VOL. 20, NO. 19, OCTOBER 2025 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

INCREMENTAL VS BATCH LEARNING: A COMPARATIVE ANALYSIS OF EFFICIENCY AND ENERGY CONSUMPTION

Fahad Alkamli, Morched Derbali, and Tariq Mohamed Ahmed
Department of Information Technology, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
E-Mail: falkamli@stu.kau.edu.sa

ABSTRACT

Given the increasing demand for machine learning algorithms suitable for deployment on devices with limited memory and power resources, there is a clear need for frameworks that prioritize power and memory efficiency. A prominent candidate is the River framework. This study compares incremental and batch learning models, with a focus on efficiency and energy consumption. We evaluate the performance of three representative machine learning algorithms, Multinomial Naive Bayes, Logistic Regression, and Adaptive Random Forest, using the River framework, which enables instance-based learning on data streams. The findings indicate that while incremental learning can reduce memory footprint and training time, it incurs higher energy consumption during the early stages. Conversely, batch learning benefits from aggregated data but faces scalability challenges in resource-constrained environments. This paper explores the trade-offs between the two approaches and guides their applicability in domains such as cybersecurity and IoT.

Index Terms: incremental learning, batch learning, data streams, river framework, online machine learning, concept drift, real-time processing.

Manuscript Received 12 November 2025; Revised 5 January 2026; Published 20 January 2026

1. INTRODUCTION

In traditional machine learning, most libraries were designed to train models on the entire dataset at once, a process commonly known as batch learning. This approach was an issue for real-time applications that needed a fast and power-efficient library, hence the development of River, which is built upon cremeandscikit-multi flow. River is constructed to support learning from streaming data that continuously changes and evolves, allowing areas like cyber security to utilize ML, which is our focus in this paper [1]. River also opened the possibilities for IoT and similar devices that have very low processing power and limited energy to take advantage of machine learning by introducing learn one, which allows the model to learn one record data time, leading to less power use and minimal memory. In the following section, we examine River and the available features that will ease the process of incremental learning.

a) Mini-batches

This allows River to learn from small batches by calling learn many, but unfortunately, not all models in River support mini-batches. Some supported models are Multinomial NB, Linear Regression, Logistic Regression, Bernoulli NB, and Complement NB.

b) Pipelines

They are a way to chain a sequence of operations that are processed in order. They allow an instance to pass through the chain oneitemata time when using learn one. The chain allows embedding of trans formers, classifiers, regresses, and clusters. Figure-1 shows a pipeline that contains a Transformer Union, which packs multiple transformers into a single transformer. The chain will also use Random Sampler, which acts as a classifier in the pipeline.

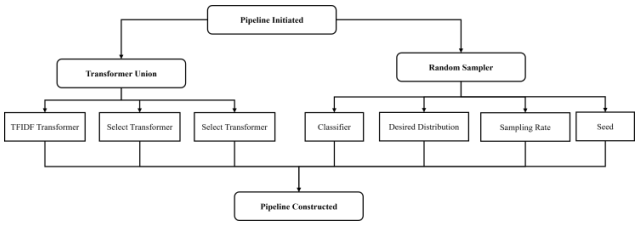

Figure-1. Example of a river pipeline with feature union and sampling.

c) Drift detection

In River, there are two abstract classes responsible for drift detection. The first is Drift Detector, which is inherited by detectors such as Page Hinkley and ADWIN. This class provides fast detection with low memory usage but does not of ferny warning signals. The second class, Drift and Warning Detector, is more advanced and integrates both warning and drift detection. Warnings are used to detect gradual or upcoming drift before it becomes severe. This class is implemented by the EDDM, DDM, HDDM_A and HDDM_W detection classes.

The rest of this paper will be structured as follows. In Section II, the literature focuses on a review of existing literature on incremental and batch learning techniques that is based on energy efficiency and real time processing of the data. The problem and the research objectives are outlined in the section III. In Section IV, the experimental setup is discussed by providing a description

of the dataset preparation, feature engineering, and learning pipelines. Section V documents and discusses the experimental performance, benchmarking similar to AUC, the amount of energy consumed, the memory utilized, and the time required to train. Under Section VI, the important observations, which include the effects of the imbalance of classes on the behavior of models, are raised. In the end, a study is concluded in Section VII with a recommendation of future research directions.

2. RELATED WORK

Incremental learning has emerged as an essential method for processing real-time streams of continuous data, as compared to the batch learning technique that presupposes a complete set of data before starting the training process. Batch learning tends to have excessive retraining costs and fails to work with non-stationary data, which excludes it, especially in the cybersecurity and IoT fields. To overcome these restrictions, Montieletal. proposed River, a Python framework focused on machine learning over data streams [1]. River accommodates instance-incremental and batch-incremental learning and combines a broad range of models for classification, regression, and clustering.

In the original study, however, there is limited evaluation of the performance of River. It only benchmarks three models based on a 1990s dataset and fails to compare the algorithms in River with more traditional batch learning methods, nor does it test them on high-frequency challenges such as cyber responses. Furthermore, key topics like managing concept drift, energy consumption, and real-time adaptability are not addressed.

Related literature comparing batch and incremental learning has pointed out the trade-offs between accuracy and flexibility. For example, Read *et al*. compared batch-incremental and instance-incremental techniques in evolving data streams and concluded that in several applications, the responsiveness of instance-incremental techniques is equivalent or even greater [2]. Similarly, Ramanath *et al.* proposed Lambda Learner, which achieves batch-level precision when applied to constrained problems under the streaming mini-batch update setting [3].

Nevertheless, there are no prior evaluations of River's incremental learning algorithms against modern batch learning baselines in terms of critical metrics such as AUC, processing time, memory usage, and energy consumption. This discrepancy motivates the proposed research, which aims to conduct a comprehensive experiment comparing River's incremental models with standard batch learning models on real-world datasets.

