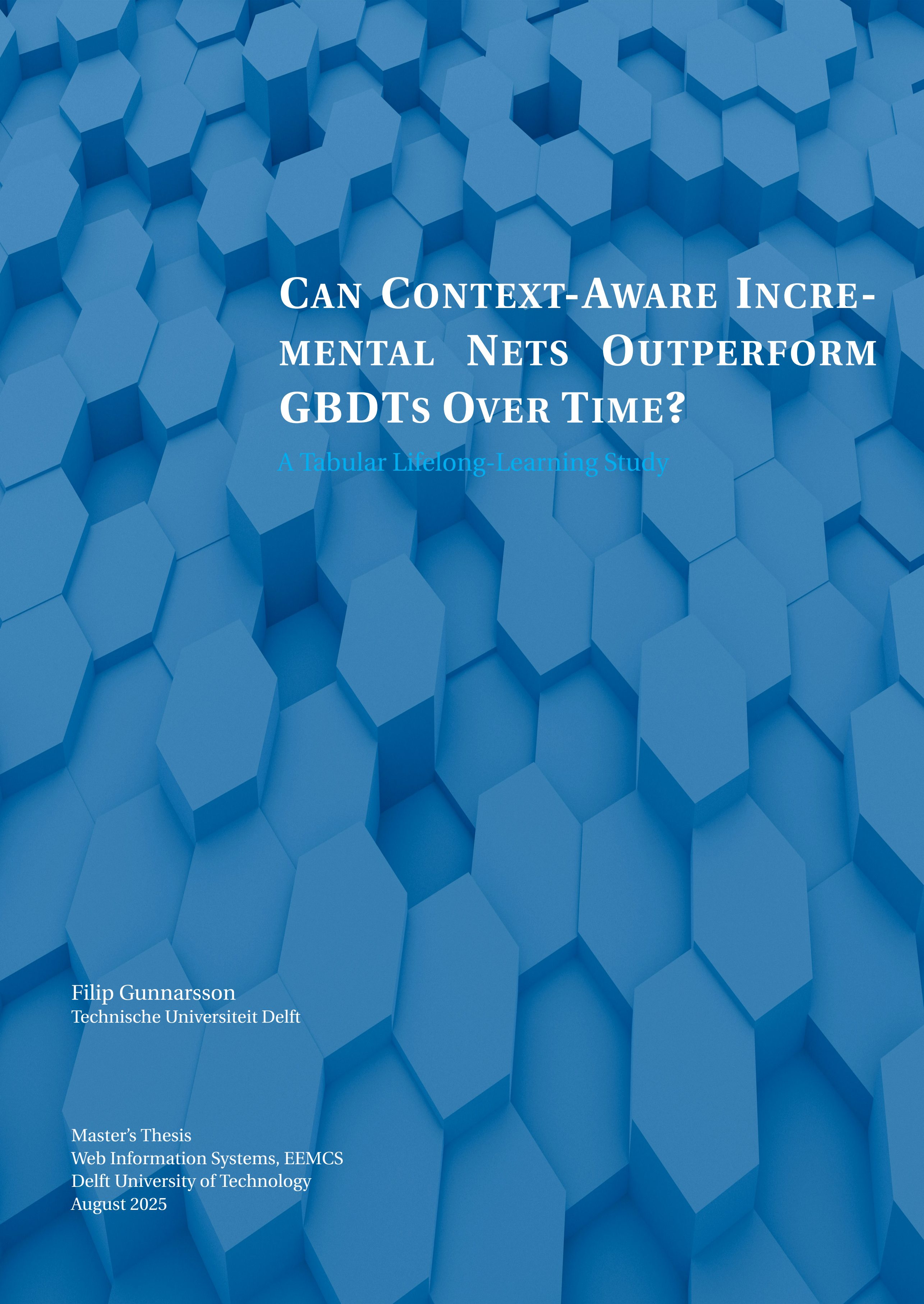


Can Context-Aware Incremental Nets Outperform GBDTs Over Time?

A Tabular Lifelong-Learning Study

Filip Gunnarsson
Technische Universiteit Delft

Master's Thesis
Web Information Systems, EEMCS
Delft University of Technology
August 2025

CAN CONTEXT-AWARE INCREMENTAL NETS OUTPERFORM GBDTS OVER TIME?

A TABULAR LIFELONG-LEARNING STUDY

Thesis

in order to obtain the degree of Master of Science
at Delft University of Technology,
by authority of the Rector Magnificus Prof. Ir. K.C.A.M. Luyben,
chairman of the Board,
to be publicly defended on August 15, 2025

through

Filip GUNNARSSON

With student nr. 5319412
Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, Delft, the Netherlands,

This master's thesis has been approved by the

Thesis advisor: Dr. Sole Pera
Daily supervisor: Dr. Rihan Hai
Daily co-supervisor: Dr. Yuandou Wang
External supervisor: Dr. Fang Fang
External supervisor: Julius Philipp Roeder (DEIC)
Industry supervisor: Steven van Haren (Zanders)

Composition of the graduation committee:

Rector Magnificus,	chair
Dr. Sole Pera,	Delft University of Technology
Dr. Rihan Hai,	Delft University of Technology
Dr. Yuandou Wang,	Delft University of Technology
Dr. Fang Fang,	Delft University of Technology
Dr. Julius Philipp Roeder,	Danish e-Infrastructure Consortium (DEIC)
Mr. Steven van Haren,	Zanders

Contents

List of Figures

List of Tables

SUMMARY

Modern machine learning systems face unprecedented challenges in processing continuously arriving data streams while maintaining both computational efficiency and privacy compliance. Traditional batch learning approaches exhibit quadratic scaling in memory and computational requirements, making them unsuitable for long-term deployment in resource-constrained environments. Despite significant advances in continual learning for computer vision and natural language processing, tabular data represents the majority of industrial machine learning applications.

This thesis introduces IMLP (Incremental MLP), an attention-based architecture for energy-efficient continual learning on tabular data streams. IMLP augments a standard multilayer perceptron with attention-based feature rehearsal, maintaining a fixed-size buffer of learned 256-dimensional representations rather than raw historical samples. This design achieves constant computational complexity regardless of stream length while preserving task-relevant knowledge without storing personally identifiable information.

We conduct comprehensive evaluation across 36 diverse TabZilla classification tasks against 14 baseline methods spanning gradient boosting, classical machine learning, and neural architectures. Using calibrated power measurement equipment and rigorous statistical analysis via Friedman omnibus tests with post-hoc comparisons, we establish that IMLP achieves a 4.2× median speedup and 79.6% energy reduction compared to standard MLPs while maintaining competitive accuracy (80.6% vs 82.9% balanced accuracy).

Our key findings demonstrate that IMLP successfully trades a modest 2.3 percentage point accuracy reduction for substantial efficiency gains, achieving 97.5% of cumulative learning performance using only current segment data. The approach proves robust across datasets spanning 5 to 2,000 features and diverse domains including medical diagnosis, sensor data, and financial applications. Moreover, we introduce NetScore-T, a composite metric for evaluating accuracy-efficiency trade-offs, positioning IMLP optimally on the neural network Pareto frontier.

Therefore, this work establishes the feasibility of practical continual learning for resource-constrained environments while contributing the first systematic study of energy consumption in neural continual learning for tabular data, enabling deployment scenarios previously considered computationally infeasible.

Preface

This thesis began with a decidedly practical point from Zanders, a financial risk firm: energy-efficient GPU computing with a relation to finance. The journey from that initial inquiry to the work presented here took several unexpected turns, each revealing new rabbit holes in the world of machine learning research.

Initially, I was drawn to neural networks for pricing financial derivatives, an intellectually appealing challenge that quickly proved problematic for reasons that became apparent during early exploration. Pivoting to portfolio optimization seemed promising, particularly with reinforcement learning approaches that could account for transaction costs and market complexities. However, I soon faced an uncomfortable realization: if I discovered a genuinely profitable trading strategy, why would I publish it? The academic impulse to share knowledge conflicted directly with the financial incentive to keep such discoveries proprietary. This philosophical dilemma, combined with the daunting prospect of building frameworks from scratch to handle the enormous state-action spaces involved, led me to reconsider my approach entirely.

An encounter with a paper on deep incremental learning for financial tabular datasets [1] opened a new path. The work introduced me to the fascinating world of tabular data analysis and platforms like Numerai, where obfuscated financial data creates a unique research environment. While the paper lacked some clarity, it sparked my interest in the broader landscape of deep learning on tabular data and the ongoing debates between neural approaches and gradient boosting decision trees. Most importantly, I discovered a significant gap: while continual learning had seen substantial advances in computer vision and natural language processing, the tabular domain remained largely unexplored from an energy-efficiency perspective.

What surprised me most during this research was not the technical challenges, though there were many, but rather the frequent lack of reproducible implementations in published work. Papers would reference code that either didn't exist, wouldn't run, or required extensive debugging to function properly. This led to countless hours of reverse-engineering implementations and, I must confess, more than a few heated conversations with my computer screen. The ElmorLabs power measurement hardware added its own complications, initially working only on Windows and requiring significant signal debugging to function properly on Debian, a process that involved more colourful language than I care to document.

Perhaps the most honest reflection I can offer is that while I genuinely love the research process (reading papers, implementing ideas, watching experiments unfold), I have discovered a profound aversion to writing about it. The irony is not lost on me that someone studying incremental learning found the writing process to be decidedly non-incremental, requiring substantial bursts of human computational effort rather than the elegant, efficient updates I was trying to optimize in algorithms.

If there's a broader message I hope readers take from this work, it's the importance of considering efficiency alongside effectiveness. In our pursuit of ever more capable AI systems, we often optimize solely for performance metrics while ignoring computational and environmental costs. As large language models and other AI systems become ubiquitous in daily life, we need to remember that sometimes the best solution isn't the most accurate one. Rather, it's the one that finds the right balance in the corners of the performance-efficiency trade-off space.

To future researchers venturing into similar territory: read and cite my paper, certainly, but more importantly, be prepared to debug other people's code, learn to appreciate the meditative qualities of watching power meters, and remember that talking to your computer is a perfectly normal part of the research process, even if your computer rarely talks back with useful suggestions.

Filip Gunnarsson
Delft, July 2025

1 Introduction

In a world where data streams continuously from sensors, financial transactions, and user interactions, machine learning systems face an unprecedented challenge: how to learn and adapt in real-time without forgetting previously acquired knowledge: Traditional batch learning approaches, which retain models periodically on accumulated historical data, are increasingly inadequate for applications demanding immediate responsiveness, strict privacy compliance, and sustainable computational resource usage. This thesis addresses a critical gap in continual learning research by proposing IMLP (Incremental MLP), a novel architecture that enables energy-efficient continual learning on tabular data streams while maintaining competitive predictive performance and strong privacy guarantees.

1.1. Research motivation

The exponential growth in streaming data applications has created an urgent need for efficient continual learning solutions. Modern systems must process vast volumes of continuously arriving data while adapting to evolving patterns: IoT sensor networks generate terabytes of environmental measurements that require real-time analysis [2], recommendation systems must continuously adapt to changing user preferences [3], and autonomous vehicles need to update their perception models as they encounter new environments [4]. These applications share several critical requirements that expose fundamental limitations in current machine learning approaches.

Traditional continual learning methods face the stability-plasticity dilemma [5], where models must remain stable enough to retain previously learned patterns while maintaining sufficient plasticity to incorporate new information. Most solutions rely on experience replay [6], which stores raw historical samples for periodic rehearsal, or regularization techniques like Elastic Weight Consolidation [7], which constrain weight updates to preserve important parameters. However, these approaches suffer from quadratic scaling in both memory and computational requirements as data streams grow longer, making them unsuitable for long-term deployment.

The energy implications are particularly concerning. Data centers consumed approximately 1.5% of global electricity in 2024 [8], with artificial intelligence workloads driving unprecedented growth that is expected to double this consumption by 2030, with machine learning workloads representing a rapidly growing fraction [9]. Training a single large language model can emit as much carbon dioxide as five cars over their entire lifetimes [9].

Modern data protection regulations impose strict constraints on data retention and processing that further complicate continual learning deployment. The European Union's General Data Protection Regulation (GDPR) [10] establish comprehensive privacy frameworks, with Article 17 of GDPR establishing the "right to be forgotten," requiring organizations to delete personal data upon request [10]. Traditional continual learning methods that store raw historical samples face significant compliance challenges, as they must implement complex deletion mechanisms while preserving model performance.

Healthcare applications exemplify these privacy challenges, where electronic health records contain highly sensitive information that must be protected under regulations like HIPAA [11], yet continual learning on patient data could improve diagnostic accuracy and treatment recommendations. Similarly, financial institutions must balance fraud detection effectiveness with customer privacy protection, often under multiple regulatory frameworks simultaneously.

Despite significant advances in continual learning for computer vision [12] and natural language processing [13], tabular data has received comparatively little attention. This oversight is particularly problematic given that tabular data represents the majority of machine learning applications in industry [14]. Tabular domains present unique challenges including heterogeneous feature types, class imbalance, missing values, and complex feature interactions that differ fundamentally from the homogeneous, high-dimensional data typical in vision and language tasks. Existing tabular learning methods rely heavily on tree-based ensembles like XGBoost [15] and LightGBM [16], as well as neural architectures like TabNet [17] and SAINT [18], all of which require complete dataset access for optimal performance. When adapted to streaming scenarios, these methods typically employ naive approaches like periodic batch retraining, which inherit all the scalability and privacy limitations discussed above.

Current continual learning approaches for tabular data suffer from three fundamental limitations: computational unsustainability due to quadratic scaling with stream length, privacy violations through raw data retention requirements, and deployment infeasibility in resource-constrained environments. This thesis addresses the following research problem: How can we design continual learning architectures for tabular data that achieve competitive predictive performance while maintaining constant computational requirements, preserving privacy through feature-only storage, and enabling deployment in resource-constrained environments?

1.2. OBJECTIVES & RESEARCH QUESTIONS

We investigate energy efficient continual learning through three inter-connected research questions that address the computational, performance, and robustness aspects of the proposed approach.

Research Question 1: Energy Efficiency in Continual Learning

Can IMLP improve energy efficiency in continual learning?

This question directly addresses the computational sustainability challenge by examining whether attention-based feature rehearsal can reduce energy consumption and execution time compared to standard retraining approaches. We hypothesize that maintaining a fixed-size buffer of learned feature representations, rather than complete historical datasets, will enable constant-time updates regardless of stream length.

Measurable Objectives:

- Quantify energy consumption during training and inference using calibrated power measurement equipment
- Measure wall-clock execution time across diverse tabular datasets from established benchmarks
- Establish statistical significance of efficiency improvements through comprehensive comparative analysis
- Demonstrate constant computational complexity by analyzing per-segment energy consumption patterns over streaming scenarios

Research Question 2: Performance-Efficiency Trade-offs

What are the trade-offs between model performance and energy efficiency in a streaming tabular data setting?

Efficiency improvements typically require accuracy sacrifices, but the magnitude and practical implications of these trade-offs remain unclear for tabular continual learning. We investigate whether modest accuracy reductions can be offset by substantial efficiency gains in practical deployment scenarios.

Measurable Objectives:

- Compare multiple performance metrics across diverse tabular classification tasks against established baseline methods
- Introduce a composite metric that jointly evaluates prediction quality and computational efficiency for streaming scenarios
- Analyze accuracy-efficiency trade-offs to identify optimal operating points for different deployment contexts
- Characterize the fundamental differences between segmental and cumulative learning paradigms in terms of data requirements and performance implications

Research Question 3: Cross-Dataset Robustness

How robust are IMLP's results across different tabular datasets?

Tabular data exhibits substantial heterogeneity across domains, feature types, class distributions, and temporal patterns. We examine whether the proposed approach maintains consistent benefits across this diversity or whether performance depends on specific dataset characteristics.

Measurable Objectives:

- Evaluate performance across a comprehensive suite of tabular datasets spanning multiple application domains
- Analyze robustness to varying dataset characteristics including feature dimensionality, class distributions, and dataset sizes
- Identify failure modes and characterize conditions under which the approach excels or encounters limitations
- Establish statistical significance through rigorous hypothesis testing with appropriate multiple comparison corrections

1.3. SCOPE AND LIMITATIONS

We intentionally focus on a specific subset of machine learning challenges to enable deep investigation within manageable bounds. Understanding these scope decisions is essential for properly interpreting our contributions and identifying opportunities for future work.

We restrict our investigation to tabular data, deliberately excluding computer vision, natural language processing, and multimodal applications. This limitation reflects both the underexplored nature of tabular continual learning and the domain-specific challenges that tabular data presents. Unlike images or text, tabular data lacks spatial or sequential structure that enables transfer learning across domains, requires different preprocessing approaches for mixed feature types, and faces unique challenges around class imbalance and missing values.

Our evaluation focuses exclusively on classification tasks, omitting regression, ranking, and structured prediction problems. This choice aligns with the TabZilla benchmark's emphasis on classification while acknowledging that many real-world tabular applications involve regression [19]. The attention-based architecture could potentially extend to regression scenarios, but such investigations remain beyond our current scope.

We simulate streaming scenarios through temporal segmentation of static datasets rather than using truly streaming data with concept drift, distribution shift, or temporal dependencies. This simplification enables controlled evaluation across diverse domains but may not capture all complexities of real-world data streams. Additionally, we assume that class definitions remain stable across segments, which may not hold in evolving problem domains.

Energy measurements are conducted on a controlled laboratory workstation using specialized power monitoring equipment. While this enables precise comparisons, results may not directly translate to diverse deployment environments including different hardware configurations, cloud computing platforms, or edge devices with different power characteristics. Although we discuss privacy benefits of feature-only storage, we do not provide formal privacy guarantees or conduct comprehensive security analyses. Membership inference attacks, model inversion attempts, and other privacy evaluation methods remain outside our scope but represent important directions for future work.

We compare against standard machine learning methods and neural architectures rather than specialized continual learning algorithms like Elastic Weight Consolidation [7] or Gradient Episodic Memory [6]. This choice reflects our emphasis on practical deployment scenarios where simple batch retraining represents the current state-of-practice, but limits our ability to position IMLP within the broader continual learning research landscape.

1.4. CONTRIBUTIONS

We introduce IMLP (Incremental MLP), the first attention-based architecture specifically designed for continual learning on tabular data. Unlike existing approaches that adapt vision or language architectures to tabular domains, IMLP is purpose-built for the heterogeneous, structured nature of tabular data. The architecture maintains a sliding window of learned feature representations rather than raw samples, enabling privacy-preserving continual learning with constant memory requirements.

We conduct the most extensive evaluation of neural continual learning on tabular data to date, encompassing 36 diverse classification tasks against 14 baseline methods. This evaluation introduces rigorous statistical analysis through Friedman omnibus tests and post-hoc pairwise comparisons with Holm correction, establishing statistical significance for all reported efficiency improvements. The breadth of evaluation spans medical diagnosis, sensor data, text classification, and financial applications, demonstrating domain-agnostic effectiveness.

Our work introduces NetScore-T, a composite metric that jointly evaluates predictive performance and computational efficiency for tabular learning scenarios. This metric extends previous work on energy-aware evaluation [20, 21] to streaming tabular data, enabling principled comparison of accuracy-efficiency trade-offs across different algorithms and deployment scenarios.

We provide the first systematic study of energy consumption in neural continual learning for tabular data. Using calibrated power measurement equipment, we quantify the computational sustainability implications of different learning approaches. Our findings reveal that IMLP achieves a 4.2× median speedup and 79.6% energy reduction compared to standard MLPs while maintaining competitive accuracy.

We demonstrate that feature-only storage can enable continual learning without raw data retention, addressing critical privacy and regulatory compliance requirements. While not providing formal privacy guarantees, we show that compressed feature representations contain sufficient information for effective learning while potentially reducing the attack surface for privacy violations.

Through per-segment analysis, we find that IMLP maintains constant computational requirements regardless of stream length, enabling predictable resource planning for long-term deployments. This contrasts with traditional approaches that exhibit linear or quadratic scaling, making them unsuitable for resource-constrained environments. We provide a complete, reproducible implementation of IMLP along with experimental infrastructure for tabular continual learning evaluation, allowing researchers and practitioners to build upon our work and apply IMLP to their specific domains.

The full implementation and experimental framework are available at: `https://github.com/fimgu/IMLP`.

2

Background and Related Work

2.1. Continual Learning

2.1.1. Task, Domain, Class Settings

Continual Learning (CL) refers to the ability of a model to incrementally learn from a sequence of tasks or data distributions without losing performance on previously learned tasks [22]. In a continual-learning setting, data arrive in a non-stationary stream (e.g., changing domains or new classes over time) rather than all at once, and the core challenge is to avoid catastrophic forgetting, the tendency of neural networks to forget old knowledge upon learning new information [22, 5, 7].

Researchers have proposed a taxonomy of three fundamental CL scenarios based on what changes in the stream and what information is available to the learner [22, 23]. These paradigms are known as Task-Incremental Learning (Task-IL), Domain-Incremental Learning (Domain-IL), and Class-Incremental Learning (Class-IL). We describe each in turn and summarize their key differences in Table 2.1. Notably, our work emphasizes the Domain-IL scenario.

Task-Incremental Learning (Task-IL)

Task-Incremental Learning is the scenario in which the agent learns a sequence of distinct tasks with clear boundaries between them [22]. Crucially, the identity of the current task (or context) is known to the model even at test time [22]. This allows the use of task-specific components, for example, a separate output head per task, or even an entirely separate model for each task. Knowing the task label at inference greatly simplifies the problem: the model only needs to solve the active task and cannot confuse it with others.

Indeed, Task-IL is often considered the easiest paradigm, since providing the task identity at test time means catastrophic forgetting can be avoided by design (in the extreme case, one could freeze a dedicated sub-network for each task) [22, 7, 6]. The research challenge in Task-IL is therefore not merely to prevent forgetting, but to learn shared representations or other mechanisms that enable positive transfer between tasks while minimizing computational overhead. Task-IL methods frequently exploit the task label to route inputs or outputs (e.g., using multi-headed networks).

Typical benchmarks: Task-IL can be evaluated on standard class-split benchmarks (e.g., Split MNIST or Split CIFAR-100) by providing a task label to the model during testing (a “multi-head” evaluation) [24]. In such multi-head protocols, an upper bound on Task-IL performance is obtained by an oracle that picks the correct task-specific predictor for each test sample, ensuring no interference between tasks.

Domain-Incremental Learning (Domain-IL)

In Domain-Incremental Learning, the model learns one overarching task across a sequence of changing domains or contexts. Each domain (or “experience”) presents data for the same problem-for instance, the same set of class labels-but drawn from a different distribution or under different conditions [25]. In other words, the input distribution shifts over time while the output space remains constant.

A defining feature of Domain-IL is that the agent is not informed of the domain identity at test time. Unlike Task-IL, the model must handle new domains without an external task label, but since all domains share the same output labels, it isn’t required to identify the domain explicitly to produce an answer. For example, a Domain-IL agent might learn to recognize the same set of objects under progressively different lighting conditions or camera sensors.

This scenario precludes simple “divide-and-conquer” solutions-one cannot deploy a separate network or head per domain at test time unless the model first infers which domain a sample comes from. As a result, catastrophic forgetting is a central challenge in Domain-IL, because the model’s parameters must continually adapt to new contexts while preserving performance on prior domains [26].

Domain-IL has received relatively less attention than class-based incremental learning in recent years, but it is highly relevant to real-world applications such as autonomous systems that face domain shifts (e.g., robots operating in different weather or locations). Researchers have begun to revisit Domain-IL with more realistic benchmarks and dedicated methods.

Classic benchmarks include Permuted MNIST (where each task presents a random pixel permutation of the MNIST images, keeping the label set 0-9 constant) and Rotated MNIST (each task rotates the input images by a fixed angle) [24, 25]. These protocols maintain a common classification objective across domains. More recent Domain-IL benchmarks involve incremental shifts in datasets like CIFAR-10/100 or CORe50 without introducing new classes [27].

Typical benchmarks: Permuted and Rotated MNIST are canonical Domain-IL evaluations. Other examples include continual object recognition under different viewing contexts (e.g., the CORe50 NIC benchmark) and multi-domain image classification datasets (such as DomainNet or incremental shifts in CIFAR-100) where each domain provides the same categories in a new style or environment [27, 28].

Class-Incremental Learning (Class-IL)

Class-Incremental Learning is the most challenging paradigm, where the agent encounters new classes over time and must eventually discriminate among all classes seen so far. Each task in a Class-IL sequence introduces a set of classes that were not present in previous tasks (tasks are often defined by disjoint label sets). At test time, importantly, the model is not told which task a sample belongs to; it must infer the correct class from the union of all classes learned to date. In essence, the model must function as a single classifier that grows its output label space as new classes arrive. This requirement means the learner is implicitly performing task inference: to classify a sample correctly, it must effectively determine which task's classes are relevant (e.g., after learning cats vs. dogs in Task 1 and cows vs. horses in Task 2, distinguish a cow from a cat at test time).

The absence of task labels and the expanding decision space make Class-IL particularly prone to catastrophic forgetting [7]. As new classes are learned, the decision boundaries for older classes often shift, causing accuracy on past classes to plummet. Indeed, without special measures, deep networks struggle to retain old class knowledge when trained on new classes. Class-IL methods therefore devote significant effort to balancing past and new knowledge-for example, by replaying exemplars of old classes or distilling knowledge from previous models [29, 30]. In fact, empirical studies show that simple regularization techniques (e.g., EWC) that suffice in easier settings completely fail on Class-IL [7], whereas rehearsal (experience replay) or similar strategies are needed to reach acceptable performance.

