	Shift magnitude				
Rate	10%	20%	30%	40%	50%
1%	48	73	74	87	93
2%	53	81	86	88	91
5%	71	85	90	95	90
10%	62	85	91	93	90
20%	48	65	74	78	80

Table 3: Capture rate (%) of injected anomalies for GB electricity demand, by anomaly rate (rows) and shift magnitude (columns), for the two demand columns. Each value is the recall of the injected ground truth, averaged over three injection seeds. Cells are shaded from red (low capture) to green (high). Detection strengthens with shift magnitude and is strongest at intermediate anomaly rates (5–10%).

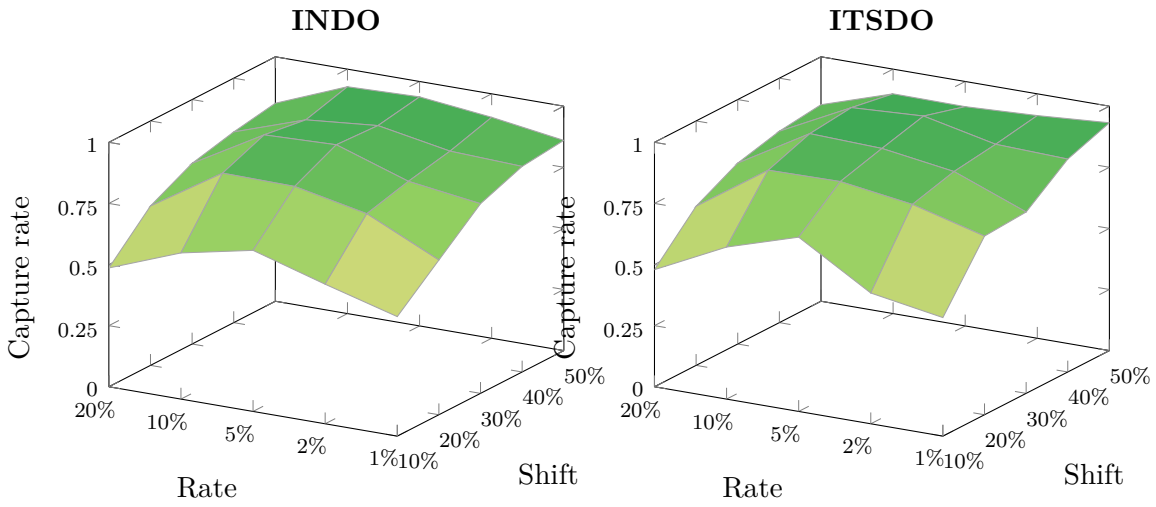

Figure 14: Capture-rate surface over anomaly rate and shift magnitude for the two demand columns (same values as Table 3; height and colour both encode capture rate). The surface rises steeply along the shift axis and ridges at the intermediate 5–10% anomaly rates, falling off at the 1% and 20% extremes.

Two effects stand out, both consistent with the rest of the thesis. First, **shift magnitude is the dominant driver of detection**. Capture climbs steeply from roughly one half at a 10% shift to 80–95% by a 40–50% shift, the same monotone trend already seen for weather (Figure 10) and for the single-rate electricity run (Figure 13). A 10% shift on demand is a genuinely hard contextual anomaly: it often stays inside the normal daily range and is unusual only relative to the time of day, so a large fraction slips past. From a 30% shift onward the model catches the clear majority (74–92%).

Second, **the anomaly rate has a sweet spot around 5–10%**. Detection is weaker at both extremes. At 1% the unsupervised search has very few positive examples to calibrate its decision threshold against, so it stays conservative; at 20% the injected anomalies contaminate the very distribution the model treats as normal, pulling the learned "normal" band toward them so that it begins to accept them. This is a direct empirical instance of the unsupervised limitation noted in Section 7, that anomalies already present in the training data are learned as normal. INDO and ITSDO produce almost identical surfaces, as expected since ITSDO is INDO plus station load and interconnector exports, so the two series move together. Across the grid Optuna again favoured the tail- and density-based detectors (ECOD and HBOS) that suit demand's tight, seasonal distribution, the same families it selected in the main electricity run (Table 1).

6.2 Experiment 2: Practical Overhead and Scaling

The second experiment answers the second research sub-question: is ML-based profiling practical in terms of pipeline overhead and resource usage?

A built-in projection tool in the profiling library benchmarks training time across different data

sizes, column counts, and trial budgets, then fits an empirical scaling law to the results. We ran it across approximately 90 benchmark configurations on the same hardware used for production. This tool lets any user predict how long training will take at their full data scale before committing to a deployment.

We express cost as wall-clock training time because that is the quantity cloud platforms charge for: compute clusters are rented per unit of time, and the same workload costs more or less depending on the cluster's core count, memory, and instance type. Predicting how long the detection step runs therefore directly predicts what it costs to operate.

6.2.1 The Scaling Formula

The formula is inspired by work in ML research showing that training behavior follows power laws. Kaplan et al. [19] showed that for the majority of AI models, training performance scales predictably with data size, model size, and compute. Each factor contributes independently, so they can be multiplied together. The result has the shape: *training performance* = *constant* × *(data size)*^{exponent} × *(model size)*^{exponent}. Bordelon et al. [4] later showed theoretically that training time, data size, and model size each carry their own power-law exponent, and those exponents can differ from each other.