3. PROBLEM DEFINITION

Traditional batch-based learning systems operate under the assumption that the complete dataset is available prior to training, which renders them unsuitable for real-time and resource-constrained environments. As data volume and velocity increase, particularly in domain such has cyber security, IoT, and streaming analytics, continuous learning models become essential. This challenge can be addressed using incremental learning frameworks, such as River, which facilitate training models dynamically without the need for full retraining.

Nevertheless, despite the growing adoption of incremental learning, few studies have thoroughly investigated its functionality, resource efficiency, and adaptability in comparison to traditional batch learning under realistic data stream conditions. In particular, the trade-offs among model accuracy, memory utilization, computational efficiency, and responsiveness to concept drift remain only partially explored within modern incremental learning libraries.

This research addresses that gap by systematically analyzing and comparing incremental and batch learning models within the River framework, alongside their batch-based counterparts, across multiple performance dimensions using real-world streaming datasets. The goal is to clarify which incremental learning strategies can deliver superior outcomes under dynamic data environments.

4. EXPERIMENTAL METHODOLOGY

In this section, we will discuss the experiment setup that was used to measure the performance of the batch and incremental model, and what software was used to extract wattage from the CPU, specifically in a Windows Environment.

A. Hardware and Software

- CPU: AMDRyzen73700X8-coreprocessor.
- RAM: 32GBwithaspeedof3200MHz.
- OS: Windows10Build19045.
- Python Version: 3.13.1.
- Libre Hardware Monitor Version: 0.9.4.
- River Version: 0.22.0.

a) Dataset construction

The Apache Web App dataset [4] consists of HTTP request logs containing features such as the request method, URL, status code, referrer, and browser.

The data set was designed to work with both the incremental training model, which requires files of the same size to showcase the nature of the incremental process, and batch learning, which needs a data set consisting of accumulated files, meaning batches will double in size as the gathering of the data increases.

- Incremental Dataset Construction: A script was created to transform the dataset, dividing it into ten parts of equal size, each with 2513 records.
- Batch Dataset Construction: a script was created to trans- form the dataset into ten parts consisting of accumulated files.

b) Feature extraction and preprocessing

The experiment includes a preprocessing function to trans- form the raw HTTP request logs into a feature-rich format for learning. The script applies the following steps:

- Label Generation: If the ”Label” column is missing, it is created by marking all non-200 HTTP status codes as malicious (1) and 200 codes as benign (0).
- Request Parsing: If ”Method” is missing, the HTTP method and URL are extracted from the raw ”request” string using regular expressions.
- Missing Value Handling: Fills missing values in” Status”, ”URL”, and ”Method” with default placeholders to prevent processing errors.
- Error Flag: A binary feature named ”shows_error” is created, where requests with status codes between 300 and 499 are marked as 1.
- Combined Text Field: A placeholder ”combined_text” is created from the ”URL” field to support potential NLP- based extensions.
- Injection Detection: Two custom features, ”sql_ injection” and ”command_injection” are derived using external functions that analyze the URL for known attack patterns.
- Uncommon Method Flag: Another binary flag ”uncom- mon_method” is generated using a function that checks if the HTTP method is rare or suspicious (e.g., DELETE, TRACE).

This transformation is applied consistently before each learning cycle to ensure uniform input formatting for both incremental and batch learners.

c) Pipeline framework justification

The experimental pipeline was implemented using the river library, which provides tools for real-time and in cre-mental machine learning [1]. Unlike traditional batch-learning frameworks such as scikit-learn [5] orimblearn [6], which require the entire data set to be loaded into memory.

- They rely on full dataset ingestion through fit()and
- Predict ().
- They lack native support for dictionary-based input and column-wise streaming transformations.
- Their resampling techniques (e.g., SMOTE, random over-sampling) are incompatible with incremental data flow.

Therefore, the use of river aligns to model real-time incremental learning workflows, where continuous updates and efficient resource use are prioritized. We have also used imblearn pipelines for batch learning to set the comparison as close as possible when executing the experiment.

d) Power and energy measurement methodology

To get an accurate measurement, we have conducted twenty trials and calculated the average for each cycle. The experiment proceeds as follows: It enters a loop that validates first if the system is in an idle state which, per our experiments, is approximately 26.8 watt and only starts the learning process when the system reaches this or below this idle state otherwise, the system will wait.

Next, the capturing of initial_wattage and start_ time will start before loading the files and processing the dataset for feature extractions and other preprocessing utilities. We have deemed to include the processing of the dataset and feature extraction due to incremental and batch learning in practice, which need to account for the size of the dataset. Hence, incremental will usually mean they will need to open and load small files, but in the case of batch learning, they will need to open and load large files, which will affect the speed and performance of the overall learning process. Finally, when the learning process finishes, we record the end._time and the end_watt and the cycle of learning continues through all parts.

The energy consumption was estimated using:

River supports stance-by-instance learning using streaming APIs.

To show a comparison of pipeline features across both River and other batch-oriented frameworks, we present the key distinguished features as follows:

$$\text{Energy (Wh)} = \frac{P_{start}+P_{end}}{2} \times \frac{t_{end}-t_{start}}{3600} \quad (1)$$

River pipeline

Online transformation of features using trans form_one().

Model updates on a per-sample basis using learn_one().

Flexible integration of multiple feature extraction methods using Transformer Union.

Batch-oriented pipelines

This equation uses the trapezoidal method for estimating energy consumption.

where P_{start} and P_{end} are the power readings at the start and end of training, respectively. The estimated energy was converted into a cost using a rate of 0.18 SAR per kWh. Figure-2 illustrates the steps involved in the experiment process.

