National College of Ireland

The Hidden Cost of Real-Time: Quantifying Performance and Economic Trade-offs Between Streaming and Batch Machine Learning Across Domains

MSc Research Project
Programme Name

Shamsa Halima Kasozi Nantale
Student ID: x23344083

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Shamsa Halima Kasozi Nantale
Student ID:	x23344083
Programme:	Programme Name
Year:	2025
Module:	MSc Research Project
Supervisor:	Christian Horn
Submission Due Date:	11/08/2025
Project Title:	The Hidden Cost of Real-Time: Quantifying Performance and Economic Trade-offs Between Streaming and Batch Machine Learning Across Domains
Word Count:	1091
Page Count:	7

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:	
Date:	14th September 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):	

The Hidden Cost of Real-Time: Quantifying Performance and Economic Trade-offs Between Streaming and Batch Machine Learning Across Domains

Shamsa Halima Kasozi Nantale
x23344083

Overview

This configuration manual describes the required software tools and settings needed to replicate the experimental setup for comparing batch versus streaming machine learning algorithms on fraud detection and electricity price prediction datasets. The experiments evaluate the systematic performance gap between traditional batch processing and real-time streaming approaches

1 Required Software Versions

The following software versions must be installed to ensure reproducibility of results: Python 3.10 serves as the core programming language, with River 0.15.0 providing the streaming ML framework and scikit-learn 1.3.0 supplying the batch ML algorithms. Optuna 3.1.0 is required for Bayesian hyperparameter optimization, while pandas 1.5.3 handles data manipulation and numpy 1.24.3 performs numerical computations. For visualization and analysis, matplotlib 3.7.1 creates the plots, seaborn 0.12.2 generates statistical visualizations, and scipy 1.10.1 conducts the statistical tests. Additionally, arff 0.9 is needed to process the electricity dataset's ARFF file format.

2 Hardware Specifications

The experiments were conducted on the following system configuration, as shown below:

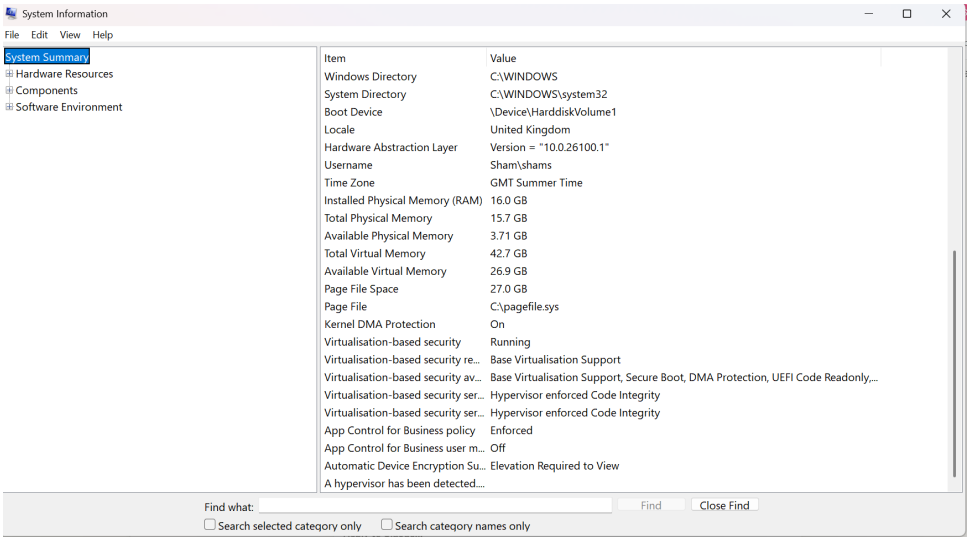

Figure 1: Hardware Specifications

The system specifications include Microsoft Windows 11 Home (Version 10.0.26100 Build 26100) running on an x64-based PC. The processor is a 13th Gen Intel(R) Core(TM) i7-13700H operating at 2.40 GHz with 14 cores and 20 logical processors. The system is an Acer Swift SFG14-71T model with 16.0 GB of installed physical memory (RAM), of which 15.7 GB is the total usable physical memory and 3.71 GB was available at the time of capture.

This configuration provides sufficient computational resources for running both batch and streaming machine learning experiments. The 16GB RAM is adequate for handling the datasets and model training processes, while the multi-core processor enables efficient parallel processing during the statistical validation and Bayesian optimization phases.

3 Development Environment

For this research, Visual Studio Code with the Jupyter Notebook extension was used to develop and execute all experiments. This setup provides an integrated environment for both code development and interactive data analysis. VS Code offers excellent support for Jupyter notebooks, allowing for cell-by-cell execution and inline visualization of results.