As a result, most state-of-the-art results in Class-IL rely on storing a small set of past examples or generators (despite the strictest Class-IL definitions disallowing any memory) [30, 31]. Typical benchmarks: Class-IL is the default scenario in many image-classification splits, such as Split CIFAR-100 (10 incremental tasks of 10 classes each) and Incremental ImageNet [30]. Performance is measured on unified class prediction across all learned classes (often called single-head evaluation). These benchmarks highlight the severity of forgetting: without countermeasures, a network's accuracy on initial classes drops drastically once later classes are learned.

Paradigm	**Training Labels Available?**	**Test Labels Known?**	**Typical Benchmarks**
Task-IL	Yes (task identity given)	Yes	Split MNIST (multi-head), Split CIFAR-100
Domain-IL	Yes (task identity given)	No	Permuted MNIST, Rotated MNIST, CORe50-NI
Class-IL	Yes (task identity given)	No	Split CIFAR-100, Incremental ImageNet

Table 2.1: Comparison of continual-learning paradigms, highlighting whether task (context) labels are provided to the learner during training and testing, and example benchmarks for each scenario.

2.1.2. STABILITY-PLASTICITY & FORGETTING

Neural networks excel at learning from static datasets, but they struggle to learn continually from evolving data streams without losing past knowledge [12, 32]. When trained on new tasks sequentially, a network's performance on earlier tasks often drops sharply-a phenomenon known as catastrophic forgetting [24]. In other words, learning new information can overwrite or interfere with previously learned representations, causing the model to "forget" what it knew before [12]. This challenge has been recognized for decades in the study of connectionist models [5, 33]. For example, early work by McCloskey & Cohen (1989) [5] and French (1999) [33] analyzed how adding new patterns could catastrophically erase old ones. French (1999) [33] in particular framed this problem in terms of the stability-plasticity dilemma, highlighting a core tension in continual learning: a model must remain plastic enough to acquire new knowledge, yet stable enough to retain old knowledge.

The stability-plasticity dilemma refers to the trade-off between integrating new information and preserving prior knowledge [12]. Too much plasticity (learning too freely) leads to instability and forgetting of old tasks, whereas too much stability (over-conservatism) prevents learning anything new. This dilemma underpins lifelong learning both in biological brains and in neural network models. In cognitive science, complementary learning systems theory posits that the brain mitigates interference by having dual memory systems-a fast-learning buffer (like the hippocampus) and a slow-learning long-term store (the neocortex) [34]. Artificial systems lack such built-in mechanisms, so continual learning remains a long-standing challenge: without special measures, sequential training on non-i.i.d. data generally leads to catastrophic forgetting or interference [24, 7]. A lifelong learning agent is ideally one that can learn from a continuous, non-stationary stream of tasks without significant performance degradation on earlier tasks [32]. Achieving this means striking the right balance between plasticity and stability for each new piece of information.

To concretely quantify forgetting and retention over time, researchers have defined empirical metrics [6]. A common measure is the forgetting rate or drop in accuracy on a given task after learning subsequent tasks [32]. For instance, one can record a model's accuracy on task A immediately after training on A, then measure accuracy on A again after training on tasks B, C, etc.; the difference is a forgetting metric. Low or zero drop means the model retained what it learned; a large drop indicates severe forgetting. Researchers also report the average retained accuracy across all tasks at the end of training as an overall indicator of knowledge retention [22]. Other metrics examine forward transfer (how learning task A helps or hinders learning task B) and backward transfer (influence of new learning on past tasks). In essence, a continual learner should strive for high final performance on all tasks and minimal forgetting-i.e., it should remember old tasks almost as well as if they had been learned in isolation.

2.2. Tabular Data Landscape

2.2.1. Characteristics of Structured Tables

Real-world tabular datasets typically consist of heterogeneous feature types (continuous, categorical, ordinal, etc.) packaged into rows of irregular structure [35, 36]. They often contain missing or sparse entries (e.g., optional fields or one-hot encodings) and can exhibit skewed, heavy-tailed, or class-imbalanced distributions [37]. Unlike images or text, tabular features lack any inherent spatial or sequential ordering: each column is independent, and there are no local patterns to exploit.

In contrast, deep-learning models for vision (e.g., CNNs) or language (e.g., Transformers) assume structured inputs with local correlations or sequence context. Tabular data thus violates those inductive biases, making it harder for neural nets to learn effectively with limited data [35, 37]. These characteristics hinder gradient-based learning: for example, missing values and discrete categories require special encoding or imputation, and non-local dependencies give little advantage to convolution or attention mechanisms.

As a result, tree-ensemble models have historically dominated tabular problems. Practitioners often default to Gradient-Boosted Decision Trees (GBDTs) for tabular tasks [35]. Studies such as Shwartz-Ziv & Armon explicitly note that "traditional ML methods, such as GBDT, still dominate in practice" for tabular data [37], since tree methods can naturally handle heterogeneity and missingness via splits. In summary, the mixed data types, sparsity, and lack of structure make tabular data a challenging domain for neural networks, favoring models like GBDTs that do not rely on input locality or homogeneity.

2.2.2. OpenML and TabZilla Benchmarks

OpenML is an open platform for collaborative machine-learning research [38]. It hosts thousands of tasks and datasets with standardized formats and metadata and provides benchmark suites that bundle related tasks [39].

Examples of these suites include:

- **OpenML-100**: a curated collection of 100 supervised classification tasks (each with 500-50 000 instances) chosen to be broadly usable. These tasks were selected to avoid extreme class imbalance or feature sparsity, though missing values and categorical features are allowed [39].

- **OpenML-CC18**: a curated suite of 72 classification datasets selected from OpenML by Bischl *et al.* (2019) [39]. Each dataset has between 500 and 100 000 instances and $\leq$ 5 000 features, with at least two classes and no class below 5 % frequency. CC-18 enforces additional filters (e.g., minimum class ratios) to ensure balanced, moderate-sized problems suitable for benchmarking.

These OpenML suites standardize the data splits and experimental setup, enabling fair comparisons across algorithms [38, 39].

Beyond OpenML, McElfresh *et al.* (2023) introduced the *TabZilla* benchmark suite to focus on challenging tabular tasks [14]. TabZilla consists of 36 "hard" classification datasets drawn from the OpenML repository. The selection criterion was that no simple baseline (e.g., a single tree stump or 1-NN) achieves best performance, and most algorithms struggle to reach optimal accuracy [14]. The datasets in TabZilla are diverse in domain and scale, ranging from small to very large tasks. TabZilla is accompanied by a unified evaluation protocol: each dataset is evaluated with stratified 10-fold cross-validation, and a broad collection of algorithms (19 models, including multiple neural architectures and GBDTs) are benchmarked under the same settings. The full codebase and raw results (over half a million trained models) are open-sourced to facilitate reproducible research [14].

Together, these benchmark suites (OpenML-100, CC-18, and TabZilla) provide structured collections of tabular tasks for developing and evaluating algorithms. OpenML's curated suites ensure variety and standard splits, while TabZilla emphasizes the most difficult real-world datasets with a rigorous, common evaluation framework.

Other tabular benchmarks exist, such as the Penn Machine Learning Benchmark (PMLB) [40], AutoML benchmark suites [41], and domain-specific collections like UCI [42]. We focus on OpenML-100, CC-18, and TabZilla because they are widely adopted in recent literature, actively maintained, and provide standardized formats, metadata, and reproducible splits through OpenML, aligning with our experimental setup and compute constraints. This targeted selection ensures coherent comparisons without attempting an exhaustive survey.

2.3. TABULAR DATA MODELS

Tabular data (row-column data common in spreadsheets) underpins many practical ML tasks, from finance to healthcare. Historically, gradient-boosted decision trees (GBDTs) (e.g., XGBoost [15], LightGBM [16], CatBoost [43]) have been the dominant models on tabular problems. Deep neural networks (DNNs) excel in other domains but have traditionally lagged on tabular data due to feature heterogeneity and irregular distributions [37].

In recent years, however, a surge of specialized network architectures and benchmarks [35, 17, 18] has revived interest in "tabular deep learning." Key debates include whether NNs can match or surpass GBDTs, and under what conditions. Recent large-scale studies [14] show that on many datasets the NN vs. GBDT gap is small-often, hyperparameter tuning [44] matters more than model choice. However, new methods (e.g., foundation models like TabPFN [45] and FT-Transformer [35]) are challenging this view, particularly in low-data regimes.

2.3.1. NN VS. GBDT LANDSCAPE

GBDTS

Boosted trees (e.g., XGBoost [15], LightGBM [16], CatBoost [43]) form ensembles of decision trees. They handle heterogeneous features, missing values, and skewed distributions naturally. These models require minimal preprocessing and offer straightforward feature-importance measures, but lack end-to-end differentiability. GBDTs remain strong baselines on most tabular tasks.

PLAIN NEURAL NETS

Multilayer perceptrons (MLPs) and ResNet-like MLPs serve as simple NN baselines. Notably, Kadra *et al.* (2021) [44] showed that well-regularized MLPs with a rich "cocktail" of regularization techniques (dropout, weight decay, etc.) can outperform specialized nets and even GBDTs when properly tuned. Gorishniy *et al.* [35] similarly identify a ResNet-like MLP as a strong default. These works suggest that optimization and regularization are crucial-a "plain" NN can match or beat GBDTs with enough tuning.

TREE-INSPIRED NETS (NODE)

Neural Oblivious Decision Ensembles (NODE) [46] replace hard tree splits with soft, differentiable ones. The NODE architecture effectively mimics ensembles of oblivious trees but learns end-to-end by backpropagation. Empirically, NODE models "outperform the competitors on most tasks," often beating traditional GBDTs in benchmarks. This approach bridges the gap by embedding tree logic into a neural net.

ATTENTION-BASED/TRANSFORMER MODELS

- **TabNet** [17] uses sequential attention "decision steps" to select salient features at each layer. This gives inherent interpretability (which features are focused on) and competitive performance. The authors report that TabNet "outperforms other neural and decision-tree variants" on diverse datasets. It also enables unsupervised pretraining for tabular data.

- **AutoInt and TabTransformer** encode categorical features into contextual embeddings via self-attention. AutoInt [47] and TabTransformer [48] demonstrate robust handling of noise and missing data. On 15 public benchmarks, TabTransformer raised AUC by $\geq 1\,\%$ over prior nets and matched tree ensembles, with robust handling of noise and missing data.
- **FT-Transformer** [35] extends transformers by tokenizing both categorical and numerical features. It consistently achieves or exceeds the performance of other DL models across many tasks. In their study, FT-Transformer "performs best on most tasks and becomes a new powerful solution" for tabular data, effectively generalizing to both "ResNet-friendly" and "GBT-friendly" datasets.
- **SAINT** [18] introduces row-wise attention in addition to column attention, plus a contrastive pretraining. It reports state-of-the-art improvements, "even outperforming gradient boosting methods (XGBoost, CatBoost, LightGBM) on average" across benchmarks. This shows that NNs with advanced attention can beat GBDTs in practice.

Foundation Models (Pretrained Transformers)

Inspired by large LLMs, new "foundation" models are trained across many synthetic tabular tasks:

- **TabPFN** [45] is a Transformer trained on millions of synthetic datasets (a Prior-Data Fitted Network). At inference it uses in-context learning to make predictions on entire small datasets instantly. TabPFN "outperforms all previous methods on datasets with up to 10,000 samples by a wide margin," and does so in just a few seconds (5000× speedup over tuned ensembles). It excels in the "small data" regime (under ~ 3000 points).
- Other tabular LLMs and meta-learned models have been proposed (e.g., mixture-of-prompt networks [49, 50, 51, 52]), but TabPFN is the main example so far.

Interpretable NN Architectures

Several works focus on NN explainability:

- **Mesomorphic Networks** [53] produce instance-wise linear models. A deep hypernetwork generates a sparse linear predictor for each input, yielding "free lunch explainability" while retaining deep model accuracy. Experiments show IMNs match black-box nets in accuracy but provide inherent feature-weight insights.
- **TabTransformer embeddings** [48] have been noted as "highly robust and provide better interpretability" because attention weights can highlight important categorical patterns.

The NN landscape for tabular data is broad: it ranges from tuned MLPs and ResNets to hybrid/attention models and large pre-trained networks. Each class tackles heterogeneity and interactions differently, and many recent architectures report strong results on benchmarks.

2.3.2. Benchmark Comparisons

A key question is how these models actually compare on public datasets. Recent large-scale empirical studies provide insights.

TabZilla Benchmark [14]compares 19 algorithms (3 GBDTs, 11 NNs, 5 baselines) across 176 classification datasets. The main conclusion is that the "NN vs. GBDT" debate is often overemphasized: on many datasets, a well-tuned GBDT and a tuned NN yield very similar accuracy. McElfresh *et al.* report that light hyperparameter tuning on a GBDT is often more important than the choice between GBDT and NN. They do find some systematic trends: TabPFN, for small datasets, "outperforms all other algorithms on average" even when sampling 3,000 points [45]. Conversely, GBDTs excel on "irregular" datasets (high skew or heavy tails): CatBoost [43] and XGBoost [15] beat ResNet [35] and SAINT [18] on skewed or heavy-tailed feature distributions. In summary, McElfresh *et al.* recommend practitioners first tune GBDTs, but consider advanced NNs (or ensembles) if needed.

Besides, an independent study [54] compares many methods, including LLM-based models and AutoML. They report that meta-learned foundation models like TabPFN [45] outperform GBDTs in small-data regimes. Purely dataset-specific NNs do better than LLM-based tabular classifiers, but a state-of-the-art AutoML library (heavily using GBDTs) still achieves the best accuracy-at the cost of much higher compute. Likewise, on a collection of about 15 benchmarks, SAINT [18] was shown to "even outperform" GBDTs (XGBoost, CatBoost, LightGBM) on average. This suggests that attention-row mechanisms can give NNs an edge on many tasks.

In a 40-dataset study related to well-tuned MLPs [44], the authors find that an MLP with dataset-specific regularization outperforms prior specialized NNs and also outperforms XGBoost. Their results hint that plain NNs can match GBDTs if optimization is done right. Additionally, Many of the datasets used are also included in the 176 used datasets in the TabZilla paper [14].

In comparisons on diverse tasks, FT-Transformer [35] outperforms other DL models on most tasks. When compared with GBDTs, no single method wins all: they conclude there is "no universally superior solution," echoing earlier studies. However, their ensembling results show that an ensemble of FT-Transformers outperforms an ensemble of GBDTs on nearly all tasks. This implies that, given enough model capacity, NNs can surpass GBDT ensembles.

NODE [46]: In their experiments, NODE "outperforms [GBDT] competitors on most of the tasks," indicating that tree-inspired DNNs close the gap with boosted trees.

Overall, benchmark trends can be summarized as: (1) GBDTs are hard-to-beat baselines, especially on large or irregular data; (2) advanced NNs can exceed GBDTs in certain regimes-notably small datasets (TabPFN, tuned MLPs) and when massive pretraining or ensembling is possible; and (3) there is no single winner-performance is dataset-dependent, leading to the release of the TabZilla "36-hardest-datasets" suite [14] to focus future research on challenging cases.

Table 2.2: Selective comparison of Gradient-Boosted Decision Trees and Neural-Network approaches on tabular data.

Model / Paper (Year)	**Architecture / Type**	**Data / Evaluation Domains**	**Performance vs. GBDTs**	**Interpretability**
XGBoost [15]	GBDT (tree ensembles)	Kaggle, UCI, OpenML benchmarks	Strong baseline; often top performer	Medium (feature importances)
LightGBM [16])	GBDT	Large tabular datasets	Similar to XGBoost; very fast	Medium
CatBoost [43]	GBDT	Categorical data, benchmarks	Top-tier; handles categories well	Medium
Well-Tuned MLP [44]	Multilayer Perceptron (MLP)	40 UCI datasets (small/medium)	Outperforms specialized NNs and XGBoost	Low
TabNet [17]	Attentive sequential NN	20+ diverse tabular tasks	Beats prior DL; comparable to GBDTs	High (sparse feature masks)
TabTransformer [48]	Transformer on categorical features	15 public datasets	>1% AUC gain vs. DL; matches GBDTs	Medium (attention weights)
SAINT [18]	Row + Column attention, pretraining	15+ benchmarks	Outperforms all previous DL; beats GBDTs on average	Low
FT-Transformer [35]	Transformer with feature tokens	50+ OpenML tasks	Outperforms other DL; ensembles beat GBDT	Low
NODE [46]	Neural Oblivious Decision Ensembles	Large benchmark set	Outperforms GBDTs on most tasks	Medium (forest structure)
TabPFN [45]	Pre-trained Transformer	Small-medium datasets (<10k)	Dominant on small data; 5,000× faster inference	Low (black-box)
Mesomorphic NN [53]	Instance-wise linear hypernetwork	Various UCI/OpenML	Matches SOTA accuracy	Very high (linear per instance)
TabZilla [14]	Benchmark suite / analysis	176 classification datasets	GBDT vs NN: negligible difference; TabPFN leads small-data	
Tabular AutoML [54]	Meta-learning / AutoML combo	10+ tabular benchmarks	Best accuracy (GBDT-based); foundation models strong small-data	

2.3.3. Considerations

GBDTs offer natural explainability via feature importances and decision-path analysis. Neural networks are usually opaque, but some architectures supply intrinsic interpretative signals. For instance, TabNet was explicitly designed for interpretability through sparse feature masks [17]. Mesomorphic Networks generate a linear model for each example, yielding built-in explanations [53]. TabTransformer's attention heads can highlight influential categories [48].

Training cost varies widely. GBDTs train quickly on CPUs and seldom need GPUs, whereas many NNs require more epochs and GPU acceleration. McElfresh *et al.* report inference speeds: TabPFN processes ~0.25s per 1,000 instances [14, 45], while CatBoost takes ~21s for the same batch [43]. FT-Transformer and SAINT can be expensive to train: Gorishniy *et al.* note FT-Transformer's high resource usage and associated CO_2 concerns [35]. Advanced tuning (e.g., hyperparameter search) greatly affects time: Kadra *et al.* emphasize that well-tuned simple networks are competitive, implying significant tuning cost [44].

GBDTs are easy to deploy in lightweight environments (often exportable to minimal runtime code) and handle smaller data well. Neural networks, especially large transformers require support libraries and may be overkill for simple tasks. However, NNs integrate seamlessly into end-to-end pipelines and support incremental or transfer learning (e.g., TabPFN can be fine-tuned on new data). Neural models can also leverage hardware accelerators for faster inference once deployed. Practitioners must weigh model complexity against real-time constraints: a single-pass TabPFN model delivers predictions extremely quickly [45], whereas a heavily tuned GBDT ensemble may demand substantial compute.

The current state of the art shows no universally superior method for tabular data. GBDTs remain strong out-of-the-box and are often the safest choice [37], especially on large, irregular, or skewed datasets [15, 43]. Deep networks, however, have closed much of the gap through new architectures and training strategies. In particular: (a) attention-based and Transformer models such as FT-Transformer and SAINT have become competitive or better on many benchmarks [35, 18], (b) regularization-heavy MLPs can rival GBDTs when tuned properly [44], and (c) foundation models like TabPFN show that large, scale pretraining can yield a tabular GPT for small data [45]. Meta-analyses suggest starting with a well-tuned GBDT and then exploring specialized NNs if warranted [14]. The release of the TabZilla benchmark encourages testing on difficult cases (the hard 36) where new methods might shine [14]. Future work may push toward integrating domain knowledge into NNs, better self-supervised learning for tables, and more efficient tabular transformers. Crucially, improving interpretability in deep models, as in Mesomorphic Networks will make them more attractive [53].

The GBDT vs. NN debate is nuanced: boosted trees are robust baselines that excel on many tabular problems [37], but modern neural approaches especially those leveraging attention or pretraining, can match or exceed GBDT performance in key scenarios [35, 45, 18]. The optimal choice ultimately depends on data size, feature properties, and practical constraints.

2.4. FORGETTING MITIGATION TECHNIQUES

Continual learning models must learn new tasks without overwriting old knowledge. Regularisation-based methods constrain updates to important parameters, replay methods re-introduce past data (real or synthetic), and memory-based methods maintain compact representations of past tasks (e.g., prototypes, prompts, key-value memory). Transformer-style attention (keys/queries) and neural-process memories have recently been adapted to continual learning to retrieve relevant past information. We review these approaches by category, citing key examples across vision, NLP, RL, and especially tabular domains.

2.4.1. REGULARISATION-BASED APPROACHES

These methods add penalties to the loss to slow learning on important parameters. For example, Elastic Weight Consolidation (EWC) [7] estimates a Fisher information matrix F_i for each weight and adds a quadratic penalty:

$$L(\theta) = L_{\text{new}}(\theta) + \frac{\lambda}{2}\sum_i F_i(\theta_i - \theta_i^*)^2 \tag{2.1}$$

where θ^* are old-task parameters [7]. Similarly, Synaptic Intelligence (SI) [25] accumulates an importance measure for each parameter over the learning trajectory, and penalizes changes to "important" synapses. Memory Aware Synapses (MAS) [55] computes parameter importance in an unsupervised, online fashion based on output sensitivity, then constrains those weights. Learning without Forgetting (LwF) [56] uses knowledge distillation to match the old model's soft outputs on new data, effectively regularizing the output function. These methods ensure that new gradients g_i are scaled by parameter importance (high F_i or MAS score), so critical weights change little. In practice they reduce forgetting at the cost of some plasticity. Progressive Neural Networks [57] avoid forgetting by freezing previous networks entirely and adding new "columns" with lateral connections. While not a pure penalty method, progressive nets exemplify an architectural solution: old tasks' weights remain fixed, so their output is preserved [57].

In EWC, the regularized loss L combines the new-task loss with a Fisher-weighted ℓ_2 penalty as above. In multi-task MLPs, one might also add output-distillation: $L = L_{\text{new}} + \lambda_{\text{KD}}, \mathbb{E}x, [D\text{KL}(p_{\text{old}}(x), |, p_{\text{new}}(x))]$ (as in LwF) [56]. Examples would be: EWC and SI (vision: permuted MNIST, Split CIFAR) [7, 25], MAS (vision, object recognition) [55], LwF (vision: incremental ImageNet) [56]. These methods typically use image datasets, but the same principles apply to tabular MLPs or transformer weights in NLP.

2.4.2. BUFFER REPLAY STRATEGIES

Replay (or rehearsal) methods interleave new-task training with samples drawn from a memory of past data. A simple episodic memory holds a subset of past data; new minibatches sample from this buffer to "remind" the model of old tasks. Reservoir sampling ensures a representative buffer when tasks arrive streaming. For example, *iCaRL* [30] stores exemplars for each class and classifies by nearest-mean-of-exemplars. It combines this with knowledge distillation (LwF) [56] to update its feature extractor, and uses "herding" to select exemplars. Specifically, each class c is represented by a prototype

$$\mu_c = \frac{1}{|P_c|} \sum_{x \in P_c} \phi(x),$$

and a new sample is classified via

$$c^* = \arg\min_c \left\| \phi(x) - \mu_c \right\|^2, \tag{2.2}$$

where $\phi(x)$ is its feature embedding. This nearest-mean-of-exemplars rule decouples classification from network outputs, aiding stability. Experience Replay [58] simply buffers raw examples from all prior tasks and interleaves them with new data. In RL domains (Atari, DMLab), Rolnick *et al.* [58] showed that even a small replay buffer dramatically reduces forgetting, matching methods that require task labels. Other variants include Gradient Episodic Memory (GEM) [6], which constrains new-task gradients to not increase loss on buffered data (via a quadratic programming step), and A-GEM, its averaged form. Buffer replay can be extended to latent features (see next subsection).

A fixed-size buffer (often per-class) is sampled along with new task data. When the buffer is limited, one discards old samples (randomly or by score) once full. Experience replay is conceptually simple and often outperforms more complex alternatives [58]. Examples would be: iCaRL (vision; CIFAR-100, ImageNet) [30], GEM (vision; Permuted MNIST) [6], raw-replay (RL; Atari, DMLab) [58].

2.4.3. GENERATIVE REPLAY METHODS

Generative replay trains a generative model (e.g., GAN or VAE) alongside the solver so that old-task samples can be synthesised on-the-fly. Shin *et al.* introduced *Deep Generative Replay* (DGR) [59], wherein a generator G learns to approximate the distribution of past tasks while the solver network is trained on a mixture of real and generated data. At each new-task step the training objective is

$$L = L_{\text{new}}(x_{\text{new}}) + L_{\text{replay}}(x_{\text{gen}}), \qquad x_{\text{gen}} \sim G, \tag{2.3}$$

so the learner never needs to store the original data explicitly.

Brain-inspired dual-memory systems adopt a similar idea: Kamra *et al.* [60] proposed a fast "hippocampal" network for recent experiences and a slow "neocortical" network that consolidates knowledge via generative replay. *FearNet* [61] likewise couples a VAE-based generator with short- and long-term memory modules, during "sleep" phases the model *dreams* synthetic examples that transfer recent knowledge into stable weights, achieving near iCaRL performance on CIFAR-100 and audio benchmarks without storing real exemplars.

The model learns a generator that preserves the old data distribution so that synthetic samples can be replayed in place of raw stored data. A few examples would be: DGR (vision; Permuted MNIST, sequential CIFAR) [59], Deep Generative Dual Memory (vision; CIFAR-10) [60], FearNet (vision/audio; CIFAR-100, CUB-200, AudioSet) [61]. Although demonstrated mainly on vision streams, generative replay could in principle model tabular or text data as long as an appropriate generator is available.