The formula uses the same structure where inputs are rows (n), columns (m), and Optuna trials (k). Hardware power directly affects the α constant across all benchmark runs. The fitted formula is:

$$T(n,\, m,\, k) \;=\; \alpha \cdot n^{\,\beta} \cdot m^{\,\delta} \cdot k^{\,\gamma}$$

Table 4 explains what each symbol means and where its value comes from.

Symbol	Name	Definition	Expected range
n	Rows	Number of records used for training	≥ 1
m	Columns	Number of `automl: true` features to profile	≥ 1
k	Trials	Models and parameter sets Optuna tries	≥ 1
α	Baseline constant	Per-fit baseline cost; depends on hardware and data	> 0
β	Row exponent	How training time grows with rows	0.3–1.2
δ	Column exponent	How training time grows with columns	0–1
γ	Trial exponent	How training time grows with trials	≈ 1

Table 4: Inputs and fitted parameters. n, m, and k are known at runtime, whereas α, β, δ, and γ are fitted from benchmark runs. The expected-range column shows where each exponent typically falls for algorithms in projection experiments in the Adaptive Profiler library, based on input benchmark data.

The exponents β, δ, γ primarily reflect properties of the algorithms and data structure rather than hardware, while α captures the baseline per-fit cost of the benchmark machine and data. The column exponent δ depends on how the columns are profiled and therefore spans 0 to 1. When several columns are fed to a single shared model, adding a column increases only the feature dimensionality of one fit, which barely changes its cost, so $\delta \to 0$. The benchmark, which fits one model over m feature columns, measures $\delta \approx 0$ for this reason. When each column instead trains its own independent model, as in this pipeline's per-column design, the total cost is the sum over columns and grows roughly linearly, so $\delta \to 1$. These parameters were fitted on one specific data batch, with Optuna drawing across the same five detectors it uses in production. Because the realized time depends on which detectors the search spends most of its trials on, and on the size and structure of the incoming data, a substantially different dataset, model mix, or machine can shift the constants. The fitted values should therefore be read as calibrated to the benchmark setup rather than as universal constants. In practice, the projection tool is re-run on a sample of the target data so that α and the exponents reflect that specific workload. All four can only be obtained by running the tool across different configurations and measuring real training times. Fitted over the roughly 90 benchmark configurations, the overall formula matches about 86% of the variation in the measured training times, so the power-law form captures most of what drives training cost.

6.2.2 Scaling Behavior Depends on Algorithm Choice

The key exponent is β, which controls how training time grows as the dataset gets larger. Table 5 shows what different values of β mean in practice.

β value	Scaling behavior	Effect of doubling the data
$= 1.0$	Linear	Training time also doubles
< 1.0	Sub-linear	Training time grows slower than the data
> 1.0	Super-linear	Training time grows faster than the data
IForest (best): 0.307	Sub-linear	Doubled data took 1.24× more time

Table 5: What different values of β mean for how training time scales with data size.

Figure 15 shows per-algorithm training time projections. The fitted β values span from 0.307 (IForest, sub-linear) to 1.175 (LOF, super-linear). The key finding is that **algorithm choice determines scaling behavior more than data size alone**. These β values were fitted on the weather benchmark data used in this work and also depend on the data's dimensionality and the hyperparameter ranges searched; they should therefore be read as calibrated to this dataset and re-fitted for substantially different data, even though they broadly track each detector's known computational complexity.

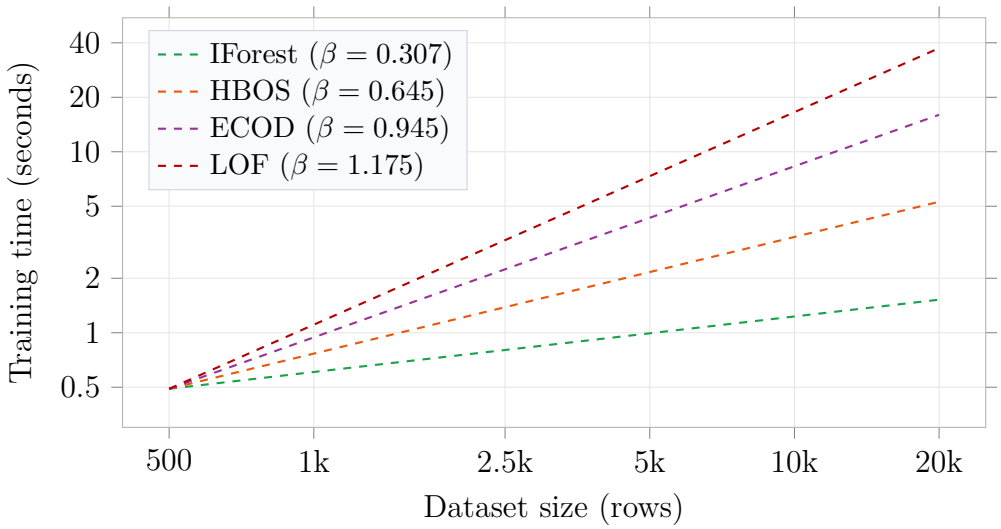

Figure 15: Per-algorithm training time projections on a log-log scale, anchored at the same starting point (500 rows). Steeper slope indicates higher β. LOF is the only super-linear algorithm.

Table 6 shows the effect of β on scaling for each algorithm when data size grows.