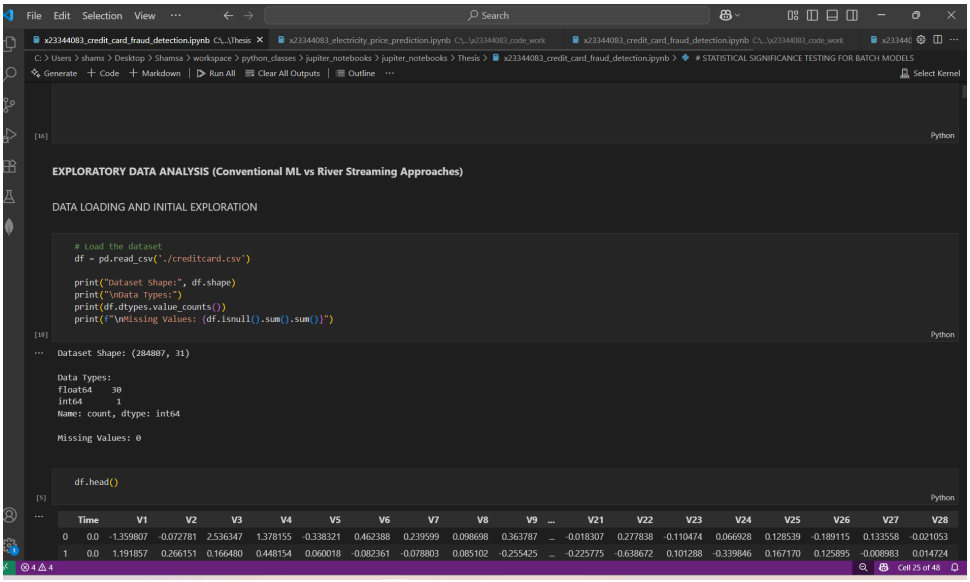

Figure 2: Hardware Specifications

To set up the development environment, first install Visual Studio Code, then install the Jupyter extension from the VS Code Extensions marketplace. The Python extension should also be installed for enhanced Python support including IntelliSense and debugging capabilities. New Jupyter notebooks can be created with the .ipynb extension directly within VS Code.

A single comprehensive notebook was created for this study containing all experimental code, from data preprocessing through model evaluation. This integrated approach allows for better reproducibility as all steps are contained in one sequential document, making it easier to trace the experimental workflow from raw data to final results.

4 Data Preparation

The experimental setup requires loading two datasets with different formats and characteristics. The credit card dataset is loaded directly from CSV format, while the electricity dataset requires conversion from ARFF to CSV format. Both datasets undergo initial quality checks and preprocessing as shown below.

```python
# Load the dataset
df = pd.read_csv('./creditcard.csv')

print("Dataset Shape:", df.shape)
print("\nData Types:")
print(df.dtypes.value_counts())
print(f"\nMissing Values: {df.isnull().sum().sum()}")
```

```python
# Load the .arff file
with open("elecNormNew.arff") as f:
    dataset = arff.load(f)

# Convert to pandas DataFrame
df = pd.DataFrame(dataset['data'], columns=[attr[0] for attr in dataset['attributes']])

# Convert the target column from 'UP'/'DOWN' strings to 1/0 integers
df['Class'] = (df['class'] == 'UP').astype(int)  # UP=1, DOWN=0
df = df.drop('class', axis=1)  # Remove original string column

# Save as CSV with proper binary target
df.to_csv("electricity_data.csv", index=False)

print("Converted successfully to electricity_data.csv")
print(f"Dataset shape: {df.shape}")
print("Class distribution:")
print(df['Class'].value_counts())
print(f"Class balance: {df['Class'].mean()*100:.1f}% UP")
```

Figure 3: Dataset Loading and Conversion Process

The credit card dataset contains 283,726 samples after duplicate removal with 30 features and exhibits extreme class imbalance with only 0.167% fraud cases. The electricity dataset contains 45,312 samples with 8 features and is relatively balanced with 42.5%

UP movements. The ARFF file is converted to CSV format with binary encoding of the target variable for compatibility with the experimental framework.