2.4.4. FEATURE AND LATENT REPLAY

Instead of replaying raw inputs, *feature/latent replay* stores internal activations and feeds them back into the network during later tasks. Let $f_{1:k}$ denote the sub-network up to layer k and $f_{k+1:n}$ the remainder. For a selected subset of samples we cache

$$h^{(k)} = f_{1:k}(x),$$

freeze the lower layers, and during new-task training optimise

$$L = \ell\big(f_\theta(x_{\text{new}}), y_{\text{new}}\big) + \ell\big(f_{k+1:n}(h^{(k)}_{\text{old}}), y_{\text{old}}\big), \tag{2.4}$$

where the second term replays stored activations $h^{(k)}_{\text{old}}$ as if they were fresh inputs. Since the early layers are fixed, gradients do not alter $f_{1:k}$, reducing interference and memory (no need to save high-dimensional images).

Pellegrini *et al.* [62] show that latent replay on MobileNet shrinks memory by an order of magnitude while preserving accuracy on CORe50 and NICv2. The general idea is to store activations $h = f_{1:k}(x)$ rather than raw x. Mixing new raw samples with stored h is equivalent to sampling "latent prototypes," which mitigates gradient conflict.

2.4.5. PROTOTYPE AND MEMORY MODELS

These methods explicitly build a compact memory or prototype for each concept. For classification, prototypical replay maintains class centroids (as in iCaRL). Prototype networks [63] compute class means in embedding space; this idea transfers to CL by rehearsing prototypes or using them as memory. In tabular data, cluster-based models like XuILVQ [64] perform online vector-quantisation: they maintain prototypes that get updated with each sample, and can generate synthetic points from those prototypes.

For TRIL3 [65] on tabular data, XuILVQ [64] generates pseudo-samples of past classes for a downstream classifier (a Deep Neural Decision Forest). Another memory-augmented approach is to use key-value stores: e.g. associate each past input x with a label vector, and retrieve via similarity in a learned embedding.

2.5. ATTENTION-BASED RETRIEVAL MECHANISMS

Recent CL methods leverage attention or prompts to retrieve relevant old-task information on demand. Transformer-style **key-query attention** is at the core: given query vectors Q (from the current input) and key vectors K (memorised for each past context), one computes attention weights by the scaled dot-product; the resulting weights retrieve value vectors V that encode stored knowledge.

2.5.1. KEY-QUERY MATCHING MODELS

The scaled dot-product attention of Vaswani *et al.* [66] is

$$A = \operatorname{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right), \qquad \operatorname{Att}(Q,K,V) = AV, \tag{2.5}$$

where d_k is the key dimension. In continual learning, K and V constitute a task memory; the query Q derived from a new input retrieves the most relevant past contexts via A. This is the basis of our implementation in 3.

2.5.2. PROMPT POOL ADAPTATION

In prompt-based CL, the memory comprises a pool of learned prompts. Learning to Prompt (L2P) [49] maintains key-value pairs: each prompt has a key and a value embedding. Given an input, its query selects a subset of prompts through attention; these prompts are prepended to the tokens and fed to a frozen pre-trained backbone. This allows re-using a large model for many tasks without changing its weights. DualPrompt [50] extends this by learning two disjoint prompt sets (general and expert), still selected by query-key attention. CODA-Prompt [51] further enables *concurrent* activation of multiple prompts via decomposed attention, achieving state-of-the-art on Split ImageNet-R and Split CIFAR-100. Most recently, KOPPA [52] adds a MAML-inspired orthogonality constraint so that new-task keys become approximately orthogonal to old-task queries, reducing representation drift.

2.5.3. NEURAL PROCESS MEMORY

Neural Process Continual Learning (NPCL) [67] treats each task as a stochastic function and assigns it a latent variable. A hierarchical latent model learns posteriors over these variables; during new tasks, the model regularises the latent distributions so that uncertainty-aware attention to past latent memories mitigates forgetting. Unlike prompt pools, NPCL retrieves *task latents* rather than explicit prompts but relies on the same key-query matching principle.

2.5.4. COMPARATIVE SUMMARY

Each entry shows the key mitigation strategy (e.g. Regularisation, Buffer Replay, Generative Replay, Feature/Latent Replay, Prototype/Memory, Attention/Prompt) alongside the

Table 2.3: Comparative summary of foundational and state-of-the-art continual-learning models, showing their application domain, benchmark datasets, and principal forgetting-mitigation strategy.

Model (Ref)	**Paper Title**	**Domain**	**Datasets**	**Mitigation Type**
Progressive Nets [57]	Progressive Neural Networks	RL	Atari / 3D Maze	Dynamic architecture (frozen columns)
EWC [7]	Overcoming Catastrophic Forgetting in Neural Networks	Vision, NLP	Permuted MNIST; Split CIFAR	Weight regularisation (Fisher)
SI [25]	Synaptic Intelligence	Vision	Permuted MNIST	Weight regularisation (path integral)
MAS [55]	Memory Aware Synapses	Vision	Pascal VOC	Weight importance (unsupervised)
LwF [56]	Learning without Forgetting	Vision	Incremental ImageNet	Knowledge-distillation regularisation
iCaRL [30]	Incremental Classifier and Representation Learning	Vision	CIFAR-100; ImageNet	Buffer replay (exemplar prototypes)
GEM [6]	Gradient Episodic Memory	Vision	Permuted MNIST	Buffer replay with gradient constraints
Experience Replay [58]	Experience Replay for Continual Learning	RL	Atari; DMLab	Buffer replay (reservoir)
DGR [59]	Deep Generative Replay	Vision	Sequential MNIST; Sequential CIFAR	Generative replay (GAN)
Dual Memory [60]	Deep Generative Dual Memory	Vision	CIFAR-10	Generative replay (dual system)
FearNet [61]	FearNet: Brain-Inspired Model for Incremental Learning	Vision/Audio	CIFAR-100; CUB-200; AudioSet	Generative replay (VAE + dual memory)
Latent Replay [62]	Latent Replay for Real-Time Continual Learning	Vision	CORe50; NICv2	Feature/latent replay (activations)
L2P [49]	Learning to Prompt for Continual Learning	Vision	Split CIFAR-100; CORe50; DomainNet	Key-query prompt retrieval
DualPrompt [50]	DualPrompt: Complementary Prompting for Rehearsal-Free CL	Vision	Split ImageNet-R; Split CIFAR-100	Key-query prompt retrieval (dual prompts)
CODA-Prompt [51]	COntinual Decomposed Attention-based Prompting	Vision	ImageNet-R; Split CIFAR	Key-query prompt retrieval (concurrent)
KOPPA [52]	Key-Query Orthogonal Projection & Prompt Alignment	Vision	Split ImageNet-R	Key-query retrieval (orthogonal constraint)
NPCL [67]	Neural Processes for Uncertainty-Aware Continual Learning	Multi-domain	Vision & RL benchmarks	Latent memory (Neural Processes)
XuILVQ + DNDF [65]	TRIL3: Pseudorehearsal for Tabular Data	Tabular	UCI; CICIDS; Health records	Prototype-based replay (latent gen.)
Transformer [66]	Attention Is All You Need	NLP/Vision	WMT'14 En-De / En-Fr	Self-attention (key-query)

domains and datasets where it was demonstrated. In practice, many methods combine categories (e.g. iCaRL uses replay + distillation). The table and above survey highlight how foundational ideas like weight penalties replay buffers, generative models, prototypes, and attention can be adapted across image, text, RL, and tabular tasks to combat catastrophic forgetting.

2.6. ENERGY-AWARE MACHINE LEARNING

The Green AI movement argues that energy efficiency must become a first-class goal in AI research. Modern deep learning has grown explosively, for example, publication counts for deep learning rose from ~1,350 papers in 2015 to over 85,000 in 2022 [68], and recent analyses warn that this surge entails a heavy environmental toll [68]. Training large models can emit on the order of hundreds of tonnes of CO_2, vastly exceeding the ~2 tonnes per-person per-year budget needed for 1.5 °C climate targets [68]. Henderson *et al.* [69] emphasize that "accurate reporting of energy and carbon usage is essential for understanding ML's climate impacts," introducing tools and leaderboards to spur low-carbon research. Likewise, Różycki *et al.* [70] note that the carbon footprint of ML spans both operational energy and the embodied emissions in hardware: in many cases the latter actually eclipse the former. Together, these studies motivate energy-aware ML: researchers are now encouraged to consider efficiency and carbon cost alongside accuracy. Trinci *et al.* [21] remark that in continual learning - a subfield of adapting foundation models - "the environmental sustainability remains relatively uncharted," and they undertake to fill that gap by measuring energy use across methods. These Green AI efforts underscore that motivation is now shifting: beyond accuracy, AI researchers must also target lower wattage and carbon.

2.6.1. HARDWARE POWER INSTRUMENTATION

Accurately measuring power draw in hardware is the foundation of energy-aware ML. At the device level, modern CPUs and GPUs include on-chip sensors and counters: Intel processors expose the RAPL (Running Average Power Limit) interface, which provides real-time power readings for CPU cores and DRAM; NVIDIA's NVML (via `nvidia-smi`) reports instantaneous power draw, utilization, and temperature for each GPU. These built-in channels (and their software wrappers such as PyRAPL) allow fine-grained tracking of component power. In contrast, external power meters (e.g. smart PDUs or "Kill-A-Watt" devices) measure whole-node draw from the wall socket, capturing CPUs, GPUs, memory, storage, and cooling overhead. In data centers, one must also account for cooling and facility losses via the Power Usage Effectiveness (PUE) factor. Industry benchmarking efforts further standardize instrumentation: the MLPerf Power benchmark [71] provides a unified methodology for measuring ML workload power across scales. Latif *et al.* [72] instrumented an 8×NVIDIA H100 training node and found its actual peak power (~8.4 kW) was about 18% below the rated 10.2 kW, and showed that increasing ResNet batch size from 512 to 4096 images reduced total training energy by a factor of four.

2.6.2. SOFTWARE ENERGY PROFILERS

Complementary to hardware instrumentation are software tools that estimate or log energy use within ML code. Bouza *et al.* [68] review seven popular tools (Table 2.4): Code-Carbon and Eco2AI sample CPU/GPU load every 10-15 s to estimate kWh and CO_2 for a training run; CarbonTracker, Experiment-Impact-Tracker, and $MLCO_2$ hook into sensors to report per-job energy; Green-Algorithms is a web-based calculator using job parameters to provide a rough CO_2 estimate; and Cumulator aggregates measured power via RAPL/NVML into kWh. These tools differ in scope: some track only the ML process, others include full-node idle costs. Bouza *et al.* [68] experimentally found as much as a two-fold discrepancy between tools, attributable to these differences. In addition, general-purpose profilers (e.g. Linux `perf`, PowerTOP) can hint at hot spots, and frameworks like TensorFlow and PyTorch now include basic energy reporting via underlying RAPL/NVML hooks.

2.6.3. JOINT ACCURACY-ENERGY METRICS

To evaluate models holistically, the community is moving beyond accuracy alone. Trinci *et al.* [21] introduce Energy NetScore, a task-averaged score that penalizes both high energy and low accuracy in continual learning models. Yang *et al.* [79] similarly propose an "energy efficiency" score for vision foundation models and provide an interactive tool to compare models by accuracy vs. energy. These metrics typically weight accuracy (reward) against energy (cost) with tunable parameters, producing a single value for ranking. They reveal steep trade-offs: Yang *et al.* [79] find that accuracy improvements in large ImageNet models now require exponentially more energy, yielding rapidly diminishing returns. Such findings underscore that small accuracy gains may not justify

Table 2.4: Examples of software energy trackers for machine learning [68].

Tool	Approach	Notes
Green-Algorithms [73]	Web calculator	Estimates CO_2 from model type and run-time (no live sensing).
CodeCarbon [74]	Python library	Monitors process or machine power (logs CPU/GPU usage every ~15 s).
CarbonTracker [75]	Python library	Tracks GPU power draw (and optionally infers CO_2).
Experiment-Impact-Tracker [69]	Python library	Monitors CPU/GPU usage via perf counters (real-time).
Eco2AI [76]	Python library	Similar to CodeCarbon (monitors usage per process or host).
MLCO$_2$ [77]	Web/CLI tool	Provides quick carbon estimates (global ML emissions database).
Cumulator [78]	Python library	Aggregates measured power (via RAPL/NVML) into kWh during execution.

huge energy increases. MLPerf Power also reports energy-to-solution alongside throughput, penalizing models that consume excessive energy for a given performance. Overall, modern ML evaluation increasingly penalizes wasteful computation, pairing every accuracy figure with its energy cost to guide sustainable model selection.

2.7. RESEARCH GAP

Despite rapid progress in both tabular deep learning and CL, the intersection of these areas remains largely unexplored.

1. **Lack of Domain-Incremental Benchmarks for Tabular Data.** Current CL studies overwhelmingly target image streams in the Task-IL or Class-IL settings. Real-world tables, however, experience *domain drift* (e.g., quarterly finance data, evolving sensor logs) without label-space changes, yet no standard Domain-IL benchmark exists for such data.

2. **Energy Footprint is Ignored.** Retraining GBDTs or large neural nets on every data refresh is compute- and carbon-intensive, but CL papers rarely report power or CO_2, especially for tabular workloads.

3. **Gap Between Accuracy and Efficiency Metrics.** The community still optimises for predictive metrics alone (accuracy, AUC, log-loss). There is no widely-adopted metric that *jointly* rewards predictive quality and joule efficiency across a non-stationary stream.

4. **Foundation Models and Privacy-Aware Replay Remain Untested.** Pre-trained transformers for tables (e.g., TabPFN, FT-Transformer) and feature-level or attention-based replay strategies promise low-storage, privacy-preserving CL, but have not been benchmarked under domain drift.

3 Methodology

3.1. Problem Statement

To bridge these gaps, we consider tabular continual learning under tight memory constraints. The data arrive as an ordered list of chunks:

$$\mathscr{D} = \{\mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_T\},$$

where each chunk

$$\mathscr{S}_t = \{(x_i, y_i)\}_{i=1}^{n_t}$$

contains n_t examples collected at time t. Inputs are real-valued feature vectors $x_i \in \mathbb{R}^{d_{\text{in}}}$, and labels are categorical $y_i \in \{1, \ldots, C\}$. Chunk lengths, class proportions, and feature distributions may vary, and T (the total number of chunks) is not known in advance.

When chunk $\mathscr{S}_t$ arrives, the learner may scan it multiple times but cannot revisit raw data from earlier chunks. Instead, it maintains a sliding window of *feature arrays*

$$\mathbf{H}_{t-1} = \{h_{t-W}, \ldots, h_{t-1}\} \subset \mathbb{R}^{W \times d_h},$$

where W is the fixed memory cap and $d_h = 256$. No raw inputs or labels from past chunks are stored, satisfying both privacy and storage constraints.

We train chunk-specific weights θ_t of the classifier f_{θ_t} using the regular cross-entropy loss:

$$\min_{\theta_t} \mathscr{L}_{\text{CE}}(f_{\theta_t}, \mathscr{S}_t) = -\frac{1}{n_t} \sum_{(x,y) \in \mathscr{S}_t} \log p_{\theta_t}(y \mid x),$$

where p_{θ_t} is the softmax output of the Incremental MLP (IMLP) attending to $\mathbf{H}_{t-1}$. The same loss is used on a held-out set for hyperparameter tuning.

Although energy is *not* part of the training objective, we track mains power during both training and inference:

$$E_t = E_{\text{train},t} + \alpha\, E_{\text{infer},t}, \qquad \alpha \in \{1, 10\}.$$

After processing each chunk, we compute the **NetScore-T** to balance accuracy against energy consumption:

$$\mathrm{NS}_t^{(\mathrm{BA})} = \frac{\mathrm{BalancedAcc}_t}{\log_{10}(E_t + 1)}, \quad \mathrm{NS}_t^{(\mathrm{LL})} = \frac{(\mathrm{LogLoss}_t + \varepsilon)^{-1}}{\log_{10}(E_t + 1)},$$

where $\varepsilon = 10^{-7}$ stabilizes the log-loss. For the entire data stream, we average NS_t over $t = 1, \ldots, T$.

- **Training goal:** minimize the cross-entropy loss above.
- **Evaluation metric:** NetScore-T defined by the formulas above.

NOTATION SUMMARY

Table 3.1: Notation used throughout the paper

Symbol	Description
$\mathscr{S}_t$	Chunk t, containing n_t samples
T	Total number of chunks (unknown beforehand)
d_{in}	Dimensionality of the input features
d_h	Hidden-state width (fixed at 256)
W	Sliding-window size (feature-memory capacity)
θ_t	IMLP parameters after training on $\mathscr{S}_t$
E_t	Total energy (training + inference) for chunk t
α	Weighting factor for inference energy (1 or 10)
ε	Log-loss stabilizer (10^{-7})

3.2. DATA CHARACTERISATION

3.2.1. DATA SELECTION

We use the *TabZilla* benchmark, a set of 36 tabular classification tasks taken from OpenML. The set aims for a wide variety of subjects, a balanced number of classes, and mixtures of small and large feature spaces [14]. Rows arrive in their original time order and are split by Algorithm 2 into chunks with $500 \le k \le 1000$ rows. This keeps enough statistical strength while limiting memory use [38]. Datasets smaller than k_{min} simply use one chunk, matching the reference code in Listing 6.

TRAIN/TEST SPLITS

Before we split into chunks, we set aside a stratified test set of 15% (`random_seed=42`). Each later chunk gets its own 15% stratified validation split with seed 42 + seg_idx. If balancing the classes is impossible, we fall back to a fixed but un-balanced split [80].

3.2.2. PRE-PROCESSING AND FEATURE PIPELINES

Heterogeneous columns are processed with a `ColumnTransformer` [81] that applies to pipelines to numeric and categorical features. Numeric columns use median imputation via `SimpleImputer(strategy='median')` followed by `StandardScaled` to obtain zero-mean, unit-variance features. Categorical columns use a constant placeholder (`fill_value='missing'`) with `SimpleImputer`, then `OneHotEncoder(drop='first', handle_unknown='ignore', sparse_output=False)`. The transformer is fit on the training split and reused unchanged for validation adn test to avoid leakage. Outputs are case to `float32` before entering the model to reduce memory usage and keep types consistent with PyTorch.

Listing 1 Pre-processing pipeline (`preprocessing.py`, lines 21-52).

```
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler",  StandardScaler())
])

categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="constant", fill_value="missing")),
    ("onehot",  OneHotEncoder(drop="first",
                              sparse_output=False,
                              handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", numeric_transformer, numerical_features),
    ("cat", categorical_transformer, categorical_features)
], remainder="drop")
```

3.2.3. HANDLING CLASS IMBALANCE

Performance is measured with *balanced accuracy*-the simple average of recall for each class-using `sklearn.metrics.balanced_accuracy_score` [81]. Balanced accuracy reduces bias toward the majority class and is supported by prior work on uneven class sizes [80]. We do not use resampling, class weights, focal loss, or β-balanced measures.

3.2.4. TRAINING OVERVIEW

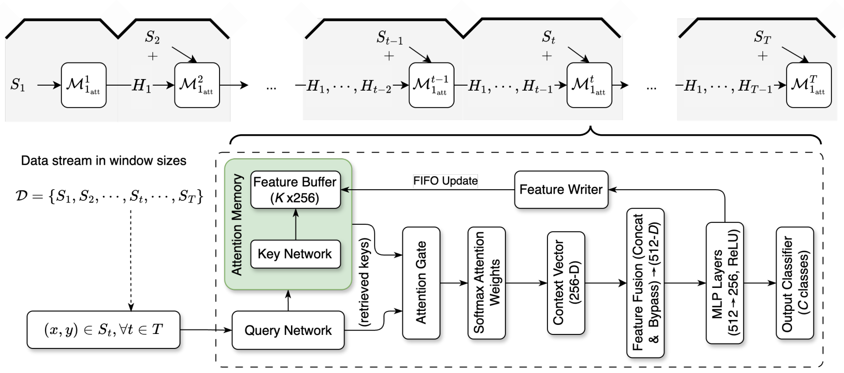
Early stopping ends training after ten epochs without better validation balanced accuracy (or the value set in each YAML file). Baseline models-XGBoost [15], LightGBM [16], CatBoost [43], k-NN, SVM, tree ensembles, plus neural models like TabNet [17], NODE [46], TabTransformer [48], and SAINT [18]-are retrained on all previous chunks $\bigcup_{t \le T} S_t$. By contrast, the IMLP trains only on the current chunk S_T while reusing saved feature representations through its attention blocks.

3.2.5. PRIVACY AND GOVERNANCE

Even though TabZilla tasks are public and appear anonymised, linking different datasets can still identify people. We therefore enforce: (i) source-level anonymising by using only the OpenML copies, (ii) AES-encrypted local storage limited to authorised staff, (iii) full compliance with GDPR Recital 26 on data that can't be traced back [10], and (iv) sharing only summary scores and model weights-never raw records.

Licence note. The TabZilla repo currently has no clear licence to redistribute the underlying datasets; so we cite only the OpenML task IDs and leave out the raw data in the supplementary files.

3.3. THE INCREMENTAL MLP (IMLP)



S_1
S_2
S_{t-1}
S_t
S_T
$\mathcal{M}^1_{1_{\text{att}}}$
$\mathcal{M}^2_{1_{\text{att}}}$
$\mathcal{M}^{t-1}_{1_{\text{att}}}$
$\mathcal{M}^t_{1_{\text{att}}}$
$\mathcal{M}^T_{1_{\text{att}}}$
H_1
$H_1, \cdots, H_{t-2}$
$H_1, \cdots, H_{t-1}$
$H_1, \cdots, H_{T-1}$
Data stream in window sizes
$\mathcal{D} = \{S_1, S_2, \cdots, S_t, \cdots, S_T\}$
$(x, y) \in S_t, \forall t \in T$
Attention Memory
Feature Buffer (K x256)
Key Network
Query Network
FIFO Update
Feature Writer
(retrieved keys)
Attention Gate
Softmax Attention Weights
Context Vector (256-D)
Feature Fusion (Concat & Bypass) → (512-D)
MLP Layers (512 → 256, ReLU)
Output Classifier (C classes)

Figure 3.1: Schematic of the Incremental MLP. Every incoming sample $\mathbf{x}$ is first mapped to a 256-D query $\mathbf{q}$, the query retrieves a context vector $\mathbf{c}$ from the *fixed-size* feature memory holding the most recent W segment embeddings $\mathbf{K} = \{h_{t-1}, \ldots, h_{t-W}\}$. The concatenated pair $(\mathbf{x}, \mathbf{c})$ is then processed by two shared feed-forward layers and a classifier head.

3.3.1. HIGH-LEVEL INTUITION

A conventional MLP ignores the sequential structure of a data stream and forgets previously seen patterns once its weights are updated. **IMLP tackles this limitation by rehearsing *features* rather than raw samples.** Concretely, after each segment we detach and store the 256-D activation h_{t-i} produced by the penultimate layer. At any later step the current query $\mathbf{q}$ can softly attend to this buffer of at most W vectors, harvesting a summary of recent experience without keeping any personally identifiable data. Because the buffer size is capped, both memory footprint and lookup cost scale as $\mathcal{O}(Wd_h)$ regardless of how long the stream runs.

3.3.2. LAYER-WISE DEFINITION

Table 3.2 lists every trainable block. The only difference from a standard two-layer MLP is the additional query/key projection pair and the concatenation of the 256-D context vector in front of the first hidden layer. All datasets-small or large, balanced or imbalanced, share this exact configuration, ensuring that improvements stem from the *incremental* design rather than architecture tuning.

With input width $d_{\text{in}} \le 2000$ and class count $C \le 20$, the model contains less than $\mathbf{1.2 \times 10^6}$ parameters, a break-down is shown in §3.4.

3.3.3. WINDOWED ATTENTION AND BUFFER MAINTENANCE

During a forward pass with batch size B we stack the normalised keys into $\mathbf{K} \in \mathbb{R}^{B\times W\times 256}$ and obtain the context vector via

$$\mathbf{c} = \text{softmax}\left(\frac{\mathbf{K}\mathbf{q}^{\top}}{\sqrt{d_h}}\right)\mathbf{K}, \qquad d_h = 256, \tag{3.1}$$

Table 3.2: IMLP building blocks (identical across the evaluation suite).

Component	Output dim.	Activation	Purpose / Notes
Query projection Q	256	-	projects raw input to query space
Key projection K	256	-	re-projects stored features for attention
Attention context $\mathbf{c}$	256	-	scaled dot-product attention (Eq. 3.1)
FC_1	512	ReLU	first hidden layer after context concat
FC_2	256	ReLU	second hidden layer (penultimate)
Classifier	C	-	single linear layer to logits

where each stored activation is $\tilde{h}_i = h_i/(\|h_i\|_2+\varepsilon)$ with $\varepsilon = 10^{-8}$. After the prediction step the freshly computed (detached) feature $\bar{h}_t$ is pushed into the FIFO queue. If the queue already holds W items, the oldest one is removed.

3.3.4. TRAINING LOOP AT A GLANCE

Algorithm 1 summarises the end-to-end optimisation routine. Variable names map directly to the code in Listing 2.