Algorithm	β	2× data	10× data
IForest	0.307	1.24× more time	2.03× more time
HBOS	0.645	1.56× more time	4.42× more time
ECOD	0.945	1.93× more time	8.81× more time
LOF	1.175	2.26× more time	14.96× more time

Table 6: How training time multiplies when data grows, per algorithm.

6.2.3 Per-Algorithm Scaling

Figure 16 summarizes the fitted β per algorithm. IForest is the most scalable. LOF is the only super-linear algorithm. Each β is fitted separately for each detector, and every fit is accurate: the fitted curve matches between 92% (IForest) and over 99% (ECOD, COPOD, and LOF) of the variation in the measured training times, so it tracks each detector's real timings closely.

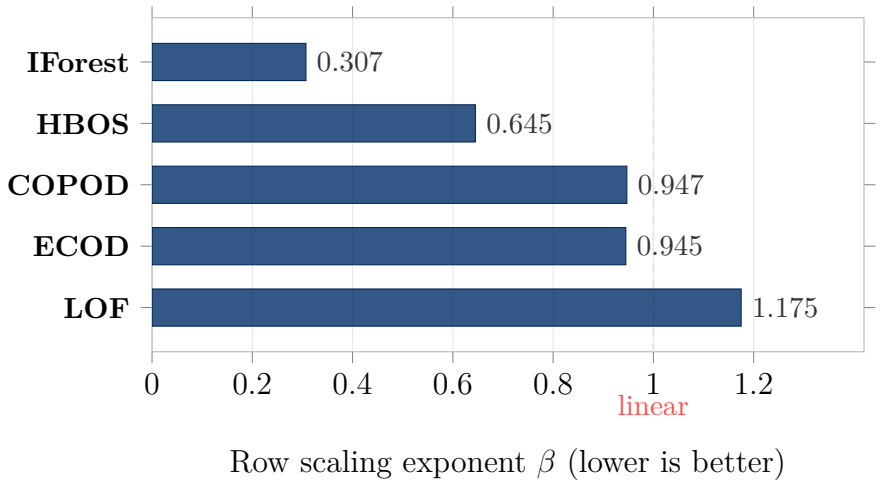

Figure 16: Fitted row scaling exponent β per algorithm from 90 benchmark observations. Each β is fitted separately per detector, and the fitted curves match the measured training times closely (from 92% of the variation for IForest to over 99% for ECOD, COPOD, and LOF).

6.2.4 How Close Is the Projected Time to Reality?

The scaling formula is only useful if the time it *estimates* is close to the time training actually *takes*. Because the projection tool is meant to be calibrated on a small sample and then extrapolated to the full dataset, its accuracy is best judged as a function of how much data it is given. We measured this on surface pressure, the most predictable column in the detection experiments (Section 6): the real training time was recorded across a grid of row counts (median of seven warm repeats, using the ECOD detector Optuna selected for this column and $k = 25$ trials per column). For each data fraction the law was fit on only the rows up to that fraction and extrapolated to the full $n_{\text{full}} = 20{,}352$ rows, then compared against the measured full-scale time. The time reported throughout is the cost of one full per-column search, that is, all 25 trials. At 25% this is a four-fold extrapolation; at 100% it reduces to interpolation. A small fixed per-fit offset is included in the fit, so the exponent reflects the true data scaling rather than being dragged down by the constant setup cost that dominates the smallest samples.

Data	Rows	β	Pred.	Actual	Error
25%	5,088	1.09	0.073 s	0.067 s	8.9%
50%	10,176	1.05	0.069 s	0.067 s	3.0%
75%	15,264	1.02	0.067 s	0.067 s	0.5%
100%	20,352	1.03	0.067 s	0.067 s	0.5%

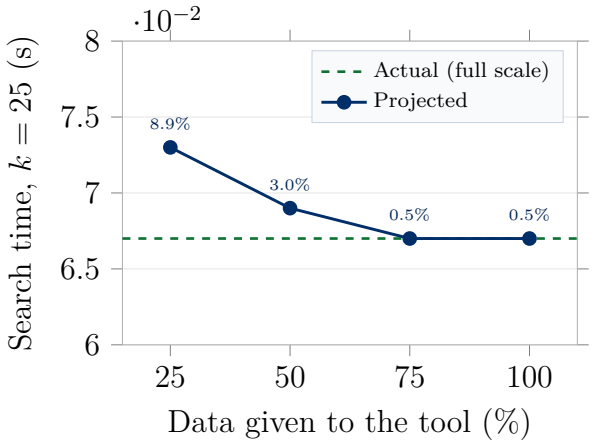

Table 7: Projected versus measured full-scale training time for surface pressure (ECOD, $n_{\text{full}} = 20{,}352$, $k = 25$), as the projection tool is calibrated on an increasing fraction of the data. Times are the cost of one full per-column search (all 25 trials). **Left:** the rows seen, fitted row exponent β, projected and actual search time, and the resulting error. **Right:** the projected time (blue, labelled with its error) converging onto the measured full-scale time (green dashed). From three quarters of the data the two are visually indistinguishable.

The estimate is already within 8.9% of reality from only a quarter of the data and tightens quickly: 3.0% at half and 0.5% once the tool has seen three quarters of the scale, where the projected and measured times coincide. The fitted exponent stays close to $\beta \approx 1$, consistent with ECOD's near-linear cost, and the error shrinks because the extrapolation reach falls from fourfold at 25% to none at 100%. In practice a data engineer can calibrate the tool on roughly half of the target data and trust the projected training time to within a few percent before committing to a full deployment, while even a 25% sample yields a reliable order-of-magnitude estimate. The columns that project most accurately are those with stable, near-linear scaling such as surface pressure, the same columns that proved easiest to profile in the detection experiments.