Table-1. Machine learning model implementations: incremental vs. batch

Model Type	Incremental Learning	Batch Learning
Naive Bayes	Multinomial NB	Multinomial NB
Linear Model	Logistic Regression	Logistic Regression
Decision Tree	Adaptive Random Forest	Random Forest

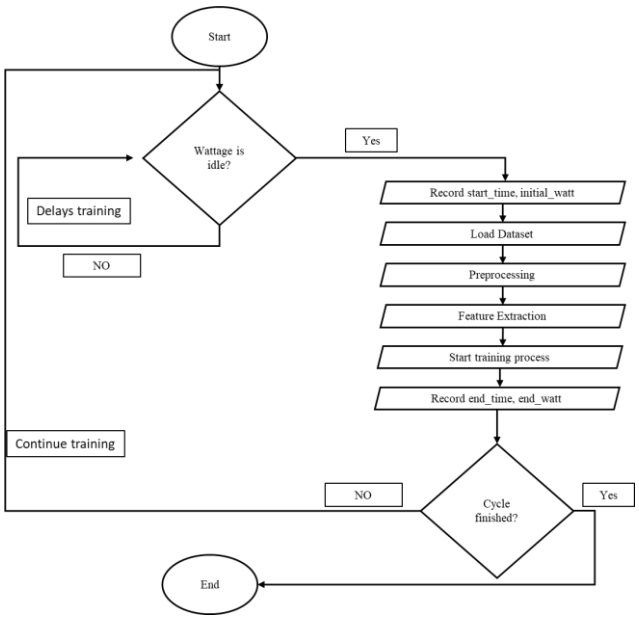

Figure-2. Power and energy measurement flow during the experiment.

e) Class imbalance strategy

To reduce the risk of class imbalance leading to poor model performance, we used imblearn. Random Sampler with the incremental learning mode. This component serves as an interface to classifiers; it trains the underlying model by under sampling and oversampling the received stream such that the effective distribution of classes follows a given distribution. We set this target distribution at 0.5 per class. For batch models, we applied Random under Sampler using a majority class reduction strategy.

5. RESULTS

In order to be statistically reliable, we conducted 20 independent trials. To indicate uncertainty, we provided the mean and the 95% confidence interval (CI95) for every batch.

The CI95 for each metric was calculated using the following formula:

$$CI_{95} = 1.96 \times \frac{\sigma}{\sqrt{n}} \quad (2)$$

where σ is the sample standard deviation, and n=20 is the number of trials.

We have run the experiment for three models in both River incremental learning and learn batch learning, and to get a reliable comparison, we have chosen one model from each category of ML as described in Table-1.

A. AUC Performance Progression

In this section, we will discuss the AUC performance across three models: Naive Bayes (Multinomial NB), Linear Model (Logistic Regression), and Decision Tree (Adaptive Random Forest+Random Forest). We have used the same parameters, set the seed to 42, and processed data the same across both batch and incremental learning.

a) Decision tree

Figure-3 illustrates the AUC progression for the Decision Tree model across incremental and batch learning. For incremental learning, the Adaptive Random Forest was used, while batch learning employed the Random Forest algorithm.

The AUC trend indicates that incremental learning achieves higher performance scores across most batches. Although there is a notable decline in scores at batch 8, the overall performance remains strong. This may be attributed to data-specific characteristics or class imbalance in batch 8. In contrast, batch learning starts with lower AUC values and demonstrates a gradual and consistent improvement, reaching competitive levels by the final batch.

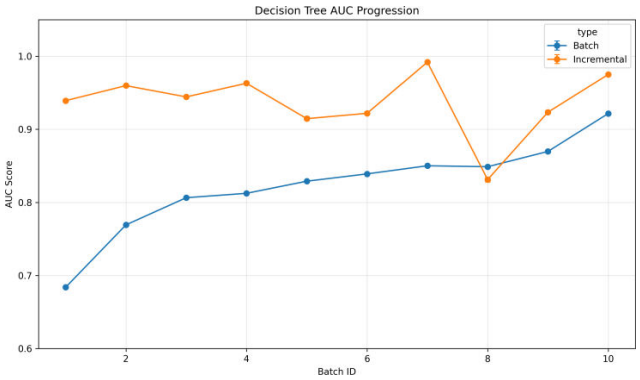

Figure-3. AUC performance progression across batches with 95% confidence intervals.

b) Linear model

Figure-4 illustrates the AUC progression for the Linear Model in incremental and batch learning. Logistic Regression was employed for both batch and incremental learning, as it is available in both libraries. The progress of the model across batch and incremental demonstrates robustness overall. For incremental learning, the model starts with a 71% score and increases gradually until batch 7, when encountering class imbalance issues, and then the model drops approximately 19%. Notably, the model was able to recover later on, reaching its best score yet at batch 10 with an AUC score of 89%. Conversely, batch learning begins with a lower score than incremental learning at 65% and progressively improves to address the class imbalance issue in batch7, concluding at batch 10 with an AUC score of 90%, surpassing incremental learning.

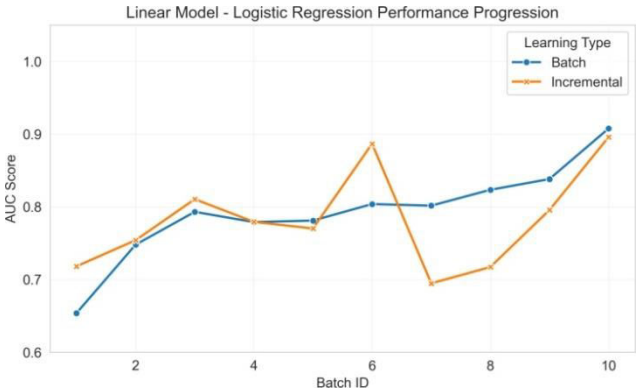

Figure-4. Linear Model AUC performance progression across batches with 95%confidence intervals.

c) Naive bayes

Figure-5 illustrates the AUC progression for the Naive Bayesin incremental and batch learning. Multinomial NB was employed for both batch and incremental learning, as it is available in both libraries.