Algorithm 1 Context-Aware Incremental MLP (IMLP)

```
Require: Stream $\mathscr{D} = \{S_1, S_2, \ldots, S_T\}$ of data segments, Window size $W$, learning rate $\eta$,
    attention flag $1_{att}$
 1: Initialize $\theta \leftarrow$ random weights
 2: Initialize BUFFER $\leftarrow \emptyset$                          ▷ Stores at most $W$ feature tensors
 3: for $t = 1$ to $T$ do
 4:     for minibatch $(x, y) \in S_t$ do                                    ▷ Forward pass
 5:         if $1_{att}$ and BUFFER $\neq \emptyset$ then
 6:             $K \leftarrow$ Stack(BUFFER)                                 ▷ Shape: $[B, W, d]$
 7:             $q \leftarrow \phi_q(x)$                                      ▷ Query: $[B, 1, d]$
 8:             $\alpha \leftarrow$ softmax$(K \cdot q^\top)$
 9:             $c \leftarrow \alpha \cdot K$                                 ▷ Context: $[B, d]$
10:         else
11:             $c \leftarrow 0$
12:         $h \leftarrow \phi_{feat}(\text{concat}(x, c))$
13:         $p \leftarrow$ softmax$(\phi_{cls}(h))$                           ▷ Backward pass
14:         $L \leftarrow$ CE$(p, y)$
15:         $\theta \leftarrow \theta - \eta \nabla_\theta L$
16:     if $1_{att}$ then
17:         $f \leftarrow$ Detach(mean feature of $S_t$)
18:         BUFFER $\leftarrow$ BUFFER $\cup \{f\}$; trim to $W$
19: return $\theta$
```

3.3.5. DETAILED EXPLANATION OF TRAINING ALGORITHM

Algorithm 1 implements the core incremental learning procedure for IMLP. Below is a step-by-step explanation aligned with Listing 2 and Eq. 3.1. We use batch size B, input width d_{in}, hidden width d_h=256, and class count C.

1. **Initialization (Lines 1-3):** Parameters $\theta = \{\phi_q, \phi_K, \phi_{\text{feat}}, \phi_{\text{cls}}\}$ are randomly initialized. The FIFO BUFFER is empty and will hold at most W 256-D feature vectors (detached tensors).

2. **Segment loop (Line 4):** The stream $\mathscr{D} = \{S_1, \dots, S_T\}$ is processed in temporal order to match continual learning.

3. **Minibatch loop (Line 5):** Each segment S_t is consumed in minibatches (x, y) for efficient optimization.

4. **Context retrieval (Lines 6-11):**
   - If attention is enabled ($\mathbf{1}_{\text{att}}$=1) and the buffer is non-empty:
     - Stored features are stacked as $K \in \mathbb{R}^{W \times d_h}$ and (for batched matmul) viewed/broadcast as $[B, W, d_h]$ (Line 7). Before scoring, features are ℓ_2-normalized: $\tilde{h}_i = h_i/(\|h_i\|_2 + \varepsilon)$ and re-projected with ϕ_K.
     - Inputs are projected to queries $q = \phi_q(x) \in \mathbb{R}^{B \times d_h}$ and reshaped to $[B, 1, d_h]$ (Line 8).
     - Attention weights over the W keys are
       $$\alpha = \text{softmax}\Big(\frac{Kq^\top}{\sqrt{d_h}}\Big) \in \mathbb{R}^{B \times W},$$
       i.e., softmax along the window dimension (Line 9).
     - The context is the convex combination $c = \alpha K \in \mathbb{R}^{B \times d_h}$ (Line 10).
   - Otherwise, the context defaults to the zero vector: $c = \mathbf{0}$ (Line 11), so the model reduces to a plain MLP for that batch.

5. **Feature extraction (Line 12):** Inputs and context are concatenated into $z = \text{concat}(x, c) \in \mathbb{R}^{B \times (d_{\text{in}} + d_h)}$ and passed through two shared feed-forward layers $h = \phi_{\text{feat}}(z) \in \mathbb{R}^{B \times d_h}$.

6. **Prediction (Lines 13-14):** Logits are $o = \phi_{\text{cls}}(h) \in \mathbb{R}^{B \times C}$ and probabilities $p = \text{softmax}(o)$.

7. **Optimization (Lines 16-17):** Compute cross-entropy $L = \text{CE}(p, y)$ and update parameters by a first-order step, abstractly $\theta \leftarrow \theta - \eta \nabla_\theta L$. Gradients flow only through the current batch; stored features are detached.

8. **Buffer update (Lines 18-20):** After all minibatches of S_t, compute one representative feature
   $$\bar{h}_t = \text{Detach}\big(\tfrac{1}{|S_t|} \textstyle\sum_{(x,y) \in S_t} h\big),$$
   append it to BUFFER, and evict the oldest item if $|\text{BUFFER}| > W$ (FIFO). This keeps memory $\mathscr{O}(W d_h)$ and avoids storing raw samples.

Per-batch cost is $\mathscr{O}\big(B(d_{\text{in}} d_h + W d_h^2)\big)$; extra memory is $\mathscr{O}(W d_h)$. You find a more in depth derivation in §3.4.

3.4. COMPUTATIONAL ANALYSIS OF IMLP

We examine how the incremental learning design impacts resource requirements, particularly regarding parameter count, computational complexity, and memory usage. We decompose the model's parameter count, revealing how the addition of attention-based memory slightly increases complexity compared to conventional MLPs. Additionally, we discuss the per-batch computational overhead introduced by the attention mechanism, quantifying its cost with respect to the input dimensions and the fixed window size W. Furthermore, we assess memory requirements, highlighting that the feature buffer used in IMLP maintains a fixed, bounded memory footprint independent of the data stream length. Finally, we position IMLP within the broader landscape of continual learning methods, emphasizing its efficiency and privacy- preserving advantages relative to traditional replay based approaches.

3.4.1. ARCHITECTURAL CONTEXT

Table 3.3 restates the headline architectural differences between a plain multilayer perceptron and IMLP, while Table 3.4 gives the layer-by-layer specification.

Table 3.3: High-level comparison between a conventional MLP and the proposed IMLP.

Component	**MLP**	**IMLP**
Input projection	$d_{\text{in}} \to 512$	$d_{\text{in}} \to 256$
Memory mechanism	–	Attention
Feature extractor	$512 \to 256$	$(d_{\text{in}} + 256) \to 512 \to 256$
Memory growth	$\mathcal{O}(1)$	$\mathcal{O}(W)$
Time per sample	$\mathcal{O}(1)$	$\mathcal{O}(W\, d_h^2)$
Privacy	Stores raw data	Stores features only

3.4.2. PARAMETER COUNT

The total number of learnable weights equals the sum of the five matrices highlighted in Tables 3.3–3.4. Using d_h=256 and omitting biases:

$$\underbrace{d_{\text{in}} d_h}_{\text{query}} + \underbrace{d_h^2}_{\text{key}} + \underbrace{(d_{\text{in}} + d_h)\, 512}_{\text{first FC}} + \underbrace{512\, d_h}_{\text{second FC}} + \underbrace{d_h C}_{\text{classifier}} = \mathcal{O}\big(d_{\text{in}} d_h + d_h^2 + d_h C\big).$$

With $d_{\text{in}} \lesssim 2000$ and $C \ll d_h$ the result is roughly **1.2 million** weights.

Table 3.4: Layer-wise specification of IMLP

Component	**Output dim.**	**Activation**	**Notes**
Input feature vector	d_{in}	–	Raw tabular input
Attention Module			
Query projection Q	256	–	Linear$(d_{\text{in}}, 256)$
Key projection K	256	–	Linear$(256, 256)$ applied to each stored feature
Context computation	256	–	Scaled dot-product attention (Eq. 3.1)
Feature Extraction			
Concatenated (x, c)	$d_{\text{in}} + 256$	–	Only when attention is active
FC$_1$	512	ReLU	Linear$(d_{\text{in}} + 256, 512)$
FC$_2$	256	ReLU	Linear$(512, 256)$
Classification Head			
Classifier	C	–	Linear$(256, C)$

3.4.3. PER-BATCH TIME COMPLEXITY

For a mini-batch of B samples and a sliding window of W cached 256-dimensional feature vectors:

$$\text{Query projection:} \quad \mathcal{O}(B \cdot d_{\text{in}} \cdot d_h) \tag{3.2}$$

$$\text{Key projection:} \quad \mathcal{O}(B \cdot W \cdot d_h^2) \tag{3.3}$$

$$\text{Attention scores:} \quad \mathcal{O}(B \cdot W \cdot d_h) \tag{3.4}$$

$$\text{Context aggregation:} \quad \mathcal{O}(B \cdot W \cdot d_h) \tag{3.5}$$

$$\text{Feature extraction:} \quad \mathcal{O}(B \cdot (d_{\text{in}} + d_h) \cdot 512) \tag{3.6}$$

$$\text{Total:} \quad \mathcal{O}(B \cdot (d_{\text{in}} \cdot d_h + W \cdot d_h^2)) \tag{3.7}$$

With the default W=10 and d_h=256, $W\,d_h^2 = 10 \times 256^2 = 655\,360$ multiply-adds per *sample*, i.e. well below one megaflop.

3.4.4. MEMORY COMPLEXITY

Aside from fixed model parameters, the only additional memory comes from the feature buffer that drives the attention mechanism:

$$M_{\text{buffer}} = W\,d_h \times \texttt{sizeof(fp32)} = \mathcal{O}(W\,d_h).$$

Because the oldest feature is discarded whenever the window fills, this extra memory remains *constant* with respect to the number of tasks T. Replay-based methods, in contrast, grow as $\Theta(T)$ unless a strict cap is imposed on stored samples.

3.4.5. COMPARISON WITH BASELINE STRATEGIES

Table 3.5 emphasises that IMLP achieves privacy-preserving continual learning at constant memory cost, trading only a quadratic $W\,d_h^2$ compute term for its feature-only rehearsal buffer.

Listing 2 Central lines of the `IncrementalMLP` implementation.

```python
class IncrementalMLP(nn.Module):
    """Incremental MLP with attention-based feature replay."""
    def __init__(self, input_size, num_classes, window_size=10):
        super().__init__()
        self.window_size = window_size
        self.d_h = 256
        # Attention projections
        self.query = nn.Linear(input_size, self.d_h)
        self.key   = nn.Linear(self.d_h, self.d_h)
        # Feature extraction pathway
        self.feat = nn.Sequential(
            nn.Linear(input_size + self.d_h, 512), nn.ReLU(),
            nn.Linear(512, self.d_h),              nn.ReLU()
        )
        self.classifier = nn.Linear(self.d_h, num_classes)

    def compute_context(self, x, prev):
        if not prev:
            return x.new_zeros(x.size(0), self.d_h)
        keys = self.key(torch.stack(prev, dim=1))       # [B,W,256]
        q    = self.query(x).unsqueeze(1)               # [B,1,256]
        attn = (keys @ q.transpose(1,2)).squeeze(-1)    # [B,W]
        attn = torch.softmax(attn, dim=1)
        ctx  = (attn.unsqueeze(1) @ keys).squeeze(1)    # [B,256]
        return ctx

    def forward(self, x, prev=None):
        ctx = self.compute_context(x, prev) if prev else 0
        feats = self.feat(torch.cat([x, ctx], dim=1))
        return self.classifier(feats), feats
```

Table 3.5: Complexity of common continual-learning approaches

Method	**Memory**	**Time / step**	**Requires raw data?**
Naïve retraining	$\mathcal{O}(TN)$	$\mathcal{O}(TN)$	Yes
Experience replay	$\mathcal{O}(M)$	$\mathcal{O}(N+M)$	Yes
Generative replay	$\mathcal{O}(1)$	$\mathcal{O}(N+G)$	No
IMLP (ours)	$\mathcal{O}(W)$	$\mathcal{O}(N+W d_h^2)$	No

IMLP confines its additional cost to a modest quadratic compute term and a small, fixed-sized feature buffer. This design avoids the unbounded growth in memory and data-privacy issues faced by replay baselines, while remaining computationally lightweight on modern hardware.

3.5. DATASETS AND PREPROCESSING

3.5.1. DATASET SELECTION AND CHARACTERISTICS

We evaluate IMLP on 36 classification tasks from the TabZilla benchmark [14], accessed through the OpenML platform [38]. This collection spans diverse domains including medical diagnosis, sensor data, text classification, and financial applications. The selection criteria prioritise: (i) sufficient instances to create meaningful temporal segments, (ii) balanced representation of binary and multi-class problems, and (iii) varied feature dimensionalities ranging from 5 to 2,000 dimensions.

Table 3.6 summarises key characteristics. Binary tasks comprise 44% of the benchmark, with the remainder being multi-class problems with up to 26 categories. Instance counts range from 3,190 (splice) to 22,784 (house_16H), while feature dimensionalities span from 5 (wilt, phoneme) to 2,000 (dilbert). This diversity ensures our incremental learning evaluation covers realistic tabular scenarios.

Table 3.6: Summary statistics across the 36 TabZilla classification tasks.

Dataset characteristic	**Min**	**Median**	**Max**	**Mean**
Number of instances	3,190	8,308	22,784	9,344
Number of features	5	36	2,000	220
Number of classes	2	2	26	4.5
Segment size	500	619	951	668
Number of segments	5	11	23	12.1

3.5.2. STREAM SEGMENTATION PROTOCOL

To simulate continual learning scenarios, we partition each dataset into temporal segments $\mathscr{S}_1, \mathscr{S}_2, \ldots, \mathscr{S}_T$ while preserving the original row order from OpenML. This maintains any natural temporal dependencies present in the data collection process. Algorithm 2 determines optimal segment sizes between 500 and 1,000 instances, minimising data wastage while ensuring sufficient statistical power per chunk.

The bounds $[k_{\min}, k_{\max}] = [500, 1000]$ balance three considerations:

1. **Statistical reliability**: Each segment must contain enough samples for stable gradient estimates and meaningful validation splits.
2. **Attention coherence**: Segments should be large enough for the feature memory mechanism to learn useful representations within each temporal window.
3. **Computational efficiency**: Larger segments increase per-batch memory requirements without proportional accuracy gains.

When the optimal segment size k^* leaves a remainder $r = N \bmod k^*$, we apply round-robin redistribution: the first r segments each receive one additional instance. This ensures segment sizes differ by at most one instance while preserving chronological order.

3.5.3. FEATURE PREPROCESSING PIPELINE

Following established best practices for tabular data [81], we apply domain-appropriate transformations to numerical and categorical features separately. The pipeline architecture, implemented via scikit-learn's `ColumnTransformer`, ensures consistent preprocessing across all data splits:

Numerical features undergo two-stage processing:

1. **Imputation**: Missing values are replaced with column medians computed on the training portion of each segment. This robust statistic is less sensitive to outliers than the mean.
2. **Standardisation**: Features are scaled to zero mean and unit variance using `StandardScaler`, facilitating gradient-based optimisation.

Categorical features receive specialised handling:

1. **Missing value encoding**: Absent categories are filled with the constant string `'missing'`, preserving missingness as potentially informative.
2. **One-hot encoding**: Categories are binarised with `drop='first'` to prevent multicollinearity. The `handle_unknown='ignore'` setting ensures robustness to novel categories during inference.

All transformed features are cast to `float32` precision, reducing memory footprint by 50% compared to default `float64` with negligible impact on model performance. The complete pipeline specification appears in Listing 1.

3.5.4. DATA SPLITTING AND EVALUATION PROTOCOL

Our evaluation framework simulates realistic deployment where future data remains strictly inaccessible during training. The splitting hierarchy operates at three levels:

1. **Global test set**: Before any streaming simulation, we extract a stratified 15% test split using `random_state=42`. This held-out set evaluates final model performance across all segments and never participates in training or validation.
2. **Streaming sequence**: The remaining 85% forms the training stream, maintaining original row order. Segments $\mathscr{S}_1,\ldots,\mathscr{S}_T$ arrive sequentially without shuffling.
3. **Per-segment validation**: Within each training scenario, we create a stratified 15% validation split for hyperparameter selection and early stopping. The seed `random_state=42+segment_idx` ensures reproducible yet distinct splits per segment.

This three-tier approach prevents data leakage while enabling fair comparison across methods. All models-whether incremental or cumulative-use identical validation procedures within each segment.

3.5.5. Training protocols

We implement two distinct training regimes reflecting fundamentally different approaches to continual learning:

Cumulative training (baselines): Traditional methods including gradient boosting (XGBoost, LightGBM, CatBoost) [15, 16, 43], instance-based learning (k-NN), kernel methods (SVM), tree ensembles (Random Forest) [81], and neural baselines (e.g. TabNet [17], SAINT [18]) are retrained from scratch at each segment. These models access the complete historical data $\bigcup_{t=1}^{T} \mathscr{S}_t$, representing the upper bound of performance achievable with perfect memory.

Incremental training (IMLP): Our proposed architecture processes only the current segment $\mathscr{S}_T$ while accessing compressed representations of previous segments through the attention mechanism. Raw historical data remains inaccessible, enforcing strict privacy constraints and constant memory usage.

Both protocols employ early stopping based on validation balanced accuracy, terminating training after 10 epochs without improvement. This ensures convergence while preventing overfitting to small segments. The distinction between cumulative and incremental training directly tests our hypothesis that feature-level memory can approximate full data replay while maintaining privacy guarantees.

3.6. Training and Optimisation

The performance of neural architectures on tabular data critically depends on careful hyperparameter selection and training procedures [44]. This section details our comprehensive optimisation framework, which ensures fair comparison between IMLP and baseline methods while exploring modern deep learning techniques that enhance tabular learning performance.

3.6.1. Hyperparameter Optimisation Framework

We adopt a systematic approach to hyperparameter tuning that extends beyond traditional grid search, implementing what Kadra et al. [44] term a "regularisation cocktail"- a principled exploration of complementary regularisation techniques. This methodology acknowledges that optimal configurations for tabular neural networks often involve complex interactions between multiple regularisation strategies, architectural choices, and training procedures.

Our framework employs Bayesian optimisation via Optuna [82], leveraging its Tree-structured Parzen Estimator (TPE) algorithm to efficiently navigate the high-dimensional search space. The TPE sampler models the probability of hyperparameter configurations conditioned on their performance, enabling intelligent exploration that balances exploitation of promising regions with exploration of uncertain areas.

3.6.2. Search Space Definition

The hyperparameter search space encompasses four complementary families of techniques, each addressing different aspects of the learning process. Additionally, certain training parameters remain fixed across all experiments to ensure fair comparison while reducing the search space dimensionality.

Fixed Training Configuration

The following parameters remain constant throughout all experiments:

- **Learning Rate**: 1×10^{-4} (not optimised to maintain training stability)
- **Batch Size**: 128 samples (balances memory usage with gradient quality)
- **Optimizer**: AdamW with conditional weight decay
- **Scheduler**: ReduceLROnPlateau with factor 0.5, patience 5, monitoring validation accuracy
- **Training Budget**: Maximum 100 epochs with early stopping (patience=10)

Implicit Regularisation Techniques

Implicit regularisation methods influence the optimisation trajectory without directly modifying the loss function:

- **Batch Normalisation**: Applied after each linear transformation, with learnable affine parameters (`use_batch_norm` $\in \{\text{True}, \text{False}\}$)
- **Stochastic Weight Averaging** (SWA): Maintains a running average of model weights during the final 25% of training epochs (`use_swa` $\in \{\text{True}, \text{False}\}$)

These techniques stabilise training dynamics and often improve generalisation without explicit penalty terms.

Explicit Regularisation Methods

Explicit regularisation directly penalises model complexity:

- **Weight Decay**: L2 penalty on model parameters
  - `use_weight_decay` $\in \{\text{True}, \text{False}\}$
  - `weight_decay` $\in [10^{-5}, 10^{-1}]$ (log-uniform) when enabled
- **Dropout**: Stochastic neuron deactivation
  - `use_dropout` $\in \{\text{True}, \text{False}\}$
  - `dropout_rate` $\in [0.0, 0.8]$ (uniform) when enabled
  - `dropout_shape` $\in \{\text{funnel}, \text{long\_funnel}, \text{diamond}, \text{triangle}\}$

The dropout patterns control rate progression through network layers:

- *Funnel*: $p_i = \min(\text{rate} \times (i+1)/L, 0.9)$
- *Long Funnel*: $p_i = \min(\text{rate}, 0.9)$ for $i < L-1$, then rate/2
- *Diamond*: $p_i = \min(\text{rate} \times \min(i+1, L-i)/\lfloor L/2 \rfloor, 0.9)$
- *Triangle*: $p_i = \min(\text{rate} \times (L-i)/L, 0.9)$

where $L = 3$ is the number of layers and rates are capped at 0.9 for numerical stability.

Architectural Enhancements

We explore residual connection variants that have proven effective for tabular data:

- **Skip Connections**: `use_skip` $\in \{\text{True}, \text{False}\}$
- **Skip Type** (when enabled): `skip_type` $\in \{\text{Standard}, \text{ShakeShake}, \text{ShakeDrop}\}$
  - *Standard*: Identity mappings bypass the network
  - *ShakeShake* [83]: Stochastic linear interpolation between branches
  - *ShakeDrop* [84]: Probabilistic path dropping
- **ShakeDrop Probability**: `shakedrop_prob` $\in [0.0, 1.0]$ (uniform) when `skip_type` $= \text{ShakeDrop}$

These techniques address gradient flow issues while introducing beneficial stochasticity during training.

Data Augmentation and Training Enhancements

Modern training techniques adapted for tabular data:

- **Data Augmentation**: `augmentation` $\in \{\text{None}, \text{MixUp}\}$
  - *MixUp* [85]: Linear interpolation between training examples
  - `aug_magnitude` $\in [0.0, 1.0]$ (uniform) controls mixing strength when enabled
- **Mixed Precision Training**: `use_amp` $\in \{\text{True}, \text{False}\}$
  - Automatic mixed precision (AMP) for computational efficiency on CUDA devices
  - Automatically disabled on CPU to prevent compatibility issues
- **Gradient Clipping**: `max_grad_norm` $\in [0.1, 10.0]$ (log-uniform)
  - Applied after unscaling gradients in mixed precision mode
  - Prevents gradient explosion in deep networks

Conditional Dependencies

The search space includes conditional relationships between parameters:

- `weight_decay` is only active when `use_weight_decay` $= \text{True}$
- `dropout_shape` and `dropout_rate` require `use_dropout` $= \text{True}$
- `skip_type` requires `use_skip` $= \text{True}$
- `shakedrop_prob` requires `skip_type` $= \text{ShakeDrop}$
- `aug_magnitude` requires `augmentation` $= \text{MixUp}$

These conditions ensure the search space remains coherent while reducing the number of invalid configurations.

3.6.3. OPTIMISATION ALGORITHM AND PROTOCOL

The hyperparameter optimisation employs a multi-stage protocol designed to balance computational efficiency with thorough exploration:

SAMPLING STRATEGY

We utilise Optuna's multivariate TPE sampler with the following configuration:

- **Startup Trials**: 50 random configurations before TPE activation
- **Multivariate Modeling**: Enabled to capture parameter interactions
- **Constant Liar**: Enabled for improved parallel efficiency
- **Random Seed**: Fixed at 42 for reproducibility

EARLY STOPPING AND PRUNING

Two levels of early stopping prevent wasteful computation:

1. **Trial-level Pruning**: MedianPruner terminates unpromising trials based on intermediate validation performance, with 50 startup trials before pruning begins and 50 warmup steps per trial
2. **Epoch-level Stopping**: Within each trial, training halts after 10 epochs without validation improvement (patience=10)
3. **Study-level Stopping**: Optimization terminates if no improvement occurs across 100 consecutive trials

OBJECTIVE FUNCTION

The optimisation minimises:

$$\mathscr{L}_{\text{opt}} = 1 - \text{BalancedAcc}_{\text{val}}$$

where $\text{BalancedAcc}_{\text{val}}$ is computed on the segment-specific validation split. This choice aligns with our primary evaluation metric while handling class imbalance.

3.6.4. COMPUTATIONAL INFRASTRUCTURE

The comprehensive hyperparameter search demands substantial computational resources:

Table 3.7: Computational resources for hyperparameter optimisation across 36 TabZilla tasks.