6.3 Experiment 3: AutoML vs. Neural Detector Search

This experiment uses the setup described in Section 5.2.3: the same five-city dataset, preprocessing, and unsupervised objective, with a single Optuna search of 25 trials per column whose only difference between conditions is the search space, classical PyOD detectors for AutoML versus PyTorch autoencoders (AE/VAE) for Neural HPO.

6.3.1 Detection Results

Table 8 and Figure 17 summarize the outcome. AutoML scored higher in 29 of 30 univariate cases and won decisively (margin > 0.02) in 27 of those 30. The three near-ties are: London surface pressure (AutoML 0.955 vs. Neural HPO 0.935, margin 0.020), New York surface pressure (AutoML 0.934 vs. Neural HPO 0.942, where Neural HPO leads by 0.008), and Tokyo temperature

(AutoML 0.038 vs. Neural HPO 0.027, margin 0.011 with both models scoring very low). In the remaining 27 cases AutoML is decisively ahead, and it runs 9.13× faster overall on the same trial budget (same 25 trials).

Metric	AutoML	Neural HPO (AE/VAE)
Mean F2 (univariate)	**0.683**	0.167
Mean Recall	**0.765**	0.196
Mean Precision	**0.618**	0.153
Mean total time / column	**2.92 s**	26.69 s

Table 8: AutoML vs. Neural Hyperparameter Optimization aggregate results across 5 cities × 6 columns.

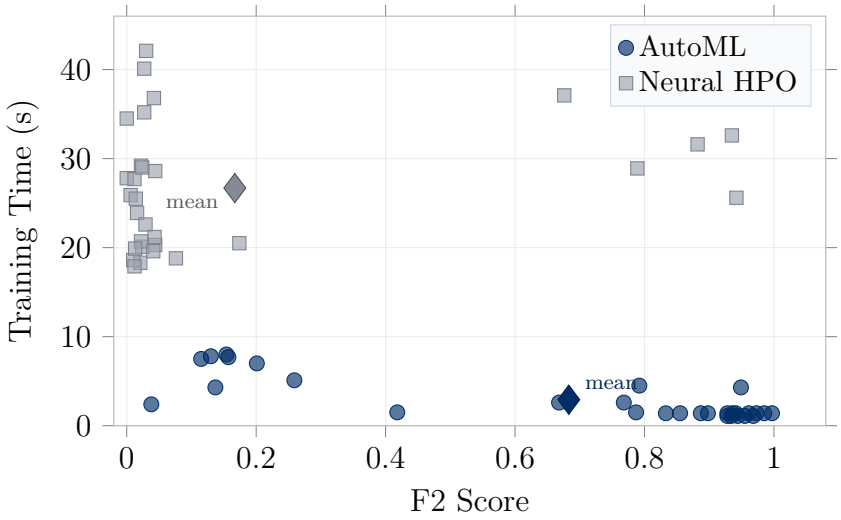

Figure 17: F2 score vs. training time for all 30 models per approach (one point per city–column pair). AutoML models cluster bottom-right: high F2, low training time. Most Neural HPO models sit top-left: low F2, high training time. The five Neural HPO outliers with high F2 are surface pressure columns, where neural reconstruction is more competitive on quality but remains around 10× slower than AutoML. Filled diamonds mark the mean of each approach.

Table 9 shows the full per-city, per-column breakdown with measured search time for each approach.

7 Discussion

The results invite a closer look at what drives them: what makes some columns easy to profile and others not, why classical AutoML outperforms neural detector search, how threshold calibration

City	Column	AutoML			Neural HPO		
		Model	F2	Time (s)	Model	F2	Time (s)
Amsterdam	App. Temp.	LOF	0.418	1.5	AE	0.006	25.9
Amsterdam	Precip.	LOF	0.201	7.0	VAE	0.000	27.8
Amsterdam	Soil Moist.	LOF	0.928	1.4	VAE	0.174	20.5
Amsterdam	Soil Temp.	LOF	0.961	1.4	AE	0.044	20.3
Amsterdam	Pressure	ECOD	0.928	1.1	VAE	0.882	31.6
Amsterdam	Temp. 2m	LOF	0.668	2.6	VAE	0.027	40.1
London	App. Temp.	LOF	0.855	1.4	VAE	0.024	20.1
London	Precip.	LOF	0.115	7.5	VAE	0.014	25.5
London	Soil Moist.	LOF	0.985	1.4	AE	0.044	28.6
London	Soil Temp.	LOF	0.936	1.4	AE	0.021	18.3
London	Pressure	ECOD	0.955	1.1	VAE	0.935	32.6
London	Temp. 2m	LOF	0.137	4.3	AE	0.010	18.6
New York	App. Temp.	LOF	0.768	2.6	VAE	0.000	34.5
New York	Precip.	LOF	0.154	8.0	VAE	0.016	23.9
New York	Soil Moist.	LOF	0.997	1.4	AE	0.029	22.6
New York	Soil Temp.	LOF	0.787	1.5	AE	0.043	21.2
New York	Pressure	ECOD	0.934	1.1	AE	0.942	25.6
New York	Temp. 2m	LOF	0.792	4.5	AE	0.022	20.7
Paris	App. Temp.	LOF	0.833	1.4	AE	0.012	27.7
Paris	Precip.	LOF	0.157	7.7	VAE	0.022	29.2
Paris	Soil Moist.	LOF	0.949	4.3	VAE	0.030	42.1
Paris	Soil Temp.	LOF	0.941	1.4	VAE	0.041	19.6
Paris	Pressure	HBOS	0.944	1.1	AE	0.676	37.1
Paris	Temp. 2m	LOF	0.259	5.1	AE	0.012	17.9
Tokyo	App. Temp.	LOF	0.898	1.4	AE	0.024	29.0
Tokyo	Precip.	LOF	0.130	7.8	AE	0.013	19.9
Tokyo	Soil Moist.	LOF	0.973	1.4	AE	0.042	36.8
Tokyo	Soil Temp.	LOF	0.887	1.4	VAE	0.076	18.8
Tokyo	Pressure	ECOD	0.968	1.1	VAE	0.789	28.9
Tokyo	Temp. 2m	COPOD	0.038	2.4	AE	0.027	35.2