For incremental, the graph displays a higher AUC score from the outset, attaining 83%, while steadily increasing across batches 2-4. In batch 5, the model experienced a 9% decline in performance but still preserved a strong performance level and promptly recovered in batch 6. This reflects the model's capability to rebound from difficult batches due to the introduction of new features and class imbalance concerns. Notably, unlike other models such as Random Forest and Logistic Regression, when encountering batch 7, a significant performance setback the model sustained a superior score, achieving its peak at 99% before declining to 89% in batch 8. Ultimately, the model conclude sat batch 10 with a final AUC score of 97%, indicating swift adaptability and robustness in the face of class imbalance.

For batch learning, the initial performance was underwhelming compared to incremental, beginning at 65%and progressively improving. The model exhibits a consistent upward trend throughout all batches and does not experience any challenges associated with class imbalance, ultimately achieving an 89% AUC score.

Table-2 illustrates the full AUC evolution of all batches in three analyzed models and in both learning conditions (incremental and batch).

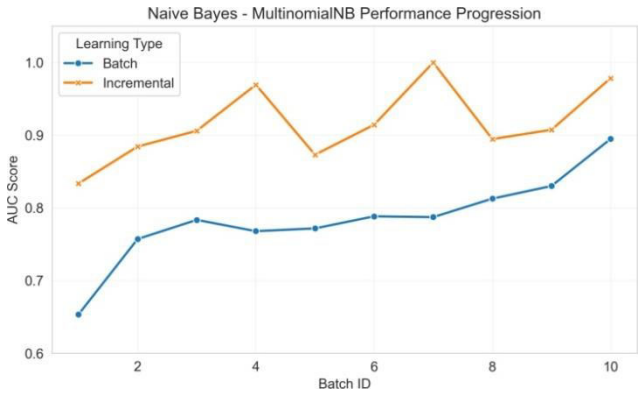

Figure-5. Naive Bayes AUC performance progression across batches with 95% confidence intervals.

Table-2. AUC Scores Across Batches (INC.= Incremental, Batc H= Batch).

Batch	ID Decision Tree Inc.	ID Decision Tree Batch	Linear Inc.	Model Batch	Naive Inc.	Bayes Batch
1	0.939	0.684	0.718	0.654	0.833	0.653
2	0.960	0.769	0.754	0.748	0.884	0.757
3	0.944	0.806	0.811	0.793	0.906	0.783
4	0.963	0.812	0.779	0.779	0.969	0.768
5	0.915	0.829	0.770	0.781	0.873	0.772
6	0.922	0.839	0.887	0.804	0.914	0.788
7	0.992	0.850	0.695	0.802	1.000	0.787
8	0.831	0.849	0.717	0.823	0.895	0.813
9	0.923	0.870	0.796	0.838	0.907	0.830
10	0.975	0.922	0.896	0.908	0.978	0.895

B. Energy Consumption

In this section, we will discuss energy consumption across the three models mentioned earlier and identify trends among batches.

a) Decision tree

Figure-6 illustrates the energy consumption progression for the Decision Tree model across incremental and batch learning. For incremental learning, the Adaptive Random Forest was used, while batch learning employed the Random Forest algorithm.

The system's energy consumption is notably higher under batch learning, increasing from 0.004Wh at the outset to over 0.05Wh by the tenth batch. In contrast, the energy requirement per batch in incremental learning remains relatively consistent, ranging between 0.02 and 0.025Wh. These results suggest that incremental learning is more energy-efficient over prolonged durations. When converted to cost, incremental learning incurs an estimated SAR 0.3894 for all batches, compared to SAR 0.3408 for batch learning, resulting in a cost difference of SAR0.00486 infavor of batch learning.

b) Linear model

In Figure-7, the experiment showed an unexpected turn of events, specifically for incremental learning, starting at batch-1 with a higher consumption than batch learning, consuming 0.003Wh compared to 0.0006Wh, translating to a cost of SAR0.00054 more per batch.

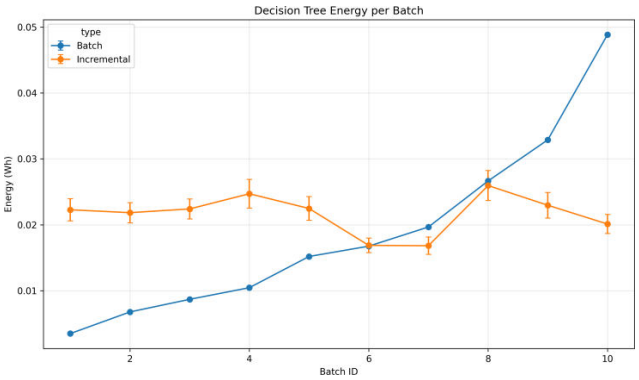

Figure-6. Decision tree energy consumption.

Furthermore, the difference in Watts and price kept climbing incrementally, reaching 0.013 Whatbatch-10compared to batch at 0.004Wh, concluding with a difference in price of SAR 0.00162.

This clearly indicates that the Logistic Regression algorithm is consuming more than the expected result for an incremental learning model.