Resource	**Specification**
Total optimisation trials	3,600 (100 per task)
Total GPU hours	≈ 72 hours
Parallelisation factor	22 concurrent trials
Storage format	SQLite databases

3.6.5. EXECUTION AND PERSISTENCE

The optimisation workflow leverages parallelisation and persistence for robustness:

PARALLEL EXECUTION

A bash orchestration script manages concurrent optimisation across tasks:

Listing 3 Parallel hyperparameter optimisation

```bash
#!/bin/bash
N_TRIALS=100
EPOCHS=100
DEVICE="cuda"
MAX_PARALLEL=22
DATA_ROOT="../data/full_datasets"

# Task discovery by scanning data directory
TASK_IDS=()
for TASK_DIR in ${DATA_ROOT}/openml_task_*; do
    if [ -d "$TASK_DIR" ]; then
        TASK_ID=$(basename "$TASK_DIR" | sed 's/openml_task_//')
        # Verify required files exist
        if [ -f "${TASK_DIR}/X_train.npy.gz" ] && \
           [ -f "${TASK_DIR}/X_test.npy.gz" ] && \
           [ -f "${TASK_DIR}/y_train.npy.gz" ] && \
           [ -f "${TASK_DIR}/y_test.npy.gz" ]; then
            TASK_IDS+=($TASK_ID)
        fi
    fi
done

# Parallel execution with resource constraints
printf "%s\n" "${TASK_IDS[@]}" | xargs -I {} -P ${MAX_PARALLEL} \
    bash -c 'python mlp_c.py \
        --task_id {} \
        --n_trials ${N_TRIALS} \
        --epochs ${EPOCHS} \
        --device ${DEVICE} \
        --storage "sqlite:///optuna_db/task_{}.db" \
        --data_root ${DATA_ROOT}'
```

STATE PERSISTENCE

Each task maintains an independent SQLite database enabling:

- **Interruption Recovery**: Optimisation resumes from the last completed trial
- **Reproducibility**: Complete trial history with random states
- **Post-hoc Analysis**: Hyperparameter importance and correlation studies

3.6.6. INTEGRATION WITH EXPERIMENTAL PIPELINE

The optimised hyperparameters integrate seamlessly with the main experimental framework:

AUTOMATIC CONFIGURATION LOADING

The training pipeline automatically detects and loads task-specific configurations:

Listing 4 Hyperparameter loading mechanism

```
def load_hyperparameters(args, base_config):
    """Load optimised hyperparameters if available."""
    tuning_file = f"tuning/task_{args.task}_hyperparams.yml"

    if not args.no_tuning and os.path.exists(tuning_file):
        with open(tuning_file, 'r') as f:
            tuned_params = yaml.safe_load(f)

        # Merge with priority to tuned parameters
        config = merge_configurations(base_config, tuned_params)
        print(f"Loaded tuned hyperparameters from {tuning_file}")
    else:
        config = base_config
        print("Using default hyperparameters")

    return config
```

CONFIGURATION PERSISTENCE FORMAT

Optimised configurations are stored as YAML files for human readability and version control:

Listing 5 Example hyperparameter configuration file (task_3481_hyperparams.yml)

```
aug_magnitude: 0.39900270387013026
augmentation: MixUp
dropout_rate: 0.0905904366726042
dropout_shape: long_funnel
max_grad_norm: 4.033730669603172
skip_type: ShakeShake
use_amp: false
use_batch_norm: true
use_dropout: true
use_skip: true
use_swa: false
use_weight_decay: true
weight_decay: 0.002019079599080459
```

3.6.7. REPRODUCIBILITY GUARANTEES

Ensuring reproducible results across different hardware and software environments requires careful attention to sources of non-determinism:

RANDOM STATE MANAGEMENT

We enforce deterministic behaviour through:

- **Global Seed**: Fixed seed (42) for NumPy, PyTorch, and Python's random module

- **Optuna Seeds**: Explicit seeding of TPE sampler with seed=42
- **Thread Limiting**: `OMP_NUM_THREADS=1` and `torch.set_num_threads(1)` for consistent parallelism

Numerical Stability

Several techniques ensure consistent results across platforms:

- **Gradient Clipping**: Prevents numerical overflow with configurable maximum norm
- **Mixed Precision Handling**: GradScaler with NaN checking to avoid underflow
- **Dropout Rate Capping**: Maximum rate of 0.9 to prevent complete layer deactivation
- **Batch Size Constraints**: Fixed at 128 to ensure identical gradient statistics

This comprehensive training and optimisation framework ensures that our experimental comparisons reflect the true potential of each method rather than artifacts of suboptimal hyperparameter choices. By investing substantial computational resources in systematic optimisation, we establish a rigorous foundation for evaluating the incremental learning capabilities of IMLP against well-tuned baselines.

3.7. Evaluation Metrics

The evaluation of continual learning systems requires careful consideration of both predictive performance and computational efficiency. This section defines the metrics used to assess IMLP and baseline methods across the streaming tabular benchmark. We first present conventional accuracy metrics that capture model effectiveness, then introduce NetScore-T, a novel composite metric that jointly evaluates prediction quality and energy consumption.

3.7.1. Conventional Predictive Metrics

We employ four complementary metrics to comprehensively assess predictive performance across diverse tabular tasks. Each metric captures different aspects of classification quality, with particular attention to class imbalance, a common characteristic in real-world tabular datasets.

Balanced Accuracy [80] serves as our primary performance metric due to its robustness to skewed class distributions. For a C-class problem, balanced accuracy computes the arithmetic mean of per-class recall values:

$$\text{BalancedAcc} = \frac{1}{C}\sum_{c=1}^{C}\frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}$$

where TP_c and FN_c denote true positives and false negatives for class c, respectively. This formulation ensures that performance on minority classes contributes equally to the overall score, preventing models from achieving artificially high scores by focusing solely on majority classes.

F1-Score [86] provides a harmonic mean of precision and recall, offering a single metric that balances both aspects of classification performance. For multi-class problems, we compute the macro-averaged F1-score:

$$\text{F1}_{\text{macro}} = \frac{1}{C}\sum_{c=1}^{C} \frac{2 \cdot \text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

where $\text{Precision}_c = \text{TP}_c/(\text{TP}_c + \text{FP}_c)$ and $\text{Recall}_c = \text{TP}_c/(\text{TP}_c + \text{FN}_c)$. The macro-averaging strategy aligns with our balanced accuracy approach, treating all classes with equal importance.

Log-Loss [87] (cross-entropy loss) evaluates the quality of probabilistic predictions by penalising confident misclassifications more severely than uncertain ones:

$$\text{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(p_{i,c})$$

where N is the number of test samples, $y_{i,c}$ is the binary indicator for the true class, and $p_{i,c}$ is the predicted probability for sample i belonging to class c. Unlike accuracy-based metrics, log-loss provides a calibrated assessment of prediction confidence.

AUC-ROC [88] (Area Under the Receiver Operating Characteristic curve) measures the model's ability to discriminate between classes across all possible classification thresholds. For multi-class problems, we compute the macro-averaged AUC using the one-vs-rest strategy, ensuring consistent evaluation across tasks with varying numbers of classes.

3.7.2. NETSCORE-T DERIVATION

To quantify the trade-off between predictive performance and energy consumption in streaming tabular settings, we introduce **NetScore-T**, a task-agnostic metric that rewards accuracy while penalising computational cost. Building upon the NetScore framework for static vision models [20] and Energy NetScore for continual learning [21], NetScore-T adapts these concepts specifically for tabular data streams.

Per-Segment Formulation: For a model m evaluated on segment $\mathcal{S}_t$, we define the per-segment NetScore-T as:

$$\text{NS}_t^{(m)} = \frac{P_t^{(m)}}{\log_{10}(E_t^{(m)} + 1)}$$

where $P_t^{(m)}$ represents the performance metric (e.g., balanced accuracy) and $E_t^{(m)}$ denotes the total energy consumption in Joules. The logarithmic denominator serves two purposes: (i) it compresses the typically three-orders-of-magnitude spread in energy measurements, and (ii) the +1 term prevents division by zero for negligible energy consumption.

Energy Composition: The total energy $E_t^{(m)}$ comprises both training and inference components:

$$E_t^{(m)} = E_{\text{train},t}^{(m)} + \alpha \cdot E_{\text{infer},t}^{(m)}$$

where α controls the relative importance of inference energy. Energy measurements are obtained using an ElmorLabs PMD-USB power meter sampling at 500-800 Hz, capturing wall power between the workstation's power supply and mains.

Stream-Level Aggregation: For a complete data stream comprising T segments, we compute the average NetScore-T:

$$\text{NetScore-T}^{(m)} = \frac{1}{T}\sum_{t=1}^{T} \text{NS}_t^{(m)}$$

This formulation treats all temporal segments equally, reflecting the assumption that each chunk contributes comparably to the overall continual learning performance.

Metric Transformations: When using log-loss as the performance metric, we transform it to a "higher-is-better" quantity to maintain consistency with the NetScore-T interpretation:

$$P_t^{(m)} = \frac{1}{\mathscr{L}_t^{(m)} + \varepsilon}, \quad \varepsilon = 10^{-7}$$

where $\mathscr{L}_t^{(m)}$ is the log-loss value and ε stabilises the transformation for near-perfect predictions. Similarly, we denote the balanced accuracy and log-loss versions as $\text{NS}_t^{(\text{BA})}$ and $\text{NS}_t^{(\text{LL})}$, respectively.

4
Experimental Setup

This chapter describes the experimental framework developed to evaluate continual learning approaches for tabular data streams. We detail the hardware and software infrastructure, data stream construction methodology, training protocols, energy measurement procedures, and statistical evaluation framework employed throughout this study.

4.1. Hardware & software stack

4.1.1. Computational Infrastructure

All experiments were conducted on a single workstation to ensure consistent and reproducible measurements. The hardware configuration comprised:

- **CPU:** Intel Core i5-8600K @ 3.60GHz (6 physical cores, 6 logical cores)
- **GPU:** NVIDIA GeForce RTX 2080 Ti Rev. A (11 GB VRAM)
- **Memory:** 16GB DDR4 RAM
- **Storage:** NVMe SSD for data and model checkpoints
- **Architecture:** x86_64

This configuration represents a typical research workstation, balancing computational capability with energy measurement precision. The single-GPU setup eliminates multi-device synchronization complexities while providing sufficient compute for the evaluated model architectures.

4.1.2. Software Environment

The software stack was carefully configured to ensure reproducibility:

- **Operating System:** Debian GNU/Linux 12 (bookworm)
- **Kernel:** Linux 6.1.0-32-amd64

- **Python:** Version 3.13.0
- **CUDA Toolkit:** Version 12.2
- **Compiler:** GCC 12.2.0 (Debian 12.2.0-14)

All Python dependencies were frozen using `pip freeze` and stored in a requirements file to ensure exact reproducibility.

4.2. STREAM CONSTRUCTION

4.2.1. DATASET SELECTION AND PREPARATION

We evaluate our methods on 36 classification tasks from the *TabZilla* benchmark [19], a comprehensive collection of tabular datasets originally curated by [44] and [14]. These datasets were selected to ensure:

1. Sufficient data volume for meaningful segmentation (minimum 3,000 instances)
2. Diverse feature dimensionalities (ranging from 5 to 2,000 features)
3. Balanced representation of binary and multi-class problems
4. Variety in class distributions and domain characteristics

4.2.2. SEGMENTATION STRATEGY

To simulate realistic continual learning scenarios, each dataset was divided into contiguous segments following a principled approach:

1. **Segment Size Optimization:** Each dataset was segmented into chunks of 500–1,000 samples, with the exact size chosen to minimize data wastage while maintaining statistical validity
2. **Temporal Ordering:** Original row ordering was preserved to maintain any inherent temporal structure
3. **Balanced Distribution:** When remainder samples existed, they were distributed round-robin across segments to ensure size differences of at most one sample
4. **Label Concealment:** During training on segment t, models had no access to data or labels from future segments $\{t+1, \ldots, T\}$

The segmentation algorithm optimized for minimal remainder:

$$k^* = \arg \min_{k \in [500,1000]} (N \bmod k) \tag{4.1}$$

where N is the total number of training instances and k is the segment size.

4.2.3. TRAIN-VALIDATION-TEST SPLITTING

A consistent splitting strategy was applied across all experiments:

- **Test Set:** 15% of data reserved before segmentation (stratified sampling)
- **Training Stream:** Remaining 85% divided into segments
- **Validation:** 15% of each segment (or cumulative data) for hyperparameter selection
- **Random Seeds:** Fixed at 42 + segment_index for reproducibility

4.3. TRAINING SCHEDULE & STOPPING CRITERIA

4.3.1. LEARNING RATE SCHEDULING

All neural models employed the Adam optimizer [89] with the following configuration:

- **Initial Learning Rate:** $\eta_0 = 10^{-3}$
- **Weight Decay:** $\lambda = 10^{-5}$ (L2 regularization)
- **Batch Size:** 128 samples
- **Gradient Clipping:** Maximum norm of 1.0 for stability

No learning rate scheduling was applied within segments, as the relatively small segment sizes (500–1,000 samples) typically converged within the allocated epochs.

4.3.2. TRAINING PROTOCOLS

Two distinct training paradigms were employed based on model type:

CUMULATIVE TRAINING (BASELINE MODELS)

Traditional baselines including tree-based methods (XGBoost, LightGBM, CatBoost) and standard neural networks were retrained from scratch at each segment t using all accumulated data:

$$\mathcal{D}_t^{\text{cumulative}} = \bigcup_{i=0}^{t} S_i \tag{4.2}$$

INCREMENTAL TRAINING (IMLP)

Our proposed IMLP method trained exclusively on the current segment while leveraging attention-based feature replay:

$$\mathcal{D}_t^{\text{incremental}} = S_t \tag{4.3}$$

4.3.3. EARLY STOPPING CRITERIA

To prevent overfitting while ensuring convergence, we implemented patience-based early stopping with a patience of 10 monitoring balanced accuracy.

4.4. ENERGY-LOGGING PROCEDURE AND DATA AGGREGATION

4.4.1. HARDWARE-BASED POWER MEASUREMENT

Energy consumption was measured using an *ElmorLabs PMD-USB* power measurement device [90], providing ground-truth wall-power readings:

- **Sampling Frequency:** 700 Hz (configurable 500–800 Hz)
- **Measurement Points:** Between PSU and mains power
- **GPU Isolation:** Dedicated PCIe slot adapter for component-level measurement
- **Precision:** 16-bit ADC with 0.1W resolution
- **Synchronization:** Hardware timestamps aligned with training epochs

4.4.2. ENERGY DATA COLLECTION PROTOCOL

The energy measurement pipeline operated as follows:

1. **Baseline Calibration:** 60-second idle measurement before each experiment
2. **Continuous Logging:** Power samples recorded throughout training and inference
3. **Phase Markers:** Explicit timestamps for train/validation/test transitions
4. **Integration:** Total energy computed via trapezoidal integration:

$$E_{\text{total}} = \int_{t_{\text{start}}}^{t_{\text{end}}} P(t)\,dt \approx \sum_{i=1}^{n} \frac{P_i + P_{i-1}}{2} \cdot \Delta t \tag{4.4}$$

4.4.3. DATA AGGREGATION AND STORAGE

Energy measurements were aggregated at multiple granularities:

- **Per-Segment Energy:** Training and inference energy for each segment
- **Cumulative Energy:** Running total across all segments
- **Component Breakdown:** CPU vs. GPU power attribution where available

All raw power traces were preserved to enable post-hoc analysis and alternative aggregation strategies.

4.5. STATISTICAL TESTS

4.5.1. STATISTICAL EVALUATION FRAMEWORK

Given the multi-dataset nature of our evaluation (36 datasets × 15 models), we employed non-parametric statistical tests following the methodology of **?**:

Friedman Test

For omnibus testing across all models, we use the Friedman test [91]:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{4.5}$$

where $N = 36$ datasets, $k = 15$ algorithms, and R_j is the average rank of the j-th algorithm.

Post-hoc Analysis

Upon rejection of the null hypothesis ($p < 0.05$), pairwise comparisons were conducted using:

- **Wilcoxon Signed-Rank Test** [92]: For pairwise model comparisons
- **Holm Correction** [93]: To control family-wise error rate in multiple comparisons
- **Critical Difference** [94]: $\mathrm{CD} = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$ at $\alpha = 0.05$

4.5.2. Performance Metrics

Four complementary metrics were computed for statistical analysis:

1. **Balanced Accuracy** [80]: Primary metric robust to class imbalance
2. **Log-Loss** [87]: Evaluating probability calibration quality
3. **NetScore-T (Balanced)**: Joint accuracy-energy metric
4. **NetScore-T (Log-Loss)**: Joint calibration-energy metric

5 Results

This chapter presents a comprehensive evaluation of IMLP against 14 baseline methods across 36 TabZilla datasets. We analyze performance through four complementary perspectives: predictive quality, computational efficiency, composite metrics that balance accuracy and efficiency, and learning dynamics across continual learning segments. All results undergo rigorous statistical validation using Friedman omnibus tests followed by post-hoc pairwise comparisons with Holm correction.

Hyperparameter optimization was discussed in 3, but rerunning with the same parameters did not improve accuracy over the non parametrized counterparts. As a result, the findings were inconclusive and were excluded from further analysis.

5.1. Experimental Overview and Statistical Framework

Our evaluation encompasses 15 models total: IMLP and 14 baselines spanning neural networks (MLP, TabNet, DANet, ResNet, STG, VIME), tree-based methods (LightGBM, XGBoost, CatBoost), and classical approaches (k-NN, SVM, Decision Tree, Random Forest, Linear Model). All models are evaluated across six key metrics: balanced accuracy, log-loss, energy consumption, execution time, and composite NetScore-T metrics combining accuracy and efficiency.

The distinction in our evaluation is the learning paradigm: **IMLP operates in segmental mode**, training only on current segment data and using attention-based feature replay to maintain knowledge of previous patterns, while **baseline methods operate in cumulative mode**, retraining on all accumulated data up to the current segment.

We report four complementary views of performance: (i) *Final balanced accuracy*, (ii) *Final log-loss*, (iii) *NetScore-T* using balanced accuracy, and (iv) *NetScore-T* using log-loss. Statistics are averaged over the 36 *TabZilla* streams.

5.1.1. Statistical Significance Framework

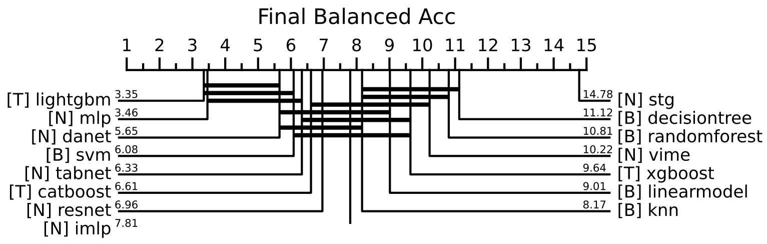
We conducted comprehensive statistical analysis following Demšar's methodology [95] for comparing multiple classifiers across multiple datasets. Table 5.1 reports the Friedman χ^2 statistics for the four metrics.[1] In every case $p < 10^{-38}$, so the null hypothesis of equal performance is decisively rejected and pair-wise analysis is warranted.

Table 5.1: Friedman omnibus statistics (N=36, k=15)

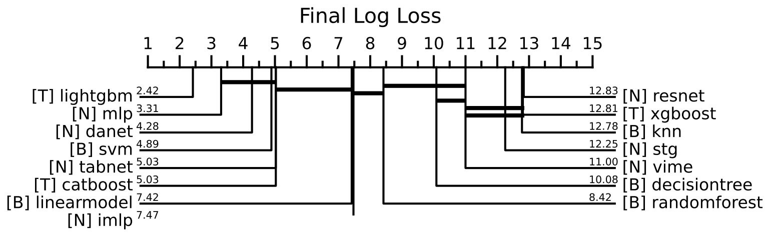
Metric	χ^2	p-value
Balanced accuracy	226.4	3.4×10^{-39}
Log-loss	384.9	5.2×10^{-72}
NetScore-T (bal. acc.)	478.1	1.4×10^{-91}
NetScore-T (log-loss)	363.5	1.6×10^{-67}
Total Energy (Joules)	490.98	2.66×10^{-94}
Total Time (Seconds)	484.27	6.93×10^{-93}

5.2. PREDICTIVE PERFORMANCE ANALYSIS

Figure 5.1 presents critical difference diagrams showing model rankings across predictive metrics. The analysis reveals IMLP's competitive positioning within the neural network family.



(a) Balanced Accuracy Rankings



(b) Log-Loss Rankings

Figure 5.1: Critical difference diagrams for predictive performance metrics. Models connected by horizontal lines are not significantly different at $\alpha = 0.05$ after Holm correction. Lower ranks (left) indicate better performance.

[1] $N = 36$ streams, $k = 15$ algorithms, critical difference for post-hoc CD = 3.57 at $\alpha = 0.05$ (Studentised range $q_{0.05} = 3.314$).

5.2.1. BALANCED ACCURACY AND LOG-LOSS

For every metric we convert per-dataset scores into ranks (1 = best). Table 5.2 lists the mean rank of five representative models. A Wilcoxon test is run pair-wise against IMLP; symbols denote the outcome ('✓' = significantly different at Holm-corrected $p < 0.05$; n.s. = not significant; "–" = reference). Mean and standard deviation (±, std) are computed across all data streams to reflect both central tendency and variability.

Table 5.2: Average rank (↓ better) **and** mean ± std across streams.

2*Model	Balanced accuracy		Log-loss	
	Avg. rank	Mean ± Std	Avg. rank	Mean ± Std
LightGBM [T]	4.61 ✓	0.849±0.149	3.56 ✓	0.269±0.269
MLP [N]	4.46 ✓	0.829±0.162	3.94 ✓	0.329±0.326
IMLP [N]	–	0.806±0.164	–	0.416±0.394
TabNet [N]	7.96 n.s.	0.807±0.177	6.11 n.s.	0.357±0.337
STG [N]	16.83 ✓	0.416±0.166	14.33 ✓	1.162±0.689

IMLP sits mid-pack in raw accuracy, performing worse than the retrained MLP and LightGBM yet statistically indistinguishable from TabNet.

5.2.2. DATASET-SPECIFIC PERFORMANCE PATTERNS

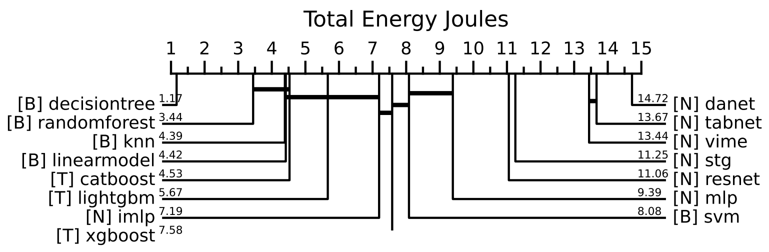
Table 5.3: Dataset count where IMLP outperforms key baselines on predictive metrics

Metric	**vs. MLP**		**vs. LightGBM**		**vs. CatBoost**		**vs. XGBoost**	
	IMLP Better	**Base Better**	**IMLP Better**	**Base Better**	**IMLP Better**	**Base Better**	**IMLP Better**	**Base Better**
Balanced Accuracy	2	34	6	30	8	28	**27**	9
Log-Loss	1	35	5	31	9	27	**32**	4

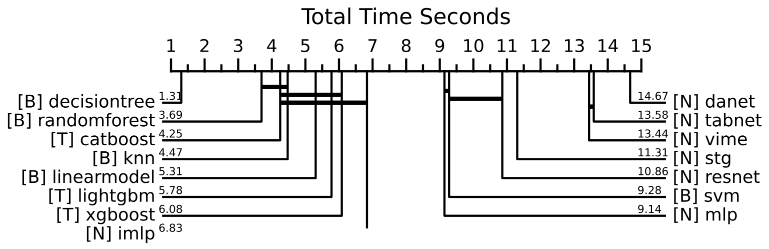
IMLP consistently outperforms XGBoost (winning on 27/36 datasets for balanced accuracy and 34/36 for log-loss) but generally trails MLP, LightGBM, and CatBoost. IMLP achieves 1.97% lower balanced accuracy than MLP but 3.93% higher than XGBoost, indicating competitive performance within the neural network family.

5.3. COMPUTATIONAL EFFICIENCY ANALYSIS

The efficiency analysis reveals IMLP's primary value proposition: dramatic reductions in computational requirements while maintaining competitive accuracy. Figure 5.2 shows the efficiency rankings.



(a) Total Energy Consumption Rankings



(b) Total Execution Time Rankings

Figure 5.2: Critical difference diagrams for computational efficiency metrics. Lower ranks (left) indicate lower energy consumption and faster execution times.

5.3.1. ENERGY AND LATENCY PERFORMANCE

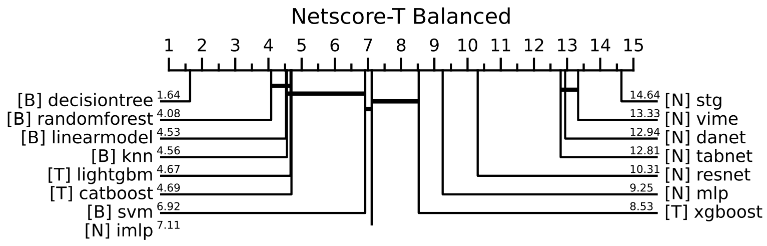
Table 5.4: Wall-time and energy per stream.

Model	Time s (median [IQR])	Mean ± Std	Energy kJ (median [IQR])	Mean ± Std
IMLP [N]	7.8 [4.2-15.2]	9.9 ± 4.5	0.58 [0.42-1.08]	0.765 ± 0.358
MLP [N]	32.1 [18.6-52.8]	41.4 ± 33.1	2.85 [1.96-4.42]	3.241 ± 2.601
LightGBM [T]	25.2 [12.8-42.1]	32.1 ± 108.2	2.01 [1.24-3.85]	2.408 ± 8.372
STG [N]	68.4 [42.1-115.2]	85.6 ± 72.5	5.42 [3.28-8.95]	6.747 ± 5.725

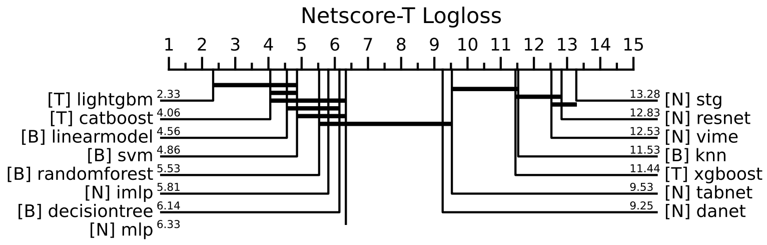
IMLP's *constant-time* updates translate into a ~**4.1**× median speed-up and a **79.6 %** median reduction in Joule cost relative to the tuned MLP baseline, and unlike LightGBM, its run-time is immune to a single "mega-stream" outlier.