Table 9: Full detection results across all cities.

trades off false positives against recall, what training on large datasets requires in practice, and where the evaluation falls short.

7.1 Detection Effectiveness

Column characteristics determine performance. Surface pressure and soil columns have tight, predictable distributions, so even small shifts are statistically unusual. Temperature is noisier due to seasonal variation, and some injected anomalies overlap with natural extreme values. Precipitation is the hardest column because it is heavily right-skewed (most hours have zero rain).

When AutoML is not the right choice. Figure 18 shows Amsterdam precipitation which follows a heavily right-skewed distribution: most hours record zero rain, and the hours that do have rainfall are highly variable with no stable seasonal shape. A model trained on this column had difficulty distinguishing an injected anomaly from a naturally extreme value in this dataset, which explains the consistently low F2 scores for precipitation across all five cities in these experiments. Columns with this kind of distribution may not be good candidates for ML-based anomaly detection. Running the profiling pipeline on a data sample before production deployment lets a data engineer identify these columns early and set `automl: false` for them rather than spending trials on a model that is unlikely to converge to a useful threshold.

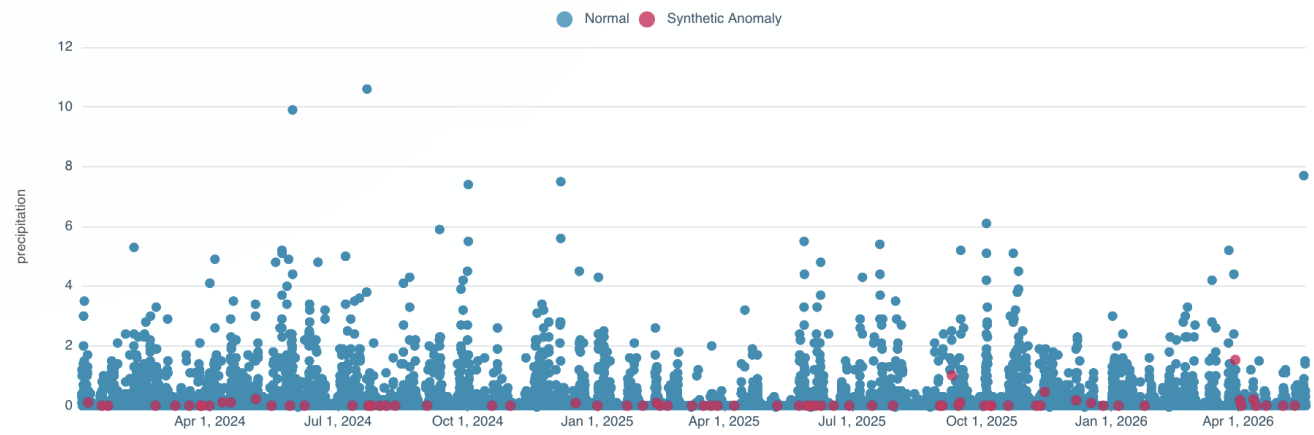

Figure 18: Amsterdam precipitation from Apr 2024 to Apr 2026. Most hours record zero rain and the rest are highly variable, leaving no stable pattern to learn. AutoML performed poorly on anomaly detection for this task due to the nature of this specific data.

Why LOF was chosen most often. LOF measures local density by comparing each value to its nearest neighbors in feature space. For univariate weather columns, this captures how isolated a value is within the distribution of recent readings. An injected shift lands in a sparse region of that distribution, far from typical values, and therefore receives a high anomaly score.

Why ECOD was selected for surface pressure. Atmospheric pressure stays within a very narrow range most of the time. ECOD scores each value by how far it sits in the tail of the observed distribution. An injected shift lands immediately in that tail, making it easy to flag. Figure 19

shows Amsterdam surface pressure: injected anomalies fall clearly outside the normal pressure band. The same pattern held in 4 of the 5 cities. Paris used HBOS for surface pressure instead, though also achieving high recall.