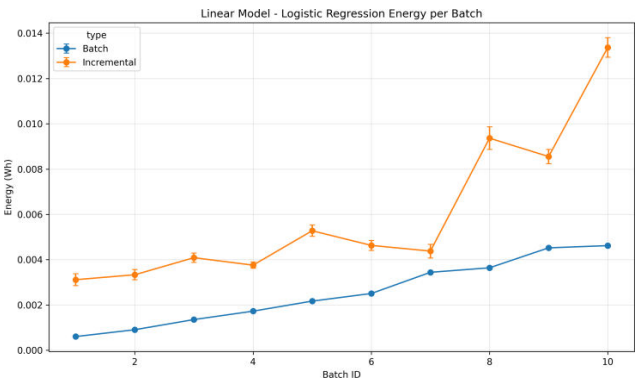

Figure-7. Linear model energy consumption.

c) Naive bayes

In the case of Multinomial NB shown in Figure-8, the incremental learning configuration uses more energy

per batch than the corresponding batch version. It consumes about 0.012Wh per batch, or close to 22 times the 0.00054Wh per batch estimate produced by the batch model, beginning with batch 1. Due to the asymptotic energy consumption of the batch model, though the difference starts to become smaller, at batch 10, the energy consumed by the incremental model still dominates that consumed by the batch by a factor of more than two.

Interestingly, the energy consumed by the incremental model never decreases significantly between batches, indicating a steady, high overhead, most probably as a result of constant updates on the new batch. Conversely, the batch model begins low with a steady rise as it retrain on an incremental size dataset.

This pattern indicates that although the computation cost of Multinomial NB in batch mode is cheap, the fixed cost is high per batch when used in incremental mode. Thus, when energy consumption is of crucial importance, the batch version is much more efficient in this model.

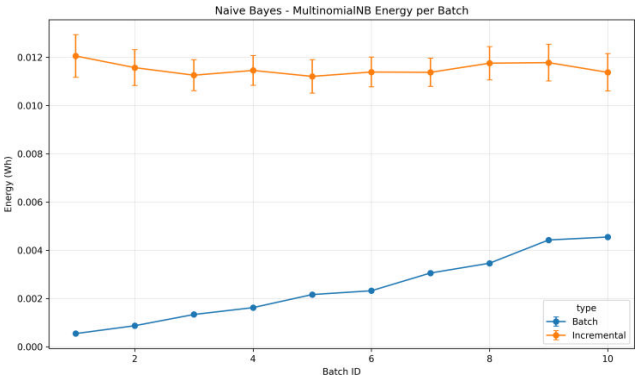

Figure-8. Naive Bayes energy consumption.

Table-3 presents the comprehensive power consumption details across all batches, while Table-4 displays the corresponding costs in SAR.

Table-3. Energy Consumption (Wh) Across Batches.

Batch ID	Decision Inc.	N Tree Batch	Linear Inc.	Model Batch	Naive Inc.	Bayes Batch
1	0.0223	0.0035	0.0031	0.0006	0.0121	0.0005
2	0.0218	0.0068	0.0033	0.0009	0.0116	0.0009
3	0.0224	0.0087	0.0041	0.0014	0.0113	0.0013
4	0.0247	0.0105	0.0038	0.0017	0.0115	0.0016
5	0.0225	0.0152	0.0053	0.0022	0.0112	0.0022
6	0.0169	0.0167	0.0046	0.0025	0.0114	0.0023
7	0.0168	0.0197	0.0044	0.0034	0.0114	0.0031
8	0.0259	0.0266	0.0094	0.0036	0.0118	0.0035
9	0.0230	0.0329	0.0086	0.0045	0.0118	0.0044
10	0.0201	0.0488	0.0134	0.0046	0.0114	0.0045

Table-4. Energy Cost (Sar) Across Batches.

Batch ID	Decision Inc.	N Tree Batch	Linear Inc.	Model Batch	Naive Inc.	Bayes Batch
1	0.0040	0.0006	0.0022	0.0001	0.0022	0.0001
2	0.0039	0.0012	0.0021	0.0002	0.0021	0.0002
3	0.0040	0.0016	0.0020	0.0002	0.0020	0.0002
4	0.0044	0.0019	0.0021	0.0003	0.0021	0.0003
5	0.0040	0.0027	0.0020	0.0004	0.0020	0.0004
6	0.0030	0.0030	0.0020	0.0004	0.0020	0.0004
7	0.0030	0.0035	0.0020	0.0005	0.0020	0.0005
8	0.0047	0.0048	0.0021	0.0006	0.0021	0.0006
9	0.0041	0.0059	0.0021	0.0008	0.0021	0.0008
10	0.0036	0.0088	0.0020	0.0008	0.0020	0.0008

C. Memory Usage

In this section, we will examine memory usage across the three models and evaluate their performance. The variation in the memory consumption observed could be affected by the size of the dataset.

a) Decision tree

Figure-9 illustrates the progression of memory usage across batches. In batch-1, memory consumption patterns between batch and incremental learning differ significantly. For batch learning, memory usage increases steadily in a linear manner, beginning at approximately 195Mb in the first batch and rising to about 214MB by the tenth. In contrast, incremental learning maintains usage close to 210 MB, with a relatively stable memory footprint across batches. This suggests that incremental learning is more efficient in terms of memory utilization.

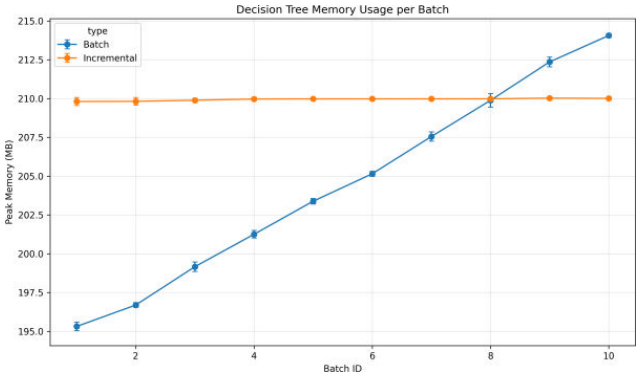

Figure-9. Decision tree memory usage.

b) Linear model

For Logistic Regression in Figure-10, the results exhibited notably better memory utilization than the Decision Tree. Beginning from batch-1, incremental learning demonstrated superior memory efficiency compared to batch learning, maintaining nearly consistent memory usage between 176 MB and 177 MB. In contrast, batch learning displays a steady increase at each batch due to data set accumulation, ultimately reaching 213MB. This indicates that model efficiency can vary significantly even under identical datasets and conditions.