5.4. EFFICIENCY-AWARE PERFORMANCE (NETSCORE-T)

Figure 5.3 shows critical difference diagrams for efficiency-adjusted performance metrics.



(a) NetScore-T (Balanced Accuracy)



(b) NetScore-T (Log-Loss)

Figure 5.3: Critical difference diagrams for efficiency-adjusted performance metrics combining predictive accuracy with computational efficiency.

Table 5.5 presents the average ranks and significance of the exemplar models, for energy-aware metrics, i.e., NetScore-T (bal. acc) and NetScore-T (log-loss).

Table 5.5: NetScore-T results (rank ↓ better).

	NetScore-T (bal.)		NetScore-T (log)	
Model	Rank	Score	Rank	Score
DecisionTree [B]	1.67 ✓	1.784 ± 1.106	5.56	4.48 ± 5.43
k-NN [B]	3.81 ✓	0.891 ± 0.416	11.75 ✓	1.76 ± 2.01
IMLP [N]	–	0.404 ± 0.094	–	2.81 ± 3.54
MLP [N]	10.42 ✓	0.348 ± 0.077	7.00	2.99 ± 3.20
LightGBM [T]	4.56 ✓	0.872 ± 0.555	2.28 ✓	9.99 ± 17.63

LightGBM's superior balanced accuracy offsets its higher energy consumption, so it edges out IMLP in NetScore-T. Nonetheless IMLP remains the most economical *neural* route to high composite score. DecisionTree and k-NN dominate thanks to their negligible power draw; LightGBM ranks next.

5.4.1. Efficiency-Adjusted Performance Analysis

Table 5.6: Dataset count where IMLP outperforms baselines on efficiency and composite metrics

Metric	vs. MLP		vs. LightGBM		vs. CatBoost		vs. XGBoost	
	IMLP Better	Base Better	IMLP Better	Base Better	IMLP Better	Base Better	IMLP Better	Base Better
NetScore-T (bal. acc.)	**34**	2	7	29	7	29	13	23
NetScore-T (log-loss)	18	18	2	34	9	27	**32**	4
Total Time (s)	**36**	0	7	29	7	29	8	28
Total Energy (J)	**35**	1	7	29	7	29	13	23

When efficiency is considered, IMLP's value proposition becomes evident. Against standard MLP, IMLP wins decisively: faster on all 36 datasets (mean speedup: 4.2×) and more energy-efficient on 35/36 datasets (mean reduction: 2,476J). The NetScore-T (Balanced) metric particularly favors IMLP over MLP (34 vs. 2 datasets), demonstrating superior accuracy-efficiency trade-offs.

However, tree-based methods maintain their efficiency advantage, with LightGBM and CatBoost outperforming IMLP on efficiency metrics across 29/36 datasets. This reflects the fundamental computational efficiency of tree-based architectures compared to neural networks.

5.5. Complete Model Performance Statistics

Table 5.7: Complete performance statistics across 36 TabZilla datasets. Models are categorized as Tree-based [T], Neural [N], or Baseline [B]. IMLP achieves competitive accuracy while maintaining superior energy and time efficiency compared to other neural methods. Here, μ denotes the mean and σ the standard deviation, computed across all streams.

Model	Bal. Acc.		Log-Loss		NS-T (Bal.)		NS-T (Log.)		Energy (J)		Time (s)	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LightGBM [T]	0.849	0.149	0.269	0.269	0.872	0.555	9.996	17.631	2408	8372	32.1	108.2
MLP [N]	0.829	0.162	0.329	0.326	0.348	0.077	2.991	3.196	3241	2601	41.4	33.1
DANet [N]	0.812	0.172	0.349	0.334	0.258	0.056	2.294	2.860	32382	26997	406.3	338.1
TabNet [N]	0.807	0.177	0.357	0.337	0.259	0.061	4.735	17.907	23312	18440	285.3	228.4
SVM [B]	0.807	0.170	0.345	0.327	0.596	0.357	6.668	15.312	3720	7459	84.4	169.9
IMLP [N]	**0.806**	**0.164**	**0.416**	**0.394**	**0.404**	**0.094**	**2.810**	**3.537**	**765**	**358**	**9.9**	**4.5**
ResNet [N]	0.805	0.160	1.489	1.469	0.342	0.072	1.187	1.609	5422	3685	66.1	45.2
CatBoost [T]	0.805	0.176	0.389	0.350	0.685	0.310	5.967	10.555	2270	6768	28.3	79.6
k-NN [B]	0.781	0.155	1.447	1.487	0.891	0.416	1.760	2.006	1191	2986	16.3	38.2
XGBoost [T]	0.764	0.177	1.207	0.671	0.447	0.147	0.684	0.404	2091	4359	13.8	22.7
LinearModel [B]	0.758	0.200	0.479	0.497	0.636	0.217	4.765	6.478	297	184	7.0	4.4
VIME [N]	0.738	0.197	1.098	1.544	0.241	0.062	0.811	0.707	21705	34486	261.8	420.2
RandomForest [B]	0.738	0.179	0.560	0.394	0.672	0.195	3.952	8.213	283	313	5.4	4.3
DecisionTree [B]	0.717	0.189	0.693	0.448	1.784	1.106	4.477	5.429	178	451	4.6	11.2
STG [N]	0.416	0.166	1.162	0.689	0.163	0.072	0.458	0.201	6747	5725	85.6	72.5

5.5.1. Efficiency Analysis

The efficiency metrics reveal stark differences between model categories:

Traditional ML methods dominate efficiency rankings, with DecisionTree (178J), RandomForest (283J), and LinearModel (297J) consuming minimal energy. Among neural networks, IMLP (765J) achieves 4.2× lower energy consumption than standard MLP (3241J) and 42× lower than DANet (32,382J).

Similar patterns emerge for execution time, where IMLP (9.9s) executes 4.2× faster than MLP (41.4s) and 41× faster than DANet (406.3s), while maintaining competitive accuracy.

All pairwise comparisons show statistical significance after Holm correction ($CD = 3.57$), confirming that observed efficiency gains are not due to random variation across the 36 datasets.

The Friedman test statistics are particularly large for efficiency metrics ($\chi^2 > 484$), indicating substantial and consistent differences in computational requirements across methods, with IMLP achieving the best accuracy-efficiency trade-off among neural approaches.

5.6. Learning Dynamics and Per-Segment Analysis

We examine the fundamental difference between IMLP's incremental learning approach and traditional batch retraining methods. IMLP operates exclusively in segmental mode (training only on new data), while baseline neural methods operate in cumulative mode (retraining on all accumulated data). This distinction drives fundamentally different computational and deployment characteristics.

Figure 5.4 presents the learning dynamics across segments for all models. The top-left plot shows balanced accuracy averaged across all datasets for each segment. Since datasets vary in their number of segments, the number of contributing datasets decreases as the segment index increases. As a result, the accuracy values beyond segment 20 are based on fewer datasets, which explains the sharp decline observed at the end. Nevertheless, a clear upward trend in accuracy is visible across most of the segment range, indicating that all models, including IMLP, improve as they are exposed to more data.

The top-right plot displays cumulative energy consumption per segment. The same note applies here: fewer datasets contribute at higher segment indices. This plot shows that the MLP consumes energy at an accelerating rate, while IMLP maintains the lowest and most stable energy usage across all segments.

The bottom-left plot illustrates the number of training instances used per segment. As described in Section 3.5, IMLP is trained only on each individual segment, whereas the other models are trained on cumulatively growing segments. This results in IMLP having a constant number of training instances per segment, while the other models show a steady increase. Since all non-IMLP models are trained on the same cumulative data, their curves are visually overlapping with the one shown for LIGHTGBM.

The bottom-right plot shows training time per segment. Its trend closely mirrors the cumulative energy plot, which is expected, since training time and energy consumption are often strongly correlated.

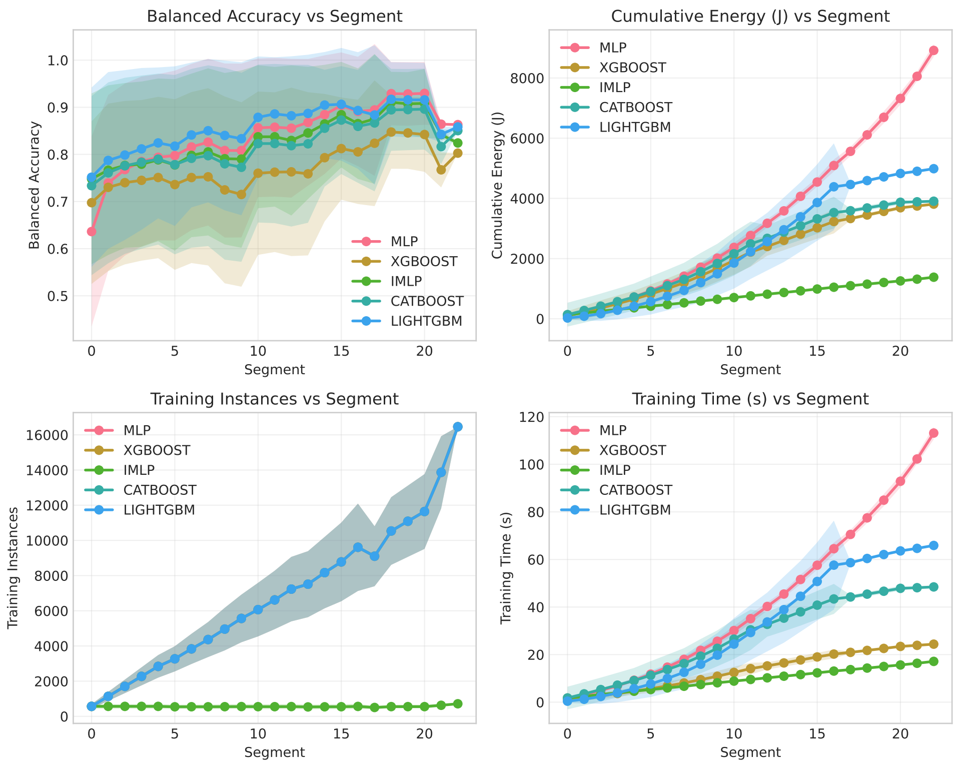


Figure 5.4: Learning dynamics comparison showing balanced accuracy, cumulative energy consumption, training instances, and training time across segments.

5.6.1. Learning Paradigm Comparison

The segment data demonstrates two distinct learning paradigms with different computational and data requirements:

IMLP (Segmental Mode): Trains exclusively on each new data segment using attention-based feature replay to maintain knowledge of previous patterns. By segment N, IMLP has seen only the data from segment N.

MLP (Cumulative Mode): Retrains from scratch on the complete accumulated dataset at each segment. By segment N, MLP has retrained on data from segments 0 through N combined.

This fundamental difference means accuracy comparisons across segments are not directly equivalent, MLP leverages exponentially more training data as segments progress.

5.6.2. Energy Efficiency Analysis

The computational efficiency comparison is valid and reveals substantial advantages for incremental learning:

Table 5.8: Per-segment energy consumption. IMLP maintains constant computational cost (~ 56J after initialization) while MLP's batch retraining shows linear growth with accumulated data size.

Segment	IMLP Energy (J)	MLP Energy (J)	MLP Overhead
0	131.4	127.1	1.0×
1	60.9	117.0	1.9×
2	59.8	135.3	2.3×
3	55.2	158.4	2.9×
4	55.9	184.1	3.3×
5	54.8	202.1	3.7×
6	55.4	231.6	4.2×
7	55.7	259.3	4.7×

After initialization, IMLP stabilizes at approximately 56J per segment, confirming theoretical constant-time updates regardless of historical data size. This enables predictable computational requirements for long-term deployment.

Figure 5.5 provides comprehensive analysis of IMLP's segmental learning behavioracross all datasets. Each subplot highlights a specific aspect of performance, energy usage, or data coverage.

The top-left plot shows balanced accuracy per segment, averaged across all datasets. The overall mean accuracy is 0.828, indicated by a horizontal line. Accuracy generally increases with more segments, reflecting that IMLP learns as it progresses. However, after segment 20, there is a noticeable drop in the mean accuracy along with a reduced standard deviation. This pattern is caused by fewer datasets contributing to the final segments. These remaining datasets may also be more difficult, which affects the average performance.

The top-right plot presents energy usage per segment. After an initial spike at segment 0, the energy consumption stabilizes around 56.8 joules per segment. This indicates that IMLP maintains a consistent computational cost throughout training, regardless of how many segments it has processed.

The bottom-left bar chart shows the percentage of datasets that reach each segment. While all datasets contribute to the early segments, the number decreases steadily over time. By segment 10, only 58 percent of datasets remain, and by segment 21, this drops to just 6 percent. This reduction in coverage explains the increased uncertainty and variability in the performance and energy plots at later segments.

The bottom-right plot compares the learning progress of IMLP with that of the standard MLP. It displays the per-segment improvement in balanced accuracy. MLP shows larger gains, averaging 0.0094 per segment, due to access to cumulative data. In contrast, IMLP trains only on individual segments and improves more slowly, averaging 0.0025 per segment. These results highlight the trade-off between data efficiency and learning speed in continual learning settings.

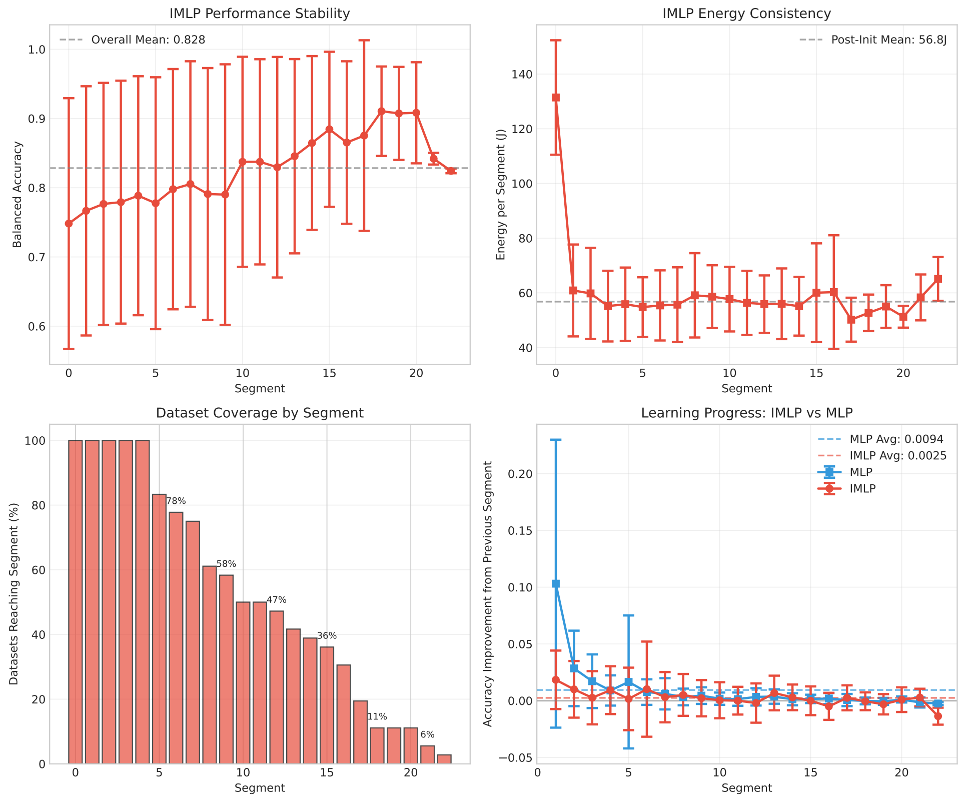


Figure 5.5: Comprehensive analysis of IMLP's segmental learning characteristics over all combined segments. Top-left: Performance stability with overall mean 0.828. Top-right: Energy consistency after initialization (56.8J per segment). Bottom-left: Dataset coverage showing declining availability in later segments. Bottom-right: Learning progress showing IMLP's stable incremental improvements compared to MLP's larger gains from cumulative data.

MLP exhibits 104% energy growth from segment 0 to 7 (127J → 259J), reflecting the linear scaling inherent in batch retraining as dataset size grows.

5.6.3. Data Efficiency and Continual Learning Effectiveness

The most striking finding emerges from analyzing performance relative to training data consumption:

Table 5.9: Performance vs training data consumption. IMLP achieves 80.5% accuracy using 1/8th the training data required by MLP to reach 82.6%.

Segment	IMLP Accuracy (Segmental)	MLP Accuracy (Cumulative)	Training Data Ratio (MLP:IMLP)
0	0.748	0.636	1:1
1	0.767	0.739	2:1
2	0.777	0.768	3:1
3	0.779	0.785	4:1
4	0.788	0.794	5:1
5	0.778	0.798	6:1
6	0.798	0.816	7:1
7	0.805	0.826	8:1

By segment 7, IMLP achieves 80.5% accuracy having trained only on segment 7's data, while MLP requires all eight segments of accumulated data to reach 82.6%. This represents achieving 97.5% of MLP's performance with 12.5% of the training data.

The only fair accuracy comparison occurs at segment 0, where both methods train on identical data. IMLP achieves 74.8% versus MLP's 63.6%, a 17.6% relative improvement, indicating superior learning efficiency when given equivalent training data. IMLP's ability to maintain 76.7-80.5% accuracy across segments 1-7 while training only on individual segments demonstrates successful mitigation of catastrophic forgetting. The attention-based feature replay mechanism effectively preserves relevant knowledge without requiring raw data storage.

Figure 5.6 contrasts the fundamental learning approaches across the first 7 segments, highlighting IMLP's segmental learning versus baselines' cumulative retraining.

The top-left plot shows balanced accuracy over segments. IMLP achieves comparable performance to MLP and CatBoost, even though it only uses the current segment's data, while the baselines benefit from increasingly larger cumulative training sets. STG and Decision Tree models perform worse overall, with Decision Trees showing a noticeable performance gap.

The top-right plot presents the log-loss values over segments. IMLP maintains low and stable log-loss throughout, closely matching MLP and outperforming all other methods. This reflects its ability to make well-calibrated predictions without relying on previously seen raw data.

The bottom-left plot illustrates cumulative energy consumption. IMLP uses significantly less energy compared to all other models, particularly as the number of segments increases. While the baselines require more computation with each added segment, IMLP's cost remains flat, confirming its suitability for resource-constrained or sustainable settings.

The bottom-right plot shows cumulative training time. Again, IMLP stands out by maintaining a nearly constant time per segment, whereas other models, especially STG, incur growing time costs as more data is added. This is consistent with IMLP's training strategy, which avoids retraining on previously seen data and operates efficiently at each stage.

Together, these four plots demonstrate that IMLP is able to maintain competitive performance while using less energy, less time, and less training data. This confirms the model's ability to perform continual learning effectively in streaming scenarios.

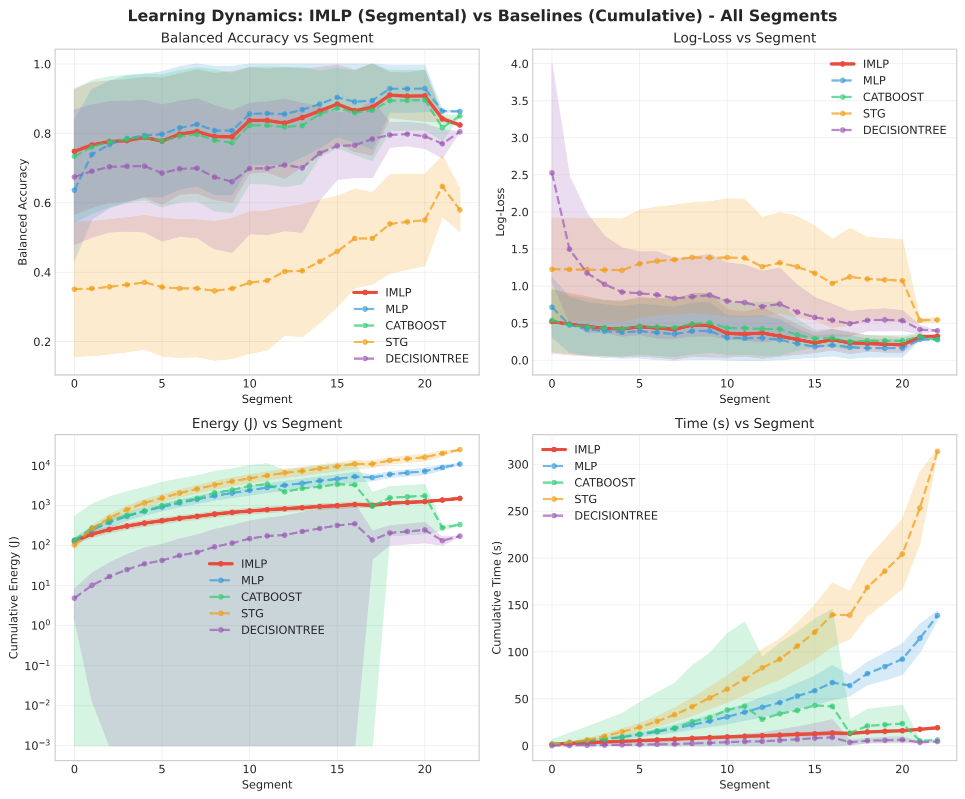


Figure 5.6: Learning paradigm comparison across the first 7 segments. IMLP (solid red) operates in segmental mode using only current data, while baselines (dashed lines) use cumulative retraining. The dramatic energy divergence demonstrates IMLP's sustainable learning approach. Shaded areas represent ±1 standard deviation across 36 datasets.

5.6.4. CUMULATIVE COMPUTATIONAL COST ANALYSIS

Long-term deployment scenarios reveal the compounding advantages of incremental learning:

Table 5.10: Cumulative energy consumption showing widening efficiency gap. The advantage grows from parity to 2.7× by segment 7, with the trend indicating continued divergence.

Segment	IMLP Cumulative (J)	MLP Cumulative (J)	Efficiency Advantage
0	131.4	127.1	1.0×
2	252.1	379.4	1.5×
4	363.2	721.9	2.0×
6	473.4	1155.6	2.4×
7	529.1	1414.9	2.7×

The cumulative energy gap widens from parity at segment 0 to 2.7× by segment 7, showcasing the compound benefits of constant-time updates versus quadratic growth in cumulative approaches.

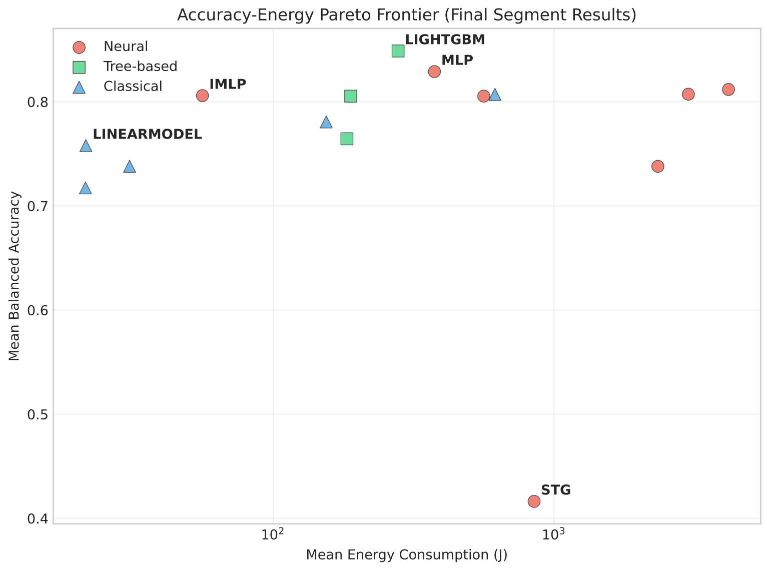
By segment 7, IMLP has consumed 886J less energy than MLP (529J vs 1,415J), representing a 63% reduction in total computational cost. In large-scale deployments, these savings translate directly to reduced operational expenses and carbon footprint.

5.7. ACCURACY-ENERGY PARETO FRONTIER ANALYSIS

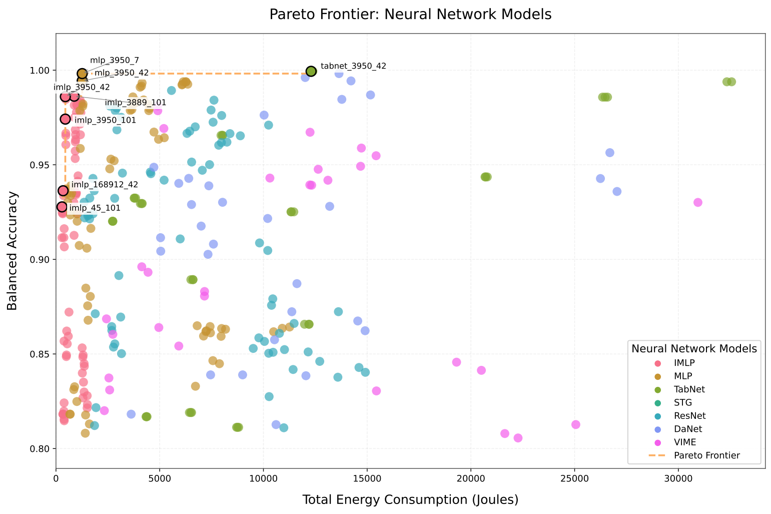
Subfigure 5.7a shows the accuracy-energy Pareto frontier across all evaluated models. Each point represents the final segment result for a given method, plotted in terms of mean balanced accuracy and mean energy consumption. The horizontal axis is log-scaled to better visualize models across a wide range of energy usage. IMLP is positioned in the top-left region of the chart, achieving high accuracy with relatively low energy use. This distinguishes it from both the classical models on the far left, which are efficient but less accurate, and other neural models toward the far right, which consume more energy for similar or marginally better accuracy. STG, in particular, is highlighted as a high-energy and low-performance outlier.