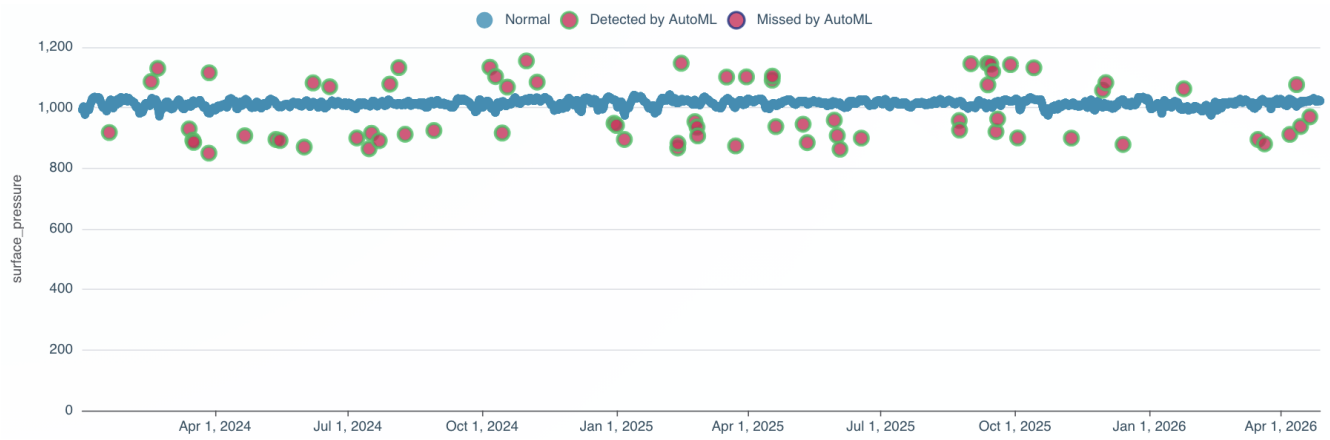

Figure 19: Amsterdam surface pressure from Apr 2024 to Apr 2026 with AutoML. ECOD selected by Optuna achieves 100% recall where all injected anomalies fall clearly outside the tight normal pressure band.

Multi-city patterns. Performance is consistent across cities for the well-structured columns (surface pressure, soil moisture), but highly variable for noisy ones like temperature. London and Tokyo show very low temperature F2 scores (0.14 and 0.04), while Amsterdam and New York are much higher (0.67 and 0.79). This suggests that temperature is harder to profile in climates with more erratic variation. In such cases, the decision threshold should be tuned on validation data to detect failure points while controlling false alarms.

Electricity data confirms the same pattern. The electricity benchmark (Section 6.1.6) reaches F2 of 0.929 and 0.902 on INDO and ITSDO respectively, consistent with the high scores seen on stable weather columns such as surface pressure and soil moisture. Optuna selected HBOS and ECOD, the same algorithm families that excelled on tight, well-bounded distributions in the weather experiments. The before-and-after comparison isolates the value of the tuning step: the untuned IForest baseline produced extra-flag rates of 0.84–1.02% while simultaneously missing more anomalies, whereas the tuned run cut the false-positive rate to 0.15–0.30% and improved recall. The electricity anomaly shifts (50–100% of the original value) are far larger than the weather injections (2–12%), so the high recall across both runs is expected.

Shift magnitude. Detection reaches near 100% for shifts of 10 to 12% of the column's value range, and drops to around 50 to 70% for smaller shifts of 4 to 6%. This threshold is useful for deciding if the model is sensitive enough for a given use case.

7.2 AutoML vs. Neural Hyperparameter Optimization

The detailed scores are reported in Section 6.3. This section focuses on why the two model families behave so differently. Classical detectors win because they match the structure of the problem: LOF, ECOD, HBOS, and COPOD score each point by its local density or tail probability, which is exactly what isolates a point anomaly embedded in a seasonal pattern. Autoencoders instead learn a single global reconstruction mapping and, as discussed in Section 3.4, tend to reconstruct anomalies almost as accurately as normal values, so reconstruction error separates the two classes poorly on low-dimensional tabular columns.

The few near-ties are consistent with this explanation rather than exceptions to it. Surface pressure is the one column where neural reconstruction is competitive, because its tight, near-Gaussian distribution makes reconstruction error a relatively reliable score. The Tokyo temperature near-tie is not a genuine win for either approach: both score below 0.04 on a column whose precipitation-like noise defeats any unsupervised model. The practical implication is that the higher search cost of the neural models buys no accuracy on this kind of data, so classical AutoML is the appropriate default for tabular ETL anomaly detection.

7.3 Threshold Calibration and the False-Positive Trade-off

The model flags some normal data. On top of detecting synthetic anomalies, the model also flags a number of normal records as anomalous. This happens when a real value is statistically rare in the training window, for example, an unusually cold week or an extreme but valid pressure reading. These records are not pipeline errors, but they fall outside the pattern the model learned from recent history.

Raising the detection threshold reduces false positives for stable columns. The profiler outputs a continuous anomaly probability score for every record. A data engineer can require a higher score before a flag is raised. For surface pressure, this reduces false positives by 59% on average across all five cities while keeping recall above 80%. Apparent temperature and soil temperature improve by 41 to 47% under similar conditions.

Threshold tuning does not help all columns. For temperature in high-variability climates like London and Paris, the model assigns similar confidence scores to natural weather extremes and to injected anomalies, so raising the threshold cuts both equally and precision does not improve. Setting `automl: false` is the better option for those columns. Precipitation should be disabled across all cities since even at the best threshold F2 stays below 0.21.