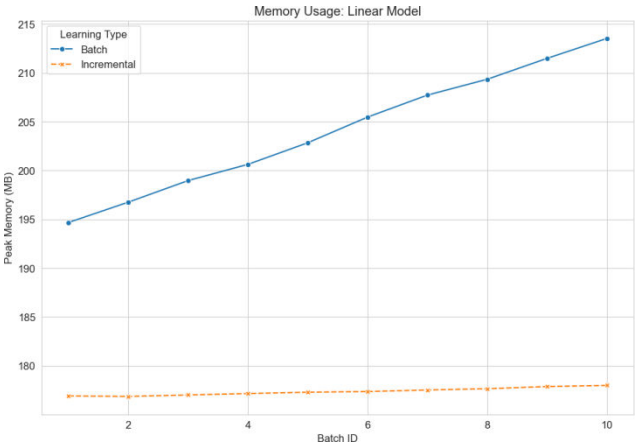

Figure-10. Linear model memory usage.

c) Naive bayes

In Figure-11, the memory usage pattern is comparable to that of the Linear Model. In this context, incremental learning exhibited greater memory efficiency than batch learning, maintaining usage around 189 MB. Conversely, batch displays a consistent increase in memory consumption, beginning at 195 MB and peaking at 212 MB by batch-10.

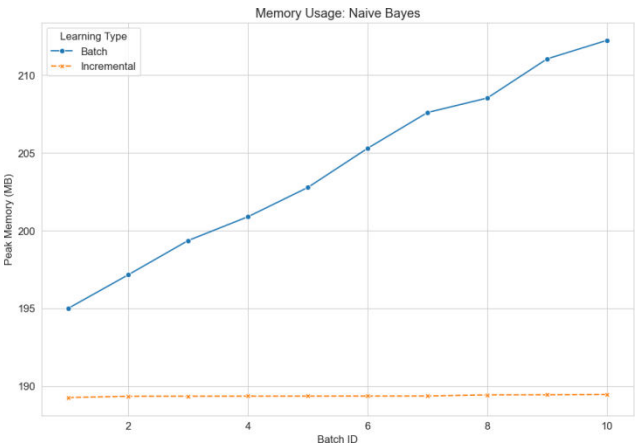

Figure-11. Naive Bayes memory usage.

Table-5 presents the complete memory consumption across all models.

Table-5. Peak Memory Usage (MB) Across Batches.

Batch ID	Decision Tree		Linear Model		Naive Bayes	
	Inc.	Batch	Inc.	Batch	Inc.	Batch
1	209.808	195.310	176.919	194.680	189.262	195.010
2	209.820	196.690	176.868	196.760	189.346	197.160
3	209.889	199.160	177.021	198.970	189.352	199.360
4	209.969	201.250	177.163	200.630	189.354	200.890
5	209.976	203.380	177.302	202.840	189.357	202.770
6	209.977	205.150	177.369	205.470	189.358	205.290
7	209.979	207.550	177.530	207.720	189.360	207.590
8	209.982	209.880	177.660	209.340	189.435	208.520
9	210.024	212.360	177.880	211.480	189.440	211.040
10	210.009	214.060	177.996	213.520	189.460	212.230

D. Processing Time per Batch

In this section, we will examine processing time per batch across the three models and evaluate their performance.

a) Decision tree

Figure-12 presents the duration in seconds required for the models to train on each batch.

The Decision Tree model exhibits a distinct contrast between batch and incremental learning in terms of overall runtime progression. The incremental learning process remains largely consistent in duration across batches, with only minor variations of a few milliseconds. Its maximum training time is approximately 2 seconds. In contrast, the training time for batch learning increases significantly, beginning at under 0.5 second sand rising sharply to over 4.5 seconds by batch 10.

This suggests that, while batch learning may initially appear faster for small datasets, it fails to scale efficiently with increasing batch sizes. The nearly linear trend of incremental learning indicates that it is particularly suitable for scenarios requiring consistent and predictable training durations, such as real-time or online learning environments.

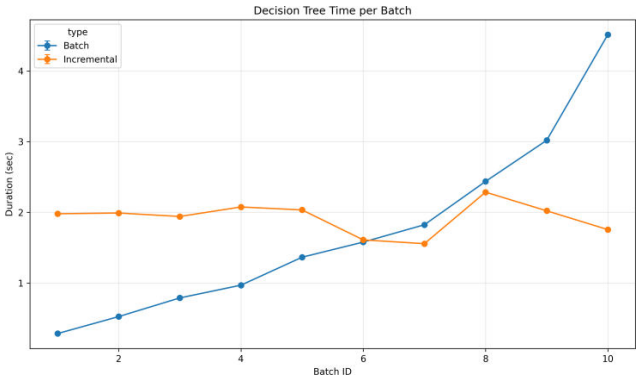

Figure-12. Decision tree processing time.

b) Linear model

For Figure-13, the incremental model requires more time per batch compared to batch learning, beginning at 0.309 seconds for batch-1 versus 0.058 seconds. Unexpectedly, incremental learning continues to increase in processing time per batch, culminating in batch-10 with a significant rise to 1.281 seconds compared to 0.442 seconds for batch. This indicates that the model encounters difficulty processing the batches, possibly due to overhead from repeated partial updates or the accumulation of learned parameters. The total cumulative training time across 10 batches amounts to 5.691seconds for incremental learning and 2.463 seconds for batch, indicating that incremental learning reduced processing time by approximately 7 seconds.