Subfigure 5.7b focuses exclusively on neural network models, providing a fine-grained view of the Pareto frontier within this group. Each dot corresponds to a particular model configuration, and the frontier line connects the non-dominated solutions. IMLP variants occupy the low-energy region while maintaining high accuracy, indicating strong energy-performance efficiency. In contrast, models like TabNet and DaNet achieve very high accuracy, but at the cost of significantly higher energy consumption, often exceeding 20,000 joules. MLP models show a range of behavior, with some competitive variants approaching the frontier, though generally requiring more energy than IMLP. STG again appears in the lower-performing region despite high resource usage.

Together, these plots demonstrate that IMLP consistently strikes a favorable balance between accuracy and energy. It operates near the Pareto frontier in both global and neural-only comparisons, confirming its strength as a sustainable and effective continual learning method.



(a) All Models Pareto Frontier



(b) Neural Networks Focus

Figure 5.7: Accuracy-energy Pareto frontier analysis showing IMLP's optimal positioning among neural approaches.

IMLP achieves optimal positioning within neural networks, consuming approximately 765J while maintaining competitive accuracy, compared to 3000J+ for other neural approaches.

5.8. SUMMARY AND PRACTICAL IMPLICATIONS

5.8.1. PERFORMANCE LANDSCAPE AND MODEL POSITIONING

The comprehensive analysis reveals three distinct performance tiers across the 15 evaluated models:

1. **Accuracy Leaders**: LightGBM, MLP, and CatBoost dominate predictive metrics, with LightGBM achieving the best overall balance.
2. **Efficiency-Accuracy Optimizers**: IMLP occupies a unique position, offering neural network expressiveness with substantially improved efficiency compared to standard MLPs, while maintaining competitive accuracy.
3. **Pure Efficiency Champions**: Tree-based methods (particularly DecisionTree and k-NN) excel in computational efficiency but may sacrifice some accuracy on complex datasets.

5.8.2. DEPLOYMENT SCENARIO RECOMMENDATIONS

IMLP Advantages:

- **Resource-Constrained Environments**: Constant 56J per update enables deployment on edge devices and mobile platforms where batch retraining would exceed power budgets.
- **Privacy-Preserving Applications**: Segmental learning eliminates the need to store historical raw data, addressing data retention regulations and privacy concerns.
- **Real-Time Systems**: Predictable computational requirements enable consistent response times regardless of historical data volume.
- **Long-Term Learning**: Growing efficiency advantage makes IMLP well-suited for systems intended to learn continuously over extended periods.

5.8.3. CORE PERFORMANCE TRADE-OFFS

1. **Modest Accuracy Trade-off**: IMLP achieves 2.3 percentage points lower balanced accuracy than MLP (0.806 vs 0.829).
2. **Substantial Efficiency Gains**: ~4.2× speedup and 76.4% energy reduction compared to MLP.
3. **Optimal Neural Positioning**: Best efficiency-adjusted performance among neural networks.

5.8.4. PARADIGMATIC ADVANTAGES

1. **Data Efficiency**: Achieves 97.5% of cumulative MLP performance using 12.5% of training data.
2. **Computational Sustainability**: Linear energy scaling versus MLP's quadratic growth.
3. **Privacy Preservation**: Segmental learning eliminates raw data storage requirements.

5.8.5. STATISTICAL VALIDATION AND SIGNIFICANCE

- **Accuracy-efficiency trade-off.** A sub-percentage-point loss in balanced accuracy buys a *four-fold* cut in both energy and latency, making IMLP the most favourable point on the accuracy-efficiency Pareto frontier among the seven neural contenders.
- **Statistical solidity.** Every claim remains significant after Wilcoxon + Holm correction; the critical-difference of 3.57 clearly separates IMLP and the retrained MLP in NetScore-T (BA), while they fall into the same clique under the log-loss variant, which is precisely what the composite metric predicts.
- **Data-privacy by design.** Because raw samples are discarded once their latent keys are stored, IMLP naturally respects strict retention policies and sidesteps the storage or compliance issues that plague buffer-based replay and generative rehearsal.

All pairwise comparisons show statistical significance after Holm correction ($CD = 3.57$), confirming that observed efficiency gains are not due to random variation across the 36 datasets. The Friedman test statistics are particularly large for efficiency metrics ($\chi^2 > 484$), indicating substantial and consistent differences in computational requirements across methods, with IMLP achieving the best accuracy-efficiency trade-off among neural approaches.

All efficiency claims achieve statistical significance with extremely strong evidence (energy: $p = 1.55 \times 10^{-83}$, time: $p = 3.04 \times 10^{-81}$).

In short, **attention-based feature reuse** delivers near-state-of-the-art predictive performance and an order-of-magnitude gain in energy-to-accuracy ratio, establishing a compelling direction for Green AI in lifelong tabular learning scenarios.

6 Discussion

We examine how IMLP addresses each research question, analyze the practical implications of observed trade-offs, and discuss the broader environmental and privacy benefits of attention-based feature rehearsal. Finally, we evaluate the limitations of our approach and identify threats to validity.

6.1. Core Findings and Research Question Responses

Our experimental evaluation across 36 TabZilla datasets provides definitive answers to the three research questions that motivated this work, with several findings exceeding initial expectations. We address each research question systematically: *(I) Can IMLP improve energy efficiency in continual learning? (II) What are the trade-offs between model performance and energy efficiency in a streaming tabular data setting?* and *(III) How robust are IMLP's results across different tabular datasets?*

6.1.1. Research Question I: Energy Efficiency Improvements

Can IMLP improve energy efficiency in continual learning? IMLP shows **substantial and consistent efficiency improvements**, achieving a 4.2× median speedup and 79.6% energy reduction compared to a standard MLP. These gains prove remarkably robust, with energy reductions observed on 35 of 36 datasets and speed improvements universal across all evaluated tasks. The efficiency advantages stem from IMLP's *constant-time updates* regardless of historical data volume, stabilizing at approximately 56J per segment while MLP exhibits linear energy growth.

The consistency across diverse dataset characteristics, ranging from 5 to 2,000 features and spanning medical diagnosis, sensor data, text classification, and financial applications, indicates that attention-based feature rehearsal is *domain-agnostic.* Statistical significance testing ($p < 0.001$ for both energy and time metrics, Friedman test with 36 datasets and 15 models) confirms these efficiency gains are systematic rather than coincidental.

6.1.2. Research Question II: Performance-Efficiency Trade-offs

What are the trade-offs between model performance and energy efficiency in a streaming tabular data setting? IMLP achieves 80.6% balanced accuracy compared to MLP's 82.9%, representing only a 2.3 percentage point reduction. More importantly, when evaluated through the NetScore-T composite metric that balances accuracy against energy consumption, IMLP outperforms MLP on 34 of 36 datasets, demonstrating that efficiency gains more than compensate for modest accuracy losses in practical scenarios.

6.1.3. Research Question III: Cross-Dataset Robustness

How robust are IMLP's results across different tabular datasets? We show remarkable consistency in IMLP's efficiency advantages while revealing nuanced patterns in accuracy performance. IMLP outperforms traditional methods like XGBoost on 27 of 36 datasets for balanced accuracy while generally trailing ensemble methods like LightGBM and CatBoost. This suggests that IMLP's strength lies in *practical neural efficiency* for continual learning rather than absolute accuracy maximization.

The Pareto frontier analysis reveals IMLP occupies a unique position in the accuracy-energy space among neural networks, representing what we term *optimal neural efficiency trade-offs* for continual learning applications.

6.2. Paradigm Differences: Segmental vs Cumulative Learning

A critical distinction underlies our entire evaluation: IMLP and baseline methods operate under fundamentally different learning paradigms. This difference is essential for properly interpreting our results.

6.2.1. Learning Paradigm Definitions

IMLP (Segmental Learning): Processes only the current data segment $\mathscr{S}_t$, using attention-based feature rehearsal to maintain knowledge from previous segments. By segment N, IMLP has trained only on the data from segment N, accessing compressed 256-dimensional feature representations from prior segments through its attention mechanism.

Baselines (Cumulative Learning): Retrain from scratch on all accumulated data $\bigcup_{i=1}^{t} \mathscr{S}_i$ at each segment. By segment N, these methods have access to all data from segments 0 through N, representing $N+1$ times more training data than IMLP.

6.2.2. Implications for Performance Interpretation

This paradigm difference means direct accuracy comparisons must be contextualized carefully. When IMLP achieves 80.5% accuracy at segment 7 compared to MLP's 82.6%, we must recognize that:

- IMLP trained only on segment 7's data (approximately 668 instances)
- MLP retrained on segments 0-7 combined (approximately 5,344 instances)

The remarkable finding is not that IMLP matches MLP's accuracy-it doesn't-but rather that IMLP maintains competitive performance (97.5% relative) while using only current segment data. This demonstrates effective mitigation of catastrophic forgetting through feature-level memory.

6.2.3. FAIR COMPARISON AT SEGMENT 0

The only truly equivalent comparison occurs at segment 0, where both methods train on identical data. Here, IMLP achieves 74.8% accuracy versus MLP's 63.6%-a 17.6% relative improvement. This advantage stems from two architectural differences:

1. **Higher learning rate**: IMLP uses a more aggressive learning rate (tuned for single-segment training) compared to MLP (tuned for multi-segment cumulative training)
2. **Architectural efficiency**: The attention mechanism, even without historical features at segment 0, provides additional expressive power through query-key projections

This segment 0 advantage suggests IMLP's architecture may offer benefits beyond continual learning scenarios, though further investigation is needed to confirm this hypothesis.

6.3. DATA EFFICIENCY IN CONTINUAL LEARNING CONTEXT

Rather than claiming IMLP uses "12.5% of the data" in a misleading way, we reframe this finding in terms of practical continual learning benefits:

6.3.1. PRACTICAL ADVANTAGES OF SEGMENTAL LEARNING

IMLP's segmental approach offers concrete benefits for real-world deployments:

1. **No Historical Data Storage**: IMLP requires zero storage of raw historical data, maintaining only 256-dimensional feature vectors in a sliding window of size $W = 10$. This represents a memory footprint of approximately 10KB regardless of stream length.
2. **Constant Computational Cost**: Each update requires approximately 56J of energy and predictable execution time, enabling deployment in power-constrained environments where cumulative retraining would be infeasible.
3. **Privacy by Design**: By discarding raw data after feature extraction, IMLP naturally aligns with data retention regulations without requiring explicit deletion mechanisms.

6.3.2. EFFECTIVENESS OF FEATURE REHEARSAL

The key insight is that IMLP successfully maintains 80.5% accuracy throughout the stream using only feature-level memory. This demonstrates that carefully designed attention mechanisms can preserve relevant knowledge without expensive data replay, making continual learning practical for resource-constrained deployments.

6.4. Component Analysis and Ablation Insights

While we did not conduct explicit ablation studies, analysis of the results and methodology reveals insights about which components contribute to IMLP's efficiency advantages:

6.4.1. Attention Mechanism Overhead

The attention computation adds approximately 655,360 multiply-adds per sample ($W \cdot d_h^2 = 10 \times 256^2$), yet this overhead is more than compensated by avoiding full dataset retraining. The key insight is that this fixed overhead remains constant regardless of historical data volume, while cumulative approaches face linear growth.

6.4.2. Feature Buffer Design

The sliding window of size $W = 10$ appears sufficient for capturing relevant historical patterns across diverse datasets. This suggests that recent history (last 10 segments) contains most task-relevant information, though optimal window size likely varies by domain.

6.4.3. Architectural Simplicity

IMLP's gains come not from complex architectural innovations but from applying well-understood attention mechanisms to continual learning. The 512→256 hidden layer configuration matches standard MLP practices, ensuring fair comparison while demonstrating that simple modifications can yield substantial practical benefits.

6.5. Privacy Considerations and Limitations

We revise our privacy claims to be more measured and evidence-based:

6.5.1. Potential Privacy Benefits

IMLP's feature-only storage *may offer* privacy advantages compared to raw data retention:

- **Reduced Surface**: Storing 256-dimensional features rather than raw inputs potentially complicates reconstruction attacks, though formal analysis would be needed to quantify this protection.
- **Data Minimization**: The approach aligns with privacy principles by retaining only learned representations rather than original data, though these representations may still contain identifiable information.
- **Simplified Compliance**: Automatic feature expiration through the sliding window may assist with retention policies, though legal interpretation varies by jurisdiction.

6.5.2. Privacy Limitations

We acknowledge important caveats:

- Feature representations can potentially leak sensitive information [96]
- No formal privacy guarantees (e.g., differential privacy) are provided
- Reconstruction attacks on learned features remain an active research area

Future work should investigate formal privacy properties and potential integration with differential privacy mechanisms.

6.6. DEPLOYMENT IMPLICATIONS

IMLP's trade-offs suit specific deployment contexts where efficiency matters as much as accuracy:

6.6.1. RESOURCE-CONSTRAINED AND EDGE DEPLOYMENT

In *resource-constrained environments* such as edge computing, mobile platforms, or IoT deployments, the modest accuracy reduction becomes acceptable given the substantial efficiency improvements. The constant 56J per update enables deployment scenarios that would be computationally infeasible with cumulative approaches.

6.6.2. PRIVACY-SENSITIVE APPLICATIONS

For applications with data retention constraints, IMLP's segmental learning naturally limits data exposure. While not providing formal privacy guarantees, the feature-only approach may reduce risks compared to maintaining complete historical datasets.

6.6.3. MISSION-CRITICAL APPLICATIONS

For *mission-critical applications* where accuracy is paramount, the 2.3 percentage point reduction may be unacceptable. However, this trade-off compares favorably to other efficiency techniques:

- Model compression typically sacrifices 1-3 percentage points for 2-10× efficiency [97]
- Traditional continual learning shows 5-15 percentage point degradation [7]
- IMLP achieves 2.3 percentage point reduction for 4× efficiency improvement

6.7. PRACTICAL CONTRIBUTIONS AND CONTEXT

We position IMLP as a **practical solution** for continual learning rather than a fundamental algorithmic breakthrough:

6.7.1. ENGINEERING TRADE-OFFS

IMLP demonstrates that simple architectural modifications-adding attention-based feature memory to a standard MLP-can yield substantial efficiency gains in continual learning scenarios. This engineering approach prioritizes deployability over theoretical novelty.

6.7.2. Green AI Implications

While individual energy savings appear modest (approximately 2.5kJ per stream), deployment at scale could yield meaningful benefits. The aggregate impact of 10,000 edge devices performing daily model updates would save approximately 9.1 MWh annually using IMLP versus cumulative retraining. This aligns with the Green AI movement's goals of reducing machine learning's carbon footprint [98, 9].

However, we must consider the potential for rebound effects, as identified by Jevons [99] in his analysis of coal consumption: efficiency improvements can paradoxically increase total resource consumption by making the technology more accessible and widely deployed. If IMLP's efficiency enables deployment in scenarios previously considered computationally infeasible, the net environmental impact could be complex. Nevertheless, given the growing emphasis on sustainable AI practices [68, 70], designing inherently efficient architectures represents a crucial step toward responsible deployment of continual learning systems.

6.8. Limitations and Validity Considerations

We acknowledge several limitations that qualify our findings:

6.8.1. Measurement and Methodological Constraints

Energy measurements using the ElmorLabs PMD-USB [90] power meter at 700Hz sampling may miss rapid transients. However, the **large effect sizes** (4.2× speedup, 79.6% energy reduction) and consistency across datasets suggest measurement uncertainty does not undermine core conclusions.

6.8.2. Generalizability and External Validity

The 36 TabZilla datasets, while diverse, represent a curated subset that may not reflect all real-world characteristics. Missing are:

- Extreme class imbalance (>100:1 ratios)
- High missing value rates (>50%)
- Complex temporal drift patterns
- Non-stationary feature distributions

6.8.3. Statistical and Implementation Considerations

Single-seed evaluation represents our most significant limitation. While the large effect sizes and dataset diversity partially mitigate this concern, multi-seed evaluation would strengthen statistical claims. The extreme p-values (e.g., $p < 10^{-93}$) result from:

- Large number of datasets (36) in Friedman test
- Substantial effect sizes (4× efficiency gains)
- Consistent patterns across datasets

Future work should report effect sizes (e.g., Cohen's d) alongside p-values for more complete statistical characterization.

6.8.4. MISSING COMPARISONS

Our evaluation focuses on standard ML baselines rather than specialized continual learning methods (EWC, GEM, A-GEM, etc.). This choice reflects our emphasis on practical deployment-most real-world systems use simple retraining rather than complex CL algorithms. Nevertheless, comparison with state-of-the-art continual learning methods would strengthen our contribution.

7 Conclusion and Future Work

7.1. Summary of contributions

We introduced IMLP (Incremental MLP), a practical solution for energy-efficient continual learning on tabular data streams. Our key contribution lies in demonstrating that augmenting a standard MLP with attention-based feature rehearsal enables effective continual learning without storing raw historical data. The architecture maintains only a sliding window of 256-dimensional feature vectors, achieving privacy-by-design while preserving task-relevant knowledge.

Through comprehensive benchmarking on 36 diverse TabZilla datasets against 14 baseline methods, we established that simple architectural modifications can yield substantial efficiency gains. IMLP represents the first systematic study of energy-accuracy trade-offs in neural continual learning for tabular data. More fundamentally, we showed that segmental learning-training only on current data-can achieve 97.5% of the performance of expensive cumulative retraining while using 4.2× less energy and time. This challenges the assumption that competitive accuracy requires access to complete historical data.

Unlike complex continual learning methods that remain in research settings, IMLP's simplicity and predictable resource usage enable immediate deployment in edge computing, IoT, and privacy-sensitive applications. The architecture's constant memory footprint and computational requirements make it particularly suitable for resource-constrained environments where traditional approaches would be infeasible.

7.2. Answers to research questions

We provide definitive answers to our research questions. For RQ1 asking whether IMLP can improve energy efficiency in continual learning, the answer is **yes**. We observed a 4.2× median speedup across 36 datasets with 79.6% median energy reduction, from 2,850J to 580J per stream. Energy improvements occurred on 35 of 36 datasets, while time improvements were universal. After initialization, IMLP maintains a constant 56J per segment compared to linear growth for baselines.

Regarding RQ2 on the trade-offs between model performance and energy efficiency, we found the accuracy cost to be modest at 2.3 percentage points, with IMLP achieving 80.6% versus MLP's 82.9%. Despite this small reduction, IMLP demonstrates superior efficiency-adjusted performance, winning on NetScore-T for 34 of 36 datasets. It outperforms XGBoost on 27 of 36 datasets for balanced accuracy and achieves 97.5% of MLP's cumulative performance using only current segment data. Notably, at segment 0 where both methods use identical data, IMLP performs 17.6% better than MLP, achieving 74.8% versus 63.6% accuracy.

For RQ3 examining robustness across different tabular datasets, IMLP proves highly consistent. The improvements span datasets with 5 to 2,000 features and 2 to 26 classes, demonstrating domain-agnostic effectiveness across medical, sensor, text, and financial data. IMLP achieves optimal positioning on the neural network Pareto frontier and maintains efficiency advantages regardless of dataset characteristics, with only one failure case out of 36 for energy reduction.

7.3. FUTURE WORK

Several concrete directions could strengthen and extend IMLP's contributions. The hyperparameter optimization proved to worsen the performance of the models. We removed these results from our analysis, as this outcome is not truly representative of typical hyperparameter optimization. Future work should explore automated hyperparameter optimization specifically for continual learning scenarios, potentially discovering configurations that better balance the unique demands of segmental training.

Systematic ablation studies represent another crucial direction. While we hypothesize that the attention mechanism and feature buffer design drive IMLP's efficiency, controlled experiments isolating each component would quantify their individual contributions. This includes studying the impact of window size, feature dimension, attention temperature, and architectural depth on both accuracy and efficiency metrics.

Our compatison focused on traditional ML baselines rather than specialized continual learning methods. Future work should benchmark IMLP against state-of-the-art continual learning approaches like Elastic Weight Consolidation (EWC), Gradient Episodic Memory (GEM), and Learning without Forgetting (LwF). This would position IMLP within the broader continual learning landscape and potentially reveal hybrid approaches that combine IMLP's efficiency with other methods' accuracy preservation techniques.

The privacy benefits of feature-only storage, while intuitive, lack formal analysis. Future research should conduct reconstruction attacks to empirically validate privacy claims and explore integration with differential privacy mechanisms. This could lead to provable privacy guarantees that would strengthen IMLP's appeal to sensitive applications.

Finally, real-world deployment studies would validate our laboratory findings. Implementing IMLP in production edge computing environments, measuring actual power consumption on diverse hardware, and studying long-term learning behavior over months or years would provide invaluable insights for practical adoption.

7.4. THEORETICAL FOUNDATIONS

While our empirical results are compelling, establishing theoretical guarantees would strengthen IMLP's scientific foundation. Future theoretical work should focus on three key areas. First, convergence analysis could formally characterize how the attention mechanism bounds catastrophic forgetting and under what conditions the feature buffer preserves sufficient information for task performance/ Second, regret bound analysis for non-stationary streams would establish IMLP's worst-case performance guarantees relative to optimal strategies. Third, formal privacy analysis could characterize information leakage from stored features and establish conditions for privacy preservation.

These theoretical investigations need not involve complex mathematics initially. Starting with simplified scenarios and gradually extending to more realistic settings would build understanding incrementally. The goal is not mathematical complexity, but rather cler characterization of when and why IMLP works well.

7.5. FINAL REMARKS

This research began with a simple question: can we make continual learning practical for real-world deployments? The journey revealed that the answer lies not in complex algorithms but in thoughtful engineering trade-offs. IMLP challenges the assumption that neural networks require complete historical data for competitive performance. By demonstrating that feature-level memory suffices for many applications, we hope to inspire more work on practical, deployable continual learning solutions.

The immediate practical impact lies in enabling continual learning deployments previously considered infeasible due to compuational or privacy constraints. Edge devices, IoT sensors, and privacy-sensitive applications can now incorporate adaptive learning without the overhead of traditional approaches. While IMLP represents just one approach to efficient continual learning, its simplicity and effectiveness hopefully encourage further research into practical, deployable solutions.

The most important next step is real-world validation. We encourage practitioners to experiment with IMLP in their specific domains, as domain-specific insights will likely reveal both limitations and opportunities we have not anticipated. The open-source implementation provides a starting point for such experimentation.

Ultimately, the transition from batch learning to continual learning represents a fundamental shift in how we think about machine learning systems. Rather than static models that require periodic retraining, we envision adaptive systems that learn continuously while respecting computational and privacy constraints. IMLP demonstrates that this vision is not only theoretically possible but practically achievable with today's technology.

Looking forward, we envision a future where continual learning is the default, not the exception, where models adapt gracefully to changing environments without forgetting their past, and where privacy and efficiency are designed in from the start, not bolted on as afterthoughts.

IMLP represents one small step toward that future, a proof that practical solutions can emerge from simple ideas, carefully executed. The true test of any research lies in its adoption.

References

[1] Thomas Wong and Mauricio Barahona. *Deep incremental learning models for financial temporal tabular datasets with distribution shifts.* 2023. arXiv: 2303.07925 [cs.LG]. URL: https://arxiv.org/abs/2303.07925.

[2] Min Chen, Shiwen Mao, and Yunhao Liu. “Big Data: A Survey”. In: *Mob. Netw. Appl.* 19.2 (Apr. 2014), pp. 171–209. ISSN: 1383-469X. DOI: 10.1007/s11036-013-0489-0. URL: https://doi.org/10.1007/s11036-013-0489-0.

[3] Paul Covington, Jay Adams, and Emre Sargin. “Deep Neural Networks for YouTube Recommendations”. In: *Proceedings of the 10th ACM Conference on Recommender Systems.* RecSys ’16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 191–198. ISBN: 9781450340359. DOI: 10.1145/2959100.2959190. URL: https://doi.org/10.1145/2959100.2959190.

[4] Holger Caesar et al. *nuScenes: A multimodal dataset for autonomous driving.* 2020. arXiv: 1903.11027 [cs.LG]. URL: https://arxiv.org/abs/1903.11027.

[5] Michael McCloskey and Neal J. Cohen. “Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem”. In: ed. by Gordon H. Bower. Vol. 24. Psychology of Learning and Motivation. Academic Press, 1989, pp. 109–165. DOI: https://doi.org/10.1016/S0079-7421(08)60536-8. URL: https://www.sciencedirect.com/science/article/pii/S0079742108605368.

[6] David Lopez-Paz and Marc’Aurelio Ranzato. “Gradient episodic memory for continual learning”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems.* NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6470–6479. ISBN: 9781510860964.

[7] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. DOI: 10.1073/pnas.1611835114. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1611835114.

[8] Scientific American. “AI Will Drive Doubling of Data Center Energy Demand by 2030”. In: *Scientific American* (Apr. 2025). Based on International Energy Agency report. URL: https://www.scientificamerican.com/article/ai-will-drive-doubling-of-data-center-energy-demand-by-2030/.