7.4 Training on Big Data

Real pipelines hold far more data than models need. A data source that has been running for years will have too many rows to retrain on every day. The Adaptive Profiler library lets engineers set a training window so only the most recent rows are used. A window of a few thousand rows is sufficient to learn stable patterns while keeping training time within the bounds shown in the

Column	Model	Default		After tuning		FP
		FP rate	F2	FP rate	F2	reduction
Temperature 2m	LOF	2.38%	0.379	1.20%	0.362	37%
Apparent Temp.	LOF	0.30%	0.754	0.15%	0.680	47%
Soil Temp.	LOF	0.13%	0.902	0.08%	0.780	41%
Precipitation	LOF	0.09%	0.151	0.00%	0.134	74%
Surface Pressure	ECOD	0.07%	0.946	0.02%	0.848	59%
Soil Moisture	LOF	0.05%	0.966	0.04%	0.855	16%

Table 10: False-positive rate and F2 score before and after threshold tuning, averaged across all five cities, each with 20,352 hourly records. Tuned thresholds were determined after the experiment using a script that aimed to balance the false-positive rate and F2 score, with injected anomalies treated as ground truth. Each FP-rate column is the mean across the five cities, and the FP-reduction column is the mean of the per-city reductions, so it does not equal the change between the two rounded, city-averaged FP-rate columns shown.

scaling experiment.

Scheduled retraining keeps detection accurate over time. In Apache Airflow, the retraining step can be added as a recurring DAG task on any schedule. The profiler's training mode runs as a single command, so a weekly or monthly job is simple to set up alongside the ingestion pipeline. If the data source changes due to seasonal shifts, sensor recalibration, or an upstream update, the next retraining run picks up the change without any manual work.

7.5 Limitations of the Evaluation

The evaluation relies on synthetically injected anomalies. Real data quality problems may look different: they may be sustained drift rather than point anomalies, or they may affect multiple columns at once.

All experiments use weather data, which has strong seasonal patterns and relatively stable distributions. The approach may behave differently on other types of data such as financial transactions or IoT sensor streams with abrupt regime changes.

If anomalies already exist in the historical training data, the model will learn to treat them as normal. This is a fundamental limitation of unsupervised learning.

The approach trains one model per column independently. Anomalies that only appear as correlated shifts across multiple columns simultaneously would not be detected by this design.

8 Conclusions and Further Work

Taken together, the experiments answer the three sub-questions on the effectiveness, practicality, and model choice of learning-based anomaly detection at ingestion. The conclusions are summarized below, followed by directions for further work.

8.1 Conclusion

This thesis investigated whether learning-based anomaly detection is a reliable, low-cost addition to rule-based quality checks at the point where data first enters ETL pipelines. The experiments confirm that AutoML-based profiling is a practical complement to rule-based checks.

Effectiveness. Machine learning detected statistically unusual values that pass all rule-based checks. On columns with stable, predictable distributions such as surface pressure and soil moisture, the approach achieved strong detection rates across all five cities, with scores from 0.93 to 1.00 (where 1.0 is perfect detection). On noisier columns such as temperature, detection varied widely from 0.04 to 0.79 depending on local climate variability.

Practicality. Integrating the detection step required minimal changes to the existing pipeline configuration. Training time grows slowly for the most scalable algorithm, Isolation Forest, keeping overhead manageable as data volume increases. Algorithm choice matters significantly at scale: the least scalable algorithm takes nearly 15 times longer when data volume grows 10-fold, while the most scalable takes only twice as long. The library is configured with one line per column, keeping the production setup practical.

Learning model. Classical machine learning outperformed deep learning architecture search in 29 of 30 column-level detection tasks. AutoML ran 9.13× faster on average and achieved a 4.1× higher mean F2-score. Simpler statistical models are naturally better suited to detecting individual unusual values in structured tabular data than deep autoencoders, which in this unsupervised setting tend to reconstruct anomalies almost as accurately as normal values and so flag them less reliably.

Together, these results show that AutoML-based adaptive profiling can improve data quality at a manageable cost. A small pilot on a sample of the target data before full deployment is enough to determine whether the approach is a good fit, based on how stable and predictable each column's values are.

8.2 Further Work

Evaluating the approach on datasets from domains such as finance, IoT, or e-commerce would establish how well it generalizes beyond weather data.

The evaluation focused on anomalies in individual columns in isolation. Extending the approach to detect anomalies that only appear as unusual combinations across multiple columns simultaneously would make detection more comprehensive.

Adding automatic drift detection and retraining triggers would enable the system to adapt automatically when the behavior of the data source changes over time, without manual intervention.

Exposing why a record was flagged would help engineers monitor model decisions and reason about AutoML behavior in production.

The case for placing detection at ingestion rather than downstream rests here on established results from the data-pipeline literature [24, 20, 25]. A controlled within-system comparison, running the same detector once at ingestion on raw records and once downstream on the transformed and aggregated warehouse tables, and measuring the resulting difference in detection accuracy and remediation cost, would turn this argument into a direct empirical result for the specific pipeline studied here.

Declaration of AI Tool Usage

During the preparation of this thesis, the following AI tools were used:

- **Claude Code** (Anthropic) was used to assist with experiment setup implementation, including pipeline configuration scripts and development of the Adaptive Profiler library. All code was reviewed, tested, and validated by the author.
- **Amazon Q** (Amazon Web Services) assisted with AWS service configuration, including S3, IAM, and EC2 setup for the experiment infrastructure. All configurations were reviewed and verified by the author.
- **Consensus** was used to assist in discovering and summarizing academic literature during the different phases of research. All cited works were independently read and evaluated by the author.
- **ChatGPT** (OpenAI) was used to improve the language, grammar, and readability of the written text. All content reflects the author's original analysis, findings, and ideas.