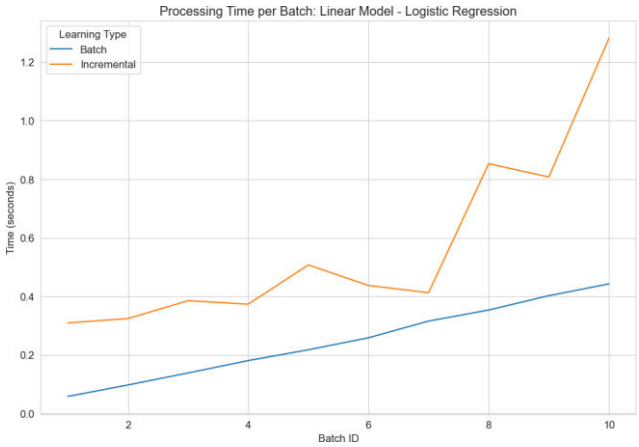

Figure-13. Linear model processing time.

c) Naive bayes

For Naive Bayes in Figure-14, a distinct trend emerges compared to the other two algorithms. The Multinomial NB in incremental learning exhibits a consistent processing time of approximately 1.0 seconds across all batches. In contrast, batch learning displays a

gradual increase, ranging from 0.053 seconds to 0.42 seconds as the dataset expands. The total cumulative training time for incremental learning amounts to 10.43 seconds, compared to 2.37 seconds for batch learning.

This indicates that Multinomial NB behaves predictably and processes batches independently.

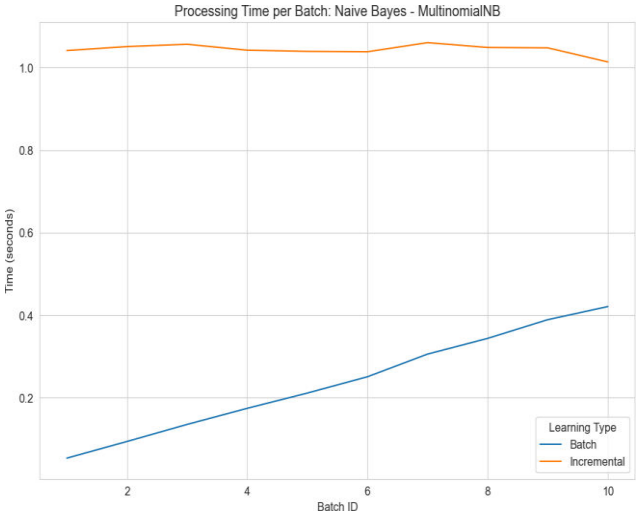

Figure-14. Naive Bayes processing time.

Figure-15 presents the total cumulative training time of all incremental models. This provides a comparison of the models against one another.

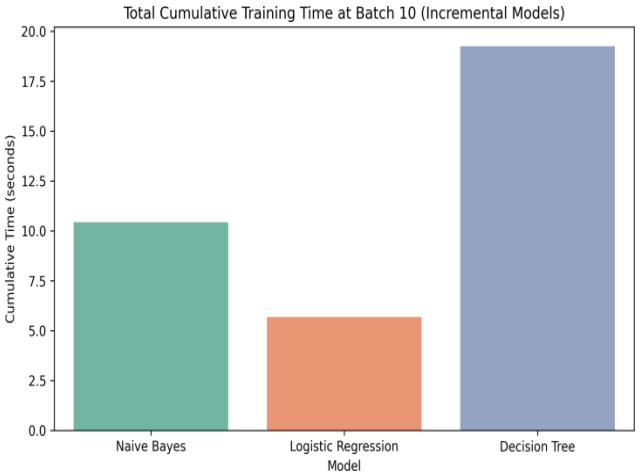

Figure-15. Total cumulative training time for incremental models.

Figure-16 presents the total cumulative training time of all batch models. This provides a comparison of the models against one another.

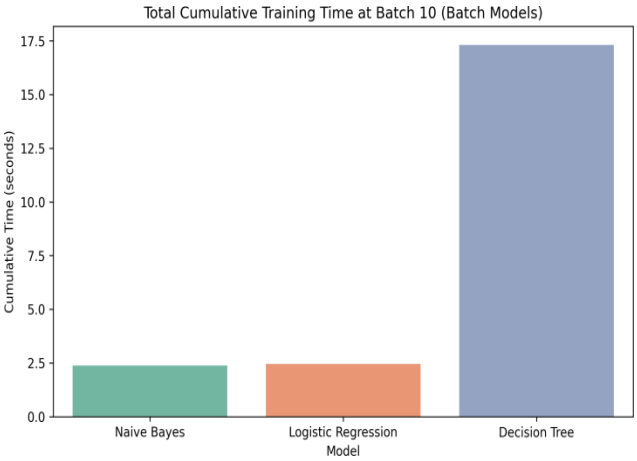

Figure-16. Total cumulative training time for batch models.

6. DISCUSSIONS

One major conclusion from this experiment is that the imbalance of several classes can greatly hurt the accuracy of a model. Because incremental learning is sensitive to small batches, any batch with a single class (e.g., onlylabel0) can drastically damage performance. As such, the fitness of every batch should be examined prior to training to reduce the possibility of model drift.