River Dataset Implementation

For streaming evaluation, both datasets require custom classes inheriting from River's FileDataset base class. The implementation follows identical methodology for both datasets, specifying dataset parameters and enabling sequential data streaming with appropriate type conversions.

```python
class CreditCardFraud(base.FileDataset):
    """Uses Credit Card Fraud Database - Local Implementation

    The datasets contains transactions made by credit cards in September 2013 by european
    cardholders. This dataset presents transactions that occurred in two days, where we have 492
    frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class
    (frauds) account for 0.1667% of all transactions.

    It contains only numerical input variables which are the result of a PCA transformation.
    Features V1, V2, ... V28 are the principal components obtained with PCA,
    the only features which have not been transformed with PCA are 'Time' and 'Amount'.
    Feature 'Time' contains the seconds elapsed between each transaction and the first
    transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be
    used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and
    it takes value 1 in case of fraud and 0 otherwise.

    """
    def __init__(self):
        super().__init__(
            n_samples=284_807,
            n_features=30,
            directory="C:/Users/shams/Desktop/Shamsa/workspace/python_classes/jupiter_notebooks/jupiter_notebooks/Thesis",
            filename="creditcard.csv",
            task=base.BINARY_CLF,  # Changed to classification
        )

    def __iter__(self):
        converters = {f"V{i}": float for i in range(1, 29)}
        converters["Class"] = int
        converters["Time"] = float
        converters["Amount"] = float
        return stream.iter_csv(self.path, target="Class", converters=converters)
```

Figure 4: River Dataset Class Implementation

The CreditCardFraud class defines 284,807 samples with 30 features for binary classification, while the ElectricityDataset class similarly defines 45,312 samples with 8 features. Both implementations use the same structure with dataset-specific parameters for n_samples, n_features, filename, and converters. The directory path must be updated to match the local file system configuration where datasets are stored. This unified approach ensures consistent streaming behavior across both experimental datasets.

5 Model Configurations

5.1 Batch Models

The batch experiments utilize three traditional machine learning algorithms from scikit-learn. Each model is configured with specific parameters to ensure reproducibility:

```python
# Train multiple conventional models
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42, max_depth=10),
    'Logistic Regression': Pipeline([('scaler', StandardScaler()),
                                     ('lr', LogisticRegression(random_state=42, max_iter=1000))])
}
```

Figure 5: Batch Model Configuration

5.2 Streaming Models

The streaming experiments employ River's incremental learning algorithms:

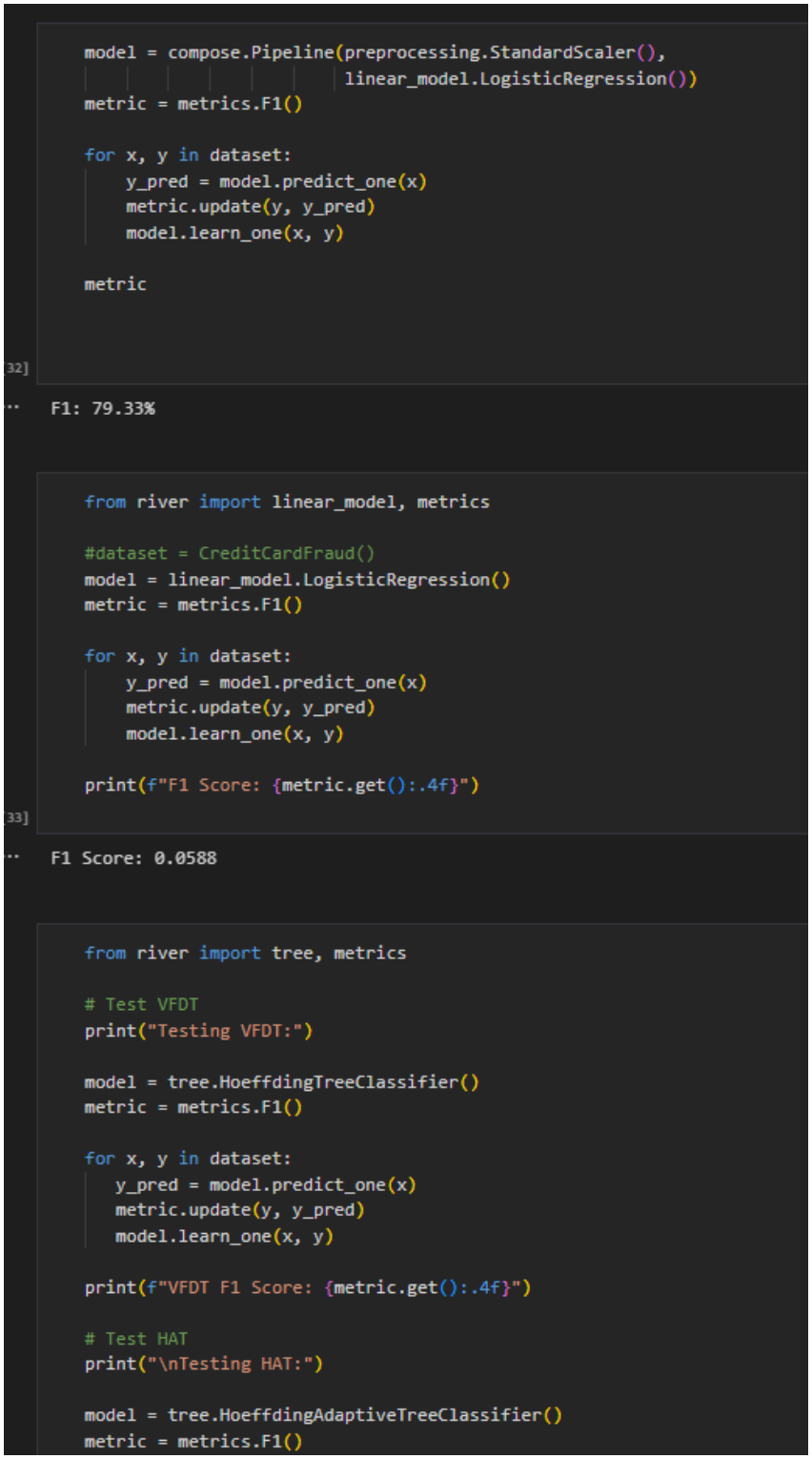

Figure 6: Streaming Model Configuration

6 Experimental Parameters

6.1 Global Settings

The experimental framework employs several global parameters to ensure consistency and reproducibility across all experiments. Processing is limited to max_samples of 50000 for computational feasibility while maintaining statistical validity. A window_size of 1000 is used for calculating moving averages and tracking performance metrics over time. The test_then_train parameter is set to True, implementing the standard streaming evaluation protocol where each sample is first used for prediction before updating the model. Statistical rigor is ensured through n_statistical_runs set to 10, with random_seeds ranging from 42 to 51 providing reproducible randomization across independent experimental runs.

6.2 Business Cost Model

The fraud detection evaluation employs literature-based cost parameters for realistic assessment. The cost per false positive is set to $18.93 based on Wallny (2022), representing customer churn cost. The cost per false negative equals the actual transaction amount, tracked dynamically during evaluation.

```
# Track actual fraud amounts for literature-based business cost
cost_per_false_positive = 18.93  # Wallny (2022)
total_missed_fraud_amount = 0.0  # Track actual amounts of missed frauds
```

Figure 7: Business Cost Configuration

6.3 Enhancement Strategy Parameters The enhancement parameters were discovered through Bayesian optimization with statistical validation, resulting in dataset-specific optimal values:

```
# EMPIRICALLY OPTIMIZED PARAMETERS (Bayesian + Statistical Validation)
enhancements = {
    'artificial_fraud': {'VFDT': 137, 'HAT': 119},
    'weighted_learning': {'VFDT': 17, 'HAT': 4},    # ← BEST PERFORMERS
    'pre_training': {'VFDT': 8751, 'HAT': 7795}
}
```

```
enhancements = {
    'weighted_learning': {'VFDT': 3, 'HAT': 3},
    'pre_training': {'VFDT': 4905, 'HAT': 4178}
}
```

Figure 8: Enhancement parameters

For the credit card dataset, weighted learning multiplies fraud case learning by 17 for VFDT and 4 for HAT. The artificial fraud strategy injects synthetic fraud labels every 137 samples for VFDT and 119 for HAT. Pre-training uses 8751 and 7795 balanced samples respectively. The electricity dataset uses more conservative parameters with weighted learning set to 3 for both algorithms and pre-training with 4905 samples for VFDT and 4178 for HAT.

7 Execution Sequence

The complete experimental pipeline requires approximately 3-4 hours to execute on the specified hardware configuration. The Bayesian optimization and statistical validation phases are the most computationally intensive, accounting for approximately 90-100 minutes of the total runtime. Users should plan accordingly and ensure adequate computational resources are available for the extended optimization process.

1. Load and preprocess datasets
2. Run batch model experiments with 10-fold cross-validation
3. Execute baseline streaming models without enhancements
4. Perform Bayesian optimization (30 trials, 3-fold CV)
5. Run enhanced streaming models with optimized parameters
6. Conduct statistical significance testing (10 independent runs)
7. Generate comparison tables and visualizations

References

Wallny, F. (2022). False positives in credit card fraud detection: Measurement and mitigation, *Proceedings of the 55th Hawaii International Conference on System Sciences*, pp. 1–10.
URL: *http://hdl.handle.net/10125/79527*