The author takes full responsibility for the accuracy and integrity of all content in this thesis.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2019)*, pages 2623–2631. Association for Computing Machinery, 2019.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24 (NeurIPS 2011)*, pages 2546–2554. Curran Associates, Inc., 2011.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[4] Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. A dynamical model of neural scaling laws. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 4345–4382. PMLR, 2024.

[5] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*, pages 93–104. Association for Computing Machinery, 2000.

[6] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J. G. B. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, 2016.

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[8] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[9] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*, pages 2201–2206. Association for Computing Machinery, 2016.

[10] Databricks. AI ETL: How artificial intelligence automates data pipelines, 2025. Industry article explaining the motivation for AI-driven ETL pipelines.

[11] DataKitchen. Flip the script on data quality: Shift-left, shift-down, and take control, 2025. Explains the shift-left data quality principle.

[12] Elexon. BMRS insights API: GB national electricity demand. Elexon Insights API documentation available at https://bmrs.elexon.co.uk/api-documentation, 2026. Accessed June 2026.

[13] Andrew Emmott, Shubhomoy Das, Thomas G. Dietterich, Alan Fern, and Weng-Keen Wong. Systematic construction of anomaly detection benchmarks from real data. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description (ODD '13)*, pages 16–21, 2013.

[14] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (HBOS): A fast unsupervised anomaly detection algorithm. In *KI 2012: Poster and Demo Track*, pages 59–63, 2012.

[15] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1705–1714. IEEE, 2019.

[16] Raghu Gopa. AI augmented ETL pipelines for automated data quality anomaly detection and governance. *International Journal of Computational and Experimental Science and Engineering*, 11(4), 2025.

[17] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. ADBench: Anomaly detection benchmark. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*. Curran Associates, Inc., 2022.

[18] Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[19] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[20] Bojan Karlas, Babak Salimi, and Sebastian Schelter. Navigating data errors in machine learning pipelines: Identify, debug, and learn. In *41st IEEE International Conference on Data Engineering (ICDE 2025)*, pages 4523–4529. IEEE, 2025.

[21] Edwin J.Y. Koh, Eiman Amini, Shruti Gaur, Miguel Becerra Maquieira, Christian Jara Heck, Geoffrey J. McLachlan, and Nick Beaton. An automated machine learning (AutoML) approach to regression models in minerals processing with case studies of developing industrial comminution and flotation models. *Minerals Engineering*, 189:107886, 2022.

[22] Koorosh Komeili Zadeh. Adaptive data profiling ETL: Source code and system design, 2026. GitHub repository containing the full pipeline implementation, profiling library, and experiment code.

[23] Koorosh Komeili Zadeh. adaptive-profiler: AutoML anomaly detection and schema-driven data quality for ETL pipelines, 2026. Python package (v0.2.0). Source: https://github.com/kooroshkz/adaptive-profiler.

[24] Sunil Kothari, Sumukha Sharma Thoppanahalli Chandramouli, Naman Khandelwal, Parth Kulshreshtha, Ashi Jain, Kriti Banka, Tanuja Chintada, Venkata Triveni, Praveen Kumar Gulipalli, Manish Mehta, and Tao Liu. Position: Early-stage quality assurance in annotation pipelines is more cost-effective than late-stage validation. *CoRR*, abs/2605.15714, 2026.

[25] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. ActiveClean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12):948–959, 2016.

[26] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. COPOD: Copula-based outlier detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1118–1123. IEEE, 2020.

[27] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George H. Chen. ECOD: Unsupervised outlier detection using empirical cumulative distribution functions. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12181–12193, 2023.

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining (ICDM)*, pages 413–422. IEEE, 2008.

[29] Raghavender Maddali. Automating data quality assurance using machine learning in ETL pipelines. *International Journal of Leading Research Publication*, 2(6), 2021.

[30] Aiswarya Raj Munappy, Jan Bosch, and Helena Holmström Olsson. Data pipeline management in practice: Challenges and opportunities. In *Product-Focused Software Process Improvement – 21st International Conference (PROFES 2020)*, pages 168–184. Springer, 2020.

[31] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton van den Hengel. Deep learning for anomaly detection: A review. *ACM Computing Surveys*, 54(2):38:1–38:38, 2022.

[32] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. HoloClean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11):1190–1201, 2017.

[33] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, 11(12):1781–1794, 2018.

[34] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*, pages 2503–2511. Curran Associates, Inc., 2015.

[35] Dezhan Tu, Yeye He, Weiwei Cui, Song Ge, Haidong Zhang, Shi Han, Dongmei Zhang, and Surajit Chaudhuri. Auto-validate by-history: Auto-program data quality constraints to validate recurring data pipelines. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2023)*, pages 4991–5003. Association for Computing Machinery, 2023.

[36] Panos Vassiliadis. A survey of extract-transform-load technology. *International Journal of Data Warehousing and Mining*, 5(3):1–27, 2009.

[37] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. Data collection and quality challenges in deep learning: A data-centric AI perspective. *The VLDB Journal*, 32(4):791–813, 2023.

[38] Yue Zhao, Zain Nasrullah, and Zheng Li. PyOD: A Python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[39] Yue Zhao, Ryan A. Rossi, and Leman Akoglu. Automatic unsupervised outlier model selection. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, pages 4489–4502. Curran Associates, Inc., 2021.

[40] Patrick Zippenfenig. Open-meteo.com weather API, 2023.