This is seen in the case of batches 7 and 8: there had been a class imbalance in batch 7. In spite of being equal to batch 8 in size, it lacked label 1, which caused the Random Sampler to over-sample class 1. Quite interestingly, the imbalance boosted AUC scores on some models; however, we do not see any relative boost in memory usage of any model in batch 7, as illustrated in Figure-17, Figure-18 and Figure-19.

The implication is that the class imbalance was not heavy on the computation front. Nonetheless, it continued to affect model behavior and the trajectories of performance, especially by injecting model instability and over-fitting in Naive Bayesor Decision Tree, whereas Logistic Regression achieved more stable generalization.

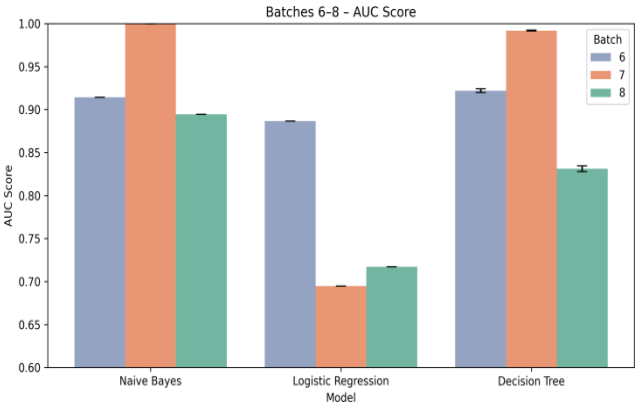

Figure-17. AUC scores for batches 6-8.

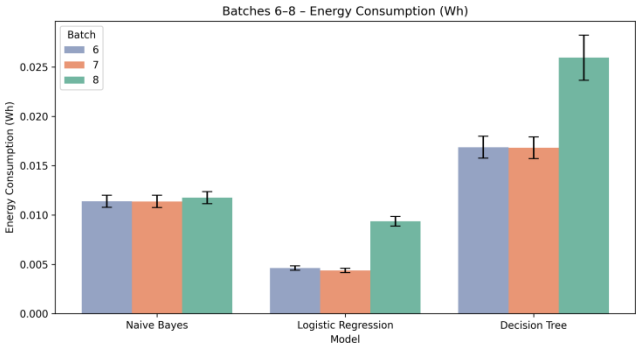

Figure-18. Energy consumption for batches 6-8.

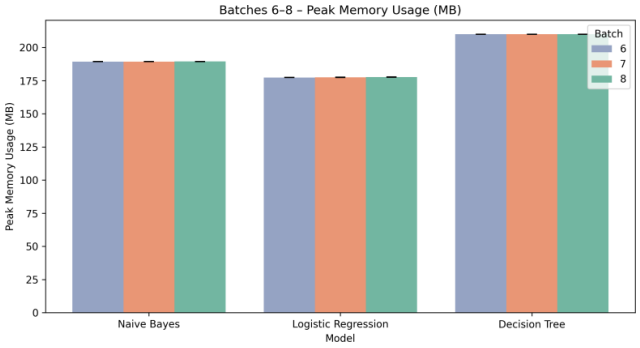

Figure-19. Memory consumption for batches 6-8.

7. CONCLUSIONS

The primary justification for selecting incremental training was its ability to rapidly and efficiently retrain the model from the point of interruption in the training process, whether caused by time constraints, the arrival of new data batches, or the emergence of novel threat vector such has newly identified exploits. However, subsequent analysis of training time and performance metrics revealed that batch quality is the principal factor influencing model performance. In this context, batch quality refers to the class distribution within a given batch. When classes are represented in a balanced manner, training time decreases, and AUC values remain within acceptable ranges.

Although incremental learning may incur a higher initial energy cost, it proves to be more energy-efficient over time, especially in streaming scenarios or prolonged deployment contexts. This advantage enhances its suitability for resource- constrained environments, such as edge devices, where energy efficiency and scalability are essential. These results underscore the importance of incorporating energy considerations into the process of selecting learning methodologies, particularly in applications where operational efficiency and power consumption are critical concerns.

One of the future directions of this work consists of extending the scope of experiments performed to cover a wider range of machine learning models and architectures. Secondly, using the models on real-end IoT devices would allow the observation of realistic values of resources, including battery life, latency, and memory, in real-time scenarios. These assessments would give a better understanding of the suitability of models for edge deployment and help create adaptive energy-aware learning systems.

ACKNOWLEDGMENT

This study partially contributes to the author's thesis research at King Abdul-Aziz University, specifically addressing one of the core research questions.

REFERENCES

[1] J. Montiel, M. Halford, S. Martiello *et al*. 2025. River: Machine learning for streaming data in Python. https://riverml.xyz,2020,accessed:June2025.

[2] J. Read, A. Bifet, B. Pfahringer and G. Holmes. 2012. Batch-incremental versus instance-incremental learning in dynamic and evolving data. In the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD). Springer. pp. 313-327.

[3] N. Ramanath, A. Kasiviswanathan, and S. Prabhakar. 2020. Lambda learner: Fast incremental learning on data streams. arXiv preprint arXiv:2010.05154, 2020. [Online]. Available: https://arxiv.org/abs/2010.05154

[4] K. A. Cahyanto, M. A. Al Hilmi and M. Mustamiin. 2022. Pengujian Rule-Based pada Dataset Log Server Menggunakan Support Vector Machine Berbasis Linear Discriminat Analysis untuk Deteksi Malicious Activity. Jurnal Teknologi In formasi dan Ilmu Komputer, 9(2): 245-254, [Online]. Available: https://jtiik.ub.ac.id/index.php/jtiik/article/view/4107

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vander-plas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research. 12: 2825-2830.

[6] G. Lemaître, F. Nogueira, and C. K. Aridas. 2017. Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning. Journal of Machine Learning Research. 18(17): 1-5.