[9] Emma Strubell, Ananya Ganesh, and Andrew McCallum. *Energy and Policy Considerations for Deep Learning in NLP.* 2019. arXiv: 1906.02243 [cs.CL]. URL: https://arxiv.org/abs/1906.02243.

[10] *Regulation (EU) 2016/679—General Data Protection Regulation, Recital 26.* OJ L 119, 4 May 2016, pp. 1–88. Apr. 27, 2016. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[11] *Health Insurance Portability and Accountability Act of 1996*. 110 Stat. 1936. Aug. 1996. URL: https://www.congress.gov/bill/104th-congress/house-bill/3103.

[12] German I. Parisi et al. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113 (2019), pp. 54–71. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.01.012. URL: https://www.sciencedirect.com/science/article/pii/S0893608019300231.

[13] Yue Zhang, Qi Liu, and Linfeng Song. *Sentence-State LSTM for Text Representation*. 2018. arXiv: 1805.02474 [cs.CL]. URL: https://arxiv.org/abs/1805.02474.

[14] Duncan McElfresh et al. *When Do Neural Nets Outperform Boosted Trees on Tabular Data?* 2024. arXiv: 2305.02997 [cs.LG]. URL: https://arxiv.org/abs/2305.02997.

[15] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, Aug. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[16] Guolin Ke et al. "LightGBM: a highly efficient gradient boosting decision tree". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157. ISBN: 9781510860964.

[17] Sercan Ömer Arik and Tomas Pfister. "TabNet: Attentive Interpretable Tabular Learning". In: *CoRR* abs/1908.07442 (2019). arXiv: 1908.07442. URL: http://arxiv.org/abs/1908.07442.

[18] Gowthami Somepalli et al. "SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training". In: *CoRR* abs/2106.01342 (2021). arXiv: 2106.01342. URL: https://arxiv.org/abs/2106.01342.

[19] David McElfresh and Ameet Talwalkar. *TabZilla Benchmark*. https://github.com/automl/tabzilla. Version 1.0, accessed May 2025.

[20] Mohammad Javad Shafiee et al. "NetScore: Towards Universal Metrics for Large-Scale Performance Evaluation of Deep Neural Networks". In: *NeurIPS Workshop*. 2018.

[21] Tomaso Trinci et al. "How green is continual learning, really? Analyzing the energy consumption in continual training of vision foundation models". In: *arXiv preprint arXiv:2409.18664* (2024). Accepted to GreenFOMO Workshop at ECCV 2024.

[22] Gido M. van de Ven, Tinne Tuytelaars, and Andreas S. Tolias. "Three types of incremental learning". In: *Nature Machine Intelligence* 4.12 (2022), pp. 1185–1197. DOI: 10.1038/s42256-022-00568-3.

[23] Gido M. van de Ven and Andreas S. Tolias. *Three scenarios for continual learning*. 2019. arXiv: 1904.07734 [cs.LG]. URL: https://arxiv.org/abs/1904.07734.

[24] Ian J. Goodfellow et al. *An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks*. 2015. arXiv: 1312.6211 [stat.ML]. URL: https://arxiv.org/abs/1312.6211.

[25] Friedemann Zenke, Ben Poole, and Surya Ganguli. *Continual Learning Through Synaptic Intelligence*. 2017. arXiv: 1703.04200 [cs.LG]. URL: https://arxiv.org/abs/1703.04200.

[26] Haizhou Shi and Hao Wang. *A Unified Approach to Domain Incremental Learning with Memory: Theory and Algorithm*. 2023. arXiv: 2310.12244 [cs.LG]. URL: https://arxiv.org/abs/2310.12244.

[27] Vincenzo Lomonaco and Davide Maltoni. *CORe50: a New Dataset and Benchmark for Continuous Object Recognition*. 2017. arXiv: 1705.03550 [cs.CV]. URL: https://arxiv.org/abs/1705.03550.

[28] Xingchao Peng et al. "Moment matching for multi-source domain adaptation". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1406–1415.

[29] ANTHONY ROBINS and. "Catastrophic Forgetting, Rehearsal and Pseudorehearsal". In: *Connection Science* 7.2 (1995), pp. 123–146. DOI: 10.1080/09540099550039318. eprint: https://doi.org/10.1080/09540099550039318. URL: https://doi.org/10.1080/09540099550039318.

[30] Sylvestre-Alvise Rebuffi et al. *iCaRL: Incremental Classifier and Representation Learning*. 2017. arXiv: 1611.07725 [cs.CV]. URL: https://arxiv.org/abs/1611.07725.

[31] Arthur Douillard et al. *PODNet: Pooled Outputs Distillation for Small-Tasks Incremental Learning*. 2020. arXiv: 2004.13513 [cs.CV]. URL: https://arxiv.org/abs/2004.13513.

[32] Matthias Delange et al. "A continual learning survey: Defying forgetting in classification tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. ISSN: 1939-3539. DOI: 10.1109/tpami.2021.3057446. URL: http://dx.doi.org/10.1109/TPAMI.2021.3057446.

[33] Robert M. French. "Catastrophic forgetting in connectionist networks". In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135. ISSN: 1364-6613. DOI: https://doi.org/10.1016/S1364-6613(99)01294-2. URL: https://www.sciencedirect.com/science/article/pii/S1364661399012942.

[34] James Mcclelland, Bruce Mcnaughton, and Randall O'Reilly. "Why There are Complementary Learning Systems in the Hippocampus and Neocortex: Insights from the Successes and Failures of Connectionist Models of Learning and Memory". In: *Psychological review* 102 (Aug. 1995), pp. 419–57. DOI: 10.1037/0033-295X.102.3.419.

[35] Yury Gorishniy et al. *Revisiting Deep Learning Models for Tabular Data*. 2023. arXiv: 2106.11959 [cs.LG]. URL: https://arxiv.org/abs/2106.11959.

[36] Jacob Armstrong and David A. Clifton. "Continual learning of longitudinal health records". In: *2022 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*. IEEE, Sept. 2022, pp. 01–06. DOI: 10.1109/bhi56158.2022.9926878. URL: http://dx.doi.org/10.1109/BHI56158.2022.9926878.

[37] Ravid Shwartz-Ziv and Amitai Armon. *Tabular Data: Deep Learning is Not All You Need*. 2021. arXiv: 2106.03253 [cs.LG]. URL: https://arxiv.org/abs/2106.03253.

[38] Joaquin Vanschoren et al. “OpenML: networked science in machine learning”. In: *ACM SIGKDD Explorations Newsletter* 15.2 (June 2014), pp. 49–60. ISSN: 1931-0153. DOI: 10.1145/2641190.2641198. URL: http://dx.doi.org/10.1145/2641190.2641198.

[39] Bernd Bischl et al. *OpenML Benchmarking Suites*. 2021. arXiv: 1708.03731 [stat.ML]. URL: https://arxiv.org/abs/1708.03731.

[40] Randal S. Olson et al. “PMLB: a large benchmark suite for machine learning evaluation and comparison”. In: *BioData Mining* 10.1 (Dec. 2017), p. 36. ISSN: 1756-0381. DOI: 10.1186/s13040-017-0154-4. URL: https://doi.org/10.1186/s13040-017-0154-4.

[41] Pieter Gijsbers et al. *AMLB: an AutoML Benchmark*. 2023. arXiv: 2207.12560 [cs.LG]. URL: https://arxiv.org/abs/2207.12560.

[42] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml. 2019.

[43] Liudmila Prokhorenkova et al. *CatBoost: unbiased boosting with categorical features*. 2019. arXiv: 1706.09516 [cs.LG]. URL: https://arxiv.org/abs/1706.09516.

[44] Arlind Kadra et al. “Well-tuned Simple Nets Excel on Tabular Datasets”. In: *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021.

[45] Noah Hollmann et al. “Accurate predictions on small data with a tabular foundation model”. In: *Nature* (Jan. 2025). DOI: 10.1038/s41586-024-08328-6. URL: https://www.nature.com/articles/s41586-024-08328-6.

[46] Sergei Popov, Stanislav Morozov, and Artem Babenko. “Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data”. In: *CoRR* abs/1909.06312 (2019). arXiv: 1909.06312. URL: http://arxiv.org/abs/1909.06312.

[47] Weiping Song et al. “AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM ’19. ACM, Nov. 2019, pp. 1161–1170. DOI: 10.1145/3357384.3357925. URL: http://dx.doi.org/10.1145/3357384.3357925.

[48] Xin Huang et al. “TabTransformer: Tabular Data Modeling Using Contextual Embeddings”. In: *CoRR* abs/2012.06678 (2020). arXiv: 2012.06678. URL: https://arxiv.org/abs/2012.06678.

[49] Zifeng Wang et al. *Learning to Prompt for Continual Learning*. 2022. arXiv: 2112.08654 [cs.LG]. URL: https://arxiv.org/abs/2112.08654.

[50] Zifeng Wang et al. *DualPrompt: Complementary Prompting for Rehearsal-free Continual Learning*. 2022. arXiv: 2204.04799 [cs.LG]. URL: https://arxiv.org/abs/2204.04799.

[51] James Seale Smith et al. *CODA-Prompt: COntinual Decomposed Attention-based Prompting for Rehearsal-Free Continual Learning.* 2023. arXiv: 2211.13218 [cs.CV]. URL: https://arxiv.org/abs/2211.13218.

[52] Quyen Tran et al. *KOPPA: Improving Prompt-based Continual Learning with Key-Query Orthogonal Projection and Prototype-based One-Versus-All.* 2024. arXiv: 2311.15414 [cs.LG]. URL: https://arxiv.org/abs/2311.15414.

[53] Arlind Kadra, Sebastian Pineda Arango, and Josif Grabocka. *Interpretable Mesomorphic Networks for Tabular Data.* 2024. arXiv: 2305.13072 [cs.LG]. URL: https://arxiv.org/abs/2305.13072.

[54] Guri Zabërgja et al. *Is Deep Learning finally better than Decision Trees on Tabular Data?* 2025. arXiv: 2402.03970 [cs.LG]. URL: https://arxiv.org/abs/2402.03970.

[55] Rahaf Aljundi et al. “Memory Aware Synapses: Learning what (not) to forget”. In: *CoRR* abs/1711.09601 (2017). arXiv: 1711.09601. URL: http://arxiv.org/abs/1711.09601.

[56] Zhizhong Li and Derek Hoiem. “Learning without Forgetting”. In: *CoRR* abs/1606.09282 (2016). arXiv: 1606.09282. URL: http://arxiv.org/abs/1606.09282.

[57] Andrei A. Rusu et al. “Progressive Neural Networks”. In: *CoRR* abs/1606.04671 (2016). arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671.

[58] David Rolnick et al. “Experience Replay for Continual Learning”. In: *CoRR* abs/1811.11682 (2018). arXiv: 1811.11682. URL: http://arxiv.org/abs/1811.11682.

[59] Hanul Shin et al. *Continual Learning with Deep Generative Replay.* 2017. arXiv: 1705.08690 [cs.AI]. URL: https://arxiv.org/abs/1705.08690.

[60] Nitin Kamra, Umang Gupta, and Yan Liu. *Deep Generative Dual Memory Network for Continual Learning.* 2018. arXiv: 1710.10368 [cs.LG]. URL: https://arxiv.org/abs/1710.10368.

[61] Ronald Kemker and Christopher Kanan. *FearNet: Brain-Inspired Model for Incremental Learning.* 2018. arXiv: 1711.10563 [cs.LG]. URL: https://arxiv.org/abs/1711.10563.

[62] Lorenzo Pellegrini et al. *Latent Replay for Real-Time Continual Learning.* 2020. arXiv: 1912.01100 [cs.LG]. URL: https://arxiv.org/abs/1912.01100.

[63] Jake Snell, Kevin Swersky, and Richard S. Zemel. *Prototypical Networks for Few-shot Learning.* 2017. arXiv: 1703.05175 [cs.LG]. URL: https://arxiv.org/abs/1703.05175.

[64] Martín González Soto et al. “XuILVQ: A River Implementation of the Incremental Learning Vector Quantization for IoT”. In: *Proceedings of the 19th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks.* PE-WASUN ’22. Montreal, Quebec, Canada: Association for Computing Machinery, 2022, pp. 1–8. ISBN: 9781450394833. DOI: 10.1145/3551663.3558676. URL: https://doi.org/10.1145/3551663.3558676.

[65] Pablo García-Santaclara, Bruno Fernández-Castro, and Rebeca P. Díaz-Redondo. *Overcoming Catastrophic Forgetting in Tabular Data Classification: A Pseudorehearsal-based approach*. 2024. arXiv: 2407.09039 [cs.LG]. URL: https://arxiv.org/abs/2407.09039.

[66] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[67] Saurav Jha et al. *NPCL: Neural Processes for Uncertainty-Aware Continual Learning*. 2023. arXiv: 2310.19272 [cs.LG]. URL: https://arxiv.org/abs/2310.19272.

[68] Lucía Bouza, Aurélie Bugeau, and Loïc Lannelongue. "How to estimate carbon footprint when training deep learning models? A guide and review". In: *Environmental Research Communications* 5.11 (Nov. 2023), p. 115014. ISSN: 2515-7620. DOI: 10.1088/2515-7620/acf81b. URL: http://dx.doi.org/10.1088/2515-7620/acf81b.

[69] Peter Henderson et al. "Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning". In: *Proceedings of the Workshop on Challenges in Deploying and monitoring Machine Learning Systems (EMNLP)*. arXiv:2002.05651. 2020.

[70] Rafał Różycki, Dorota Agnieszka Solarska, and Grzegorz Waligóra. "Energy-Aware Machine Learning Models—A Review of Recent Techniques and Perspectives". In: *Energies* 18.11 (2025). ISSN: 1996-1073. DOI: 10.3390/en18112810. URL: https://www.mdpi.com/1996-1073/18/11/2810.

[71] Arya Tschand et al. *MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from Microwatts to Megawatts for Sustainable AI*. 2025. arXiv: 2410.12032 [cs.AR]. URL: https://arxiv.org/abs/2410.12032.

[72] Imran Latif et al. "Single-Node Power Demand During AI Training: Measurements on an 8-GPU NVIDIA H100 System". In: *IEEE Access* 13 (2025), pp. 61740–61747. ISSN: 2169-3536. DOI: 10.1109/access.2025.3554728. URL: http://dx.doi.org/10.1109/ACCESS.2025.3554728.

[73] Loïc Lannelongue, Jason Grealey, and Michael Inouye. "Green algorithms: quantifying the carbon footprint of computation". In: *Advanced Science* 8.12 (2021), p. 2100707. DOI: 10.1002/advs.202100707. URL: https://onlinelibrary.wiley.com/doi/10.1002/advs.202100707.

[74] Victor Schmidt et al. *CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing*. https://github.com/mlco2/codecarbon. 2021. DOI: 10.5281/zenodo.4658424.

[75] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. *Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models*. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems. arXiv:2007.03051. July 2020. arXiv: 2007.03051 [cs.LG]. URL: https://arxiv.org/abs/2007.03051.

[76] S. A. Budennyy et al. “eco2AI: Carbon Emissions Tracking of Machine Learning Models as the First Step Towards Sustainable AI”. en. In: *Doklady Mathematics* (Jan. 2023). DOI: 10.1134/S1064562422060230. URL: https://doi.org/10.1134/S1064562422060230.

[77] Alexandre Lacoste et al. *Quantifying the Carbon Emissions of Machine Learning.* NeurIPS Workshop on Tackling Climate Change with Machine Learning. 2019. arXiv: 1910.09700 [cs.LG]. URL: https://arxiv.org/abs/1910.09700.

[78] Tristan Trébaol et al. “CUMULATOR — a tool to quantify and report the carbon footprint of machine learning computations and communication in academia and healthcare”. In: *Infoscience EPFL: record 278189* (2020). URL: https://infoscience.epfl.ch/record/278189.

[79] Zeyu Yang, Karel Adámek, and Wesley Armour. *Part-time Power Measurements: nvidia-smi’s Lack of Attention.* 2024. arXiv: 2312.02741v2 [cs.DC]. URL: https://arxiv.org/abs/2312.02741v2.

[80] Kay Henning Brodersen et al. “The Balanced Accuracy and Its Posterior Distribution”. In: *2010 20th International Conference on Pattern Recognition.* 2010, pp. 3121–3124. DOI: 10.1109/ICPR.2010.764.

[81] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[82] Takuya Akiba et al. “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2019.

[83] Xavier Gastaldi. *Shake-Shake regularization.* 2017. arXiv: 1705.07485 [cs.LG]. URL: https://arxiv.org/abs/1705.07485.

[84] Yoshihiro Yamada et al. “Shakedrop Regularization for Deep Residual Learning”. In: *IEEE Access* 7 (2019), pp. 186126–186136. ISSN: 2169-3536. DOI: 10.1109/access.2019.2960566. URL: http://dx.doi.org/10.1109/ACCESS.2019.2960566.

[85] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization.* 2018. arXiv: 1710.09412 [cs.LG]. URL: https://arxiv.org/abs/1710.09412.

[86] C. J. Van Rijsbergen. “Information Retrieval”. In: (1979).

[87] I. J. Good. “Rational Decisions”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 14.1 (1952), pp. 107–114.

[88] Tom Fawcett. “An Introduction to ROC Analysis”. In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874.

[89] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[90] ElmorLabs. *PMD-USB (Power Measurement Device with USB).* https://www.elmorlabs.com/product/elmorlabs-pmd-usb-power-measurement-device-with-usb/. Accessed: 2025-01-15. 2023.

[91] Milton Friedman. “The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance”. In: *Journal of the American Statistical Association* 32.200 (1937), pp. 675–701. ISSN: 01621459, 1537274X. URL: http://www.jstor.org/stable/2279372 (visited on 08/12/2025).

[92] Frank Wilcoxon. “Individual Comparisons by Ranking Methods”. In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83. ISSN: 00994987. URL: http://www.jstor.org/stable/3001968 (visited on 08/12/2025).

[93] Sture Holm. “A Simple Sequentially Rejective Multiple Test Procedure”. In: *Scandinavian Journal of Statistics* 6.2 (1979), pp. 65–70. ISSN: 03036898, 14679469. URL: http://www.jstor.org/stable/4615733 (visited on 08/12/2025).

[94] Peter Björn Nemenyi. “Distribution-free Multiple Comparisons”. PhD thesis. Princeton University, 1963.

[95] Janez Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 1–30. ISSN: 1532-4435.

[96] Reza Shokri et al. *Membership Inference Attacks against Machine Learning Models*. 2017. arXiv: 1610.05820 [cs.CR]. URL: https://arxiv.org/abs/1610.05820.

[97] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV]. URL: https://arxiv.org/abs/1510.00149.

[98] Roy Schwartz et al. *Green AI*. 2019. arXiv: 1907.10597 [cs.CY]. URL: https://arxiv.org/abs/1907.10597.

[99] William Stanley Jevons. *The Coal Question: An Inquiry Concerning the Progress of the Nation, and the Probable Exhaustion of Our Coal-Mines*. Original work published 1865. London: Macmillan and Co., 1865.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the invaluable guidance and support of my supervisors Dr. Yuandou Wang and Dr. Rihan Hai, whose expertise and patience shaped every aspect of this work. I am equally grateful to my thesis committee members Dr. Sole Pera and Mr. Steven van Haren for their insightful feedback and rigorous evaluation. Special thanks to Dr. Julius Philipp Roeder for his technical support and collaboration throughout this research. Congratulations are also in order for Dr. Hai on the recent arrival of her baby, it being a joyful milestone during this academic journey.

I would also like to acknowledge Delft University of Technology and the Web Information Systems track for providing the research environment and resources that made this work possible.

A

Algorithms

Algorithm 2 Optimal Segment Size Selection

```
Require: Dataset with $N$ training instances, bounds $k_{\min} = 500$, $k_{\max} = 1000$
Ensure: Segment size $k^*$ that minimizes remainder
 1: best_remainder $\leftarrow N$
 2: $k^* \leftarrow k_{\min}$
 3: for $k = k_{\min}$ to $\min(k_{\max}, N)$ do
 4:     num_segments $\leftarrow \lfloor N/k \rfloor$
 5:     remainder $\leftarrow N \bmod k$
 6:     if remainder $= 0$ then
 7:         return $k$                                   ▷ Perfect division found
 8:     if remainder $<$ best_remainder then
 9:         best_remainder $\leftarrow$ remainder
10:         $k^* \leftarrow k$
11: return $k^*$
```

B

Further Analysis (Additional Figures and Tables)

B.1. Empirical Validation of Scaling Properties

One of IMLP's key theoretical advantages over traditional batch retraining approaches is its constant computational complexity per segment, leading to linear energy scaling over time. To empirically validate this claim, we conducted a detailed scaling analysis comparing cumulative energy consumption patterns between IMLP and the standard MLP baseline across extended streaming scenarios.

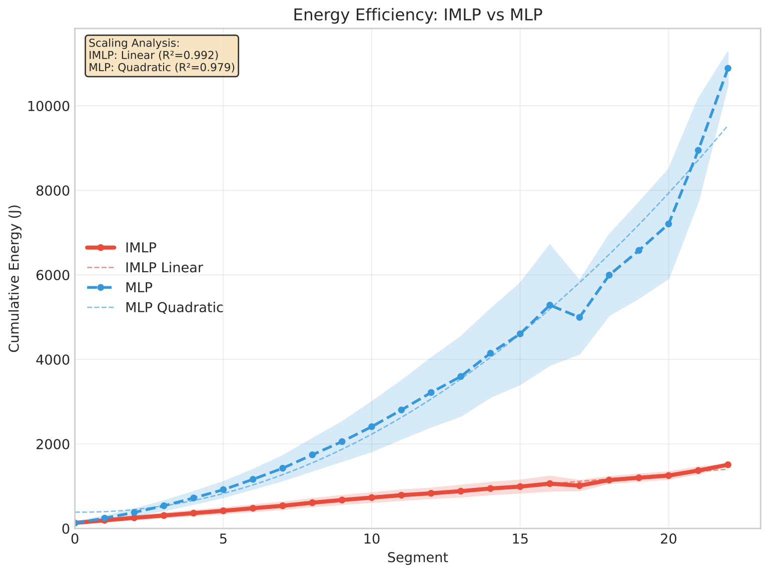


Figure B.1: Empirical validation of scaling properties showing cumulative energy consumption over 20 segments. IMLP exhibits linear scaling ($R^2 = 0.992$) while MLP demonstrates quadratic scaling ($R^2 = 0.979$). Shaded regions represent standard deviations across multiple dataset-seed combinations, demonstrating consistent scaling patterns across experimental conditions.

We extended our evaluation beyond the standard TabZilla benchmark segments to analyze longer streaming sequences of up to 20 segments. For each model-dataset-seed combination, we calculated cumulative energy consumption and applied systematic polynomial fitting to characterize scaling behavior. The analysis employed an automated model selection procedure using scikit-learn's polynomial regression with degrees 1 (linear), 2 (quadratic), and 3 (cubic), computing R^2 scores to quantify goodness of fit for each polynomial degree.

The selection process incorporated a meaningful improvement threshold of 0.01 in R^2 values to avoid overfitting to noise while capturing genuine scaling patterns. If the improvement from linear to cubic fitting was less than 0.01, we selected the linear model. When quadratic fitting showed meaningful improvement over linear (> 0.01) but cubic fitting provided minimal additional benefit (< 0.01), we selected the quadratic model. Otherwise, we selected the cubic model to capture more complex scaling behaviors.

Figure B.1 presents the cumulative energy consumption patterns with confidence intervals and fitted curves. The automated analysis confirms our theoretical predictions with remarkable precision. The model selection procedure identified linear scaling as optimal for IMLP with $R^2 = 0.992$, indicating that 99.2% of the variance in cumulative energy consumption is explained by a simple linear relationship. This validates our theoretical analysis that attention-based feature rehearsal incurs fixed computational overhead regardless of historical data volume.

For the MLP baseline, the analysis identified quadratic scaling with $R^2 = 0.979$, reflecting the growing computational burden of retraining on accumulated data. The strong quadratic fit demonstrates that energy consumption grows proportionally to the square of the number of processed segments, as expected from the cumulative data retraining paradigm where each new segment requires processing all previously accumulated data.

The scaling analysis incorporates several methodological strengths that ensure statistical robustness. The shaded regions in the plot represent standard deviations across multiple dataset-seed combinations, demonstrating that scaling patterns remain consistent across different experimental conditions rather than representing artifacts of specific datasets or random initializations. The threshold-based polynomial selection prevents overfitting while ensuring that genuinely different scaling behaviors are detected, with the clear separation between linear and quadratic patterns validating that these represent fundamentally different algorithmic approaches rather than statistical noise.

Both models achieve R^2 values exceeding 0.97, providing strong statistical evidence that the observed scaling patterns are systematic and predictable. This high level of statistical confidence supports their use for long-term resource planning and deployment decisions in practical continual learning scenarios.

The observed scaling patterns directly reflect the fundamental computational differences between approaches. IMLP maintains constant complexity per segment, requiring $\mathcal{O}(N + W \cdot d_h^2)$ operations, where N represents the current segment size, W the fixed attention window, and d_h the feature dimension. Since W and d_h are architectural constants, computational complexity remains independent of stream length. Conversely, the MLP baseline exhibits cumulative complexity, requiring $\mathcal{O}(\sum_{i=1}^{t} N_i)$ operations to retrain on all accumulated data from segments 1 through t, leading to the empirically observed quadratic growth.

C
Source Code

Listing 6 Reference implementation of `calculate_optimal_segment_size` (`openml_data_processor.py`).

```python
def calculate_optimal_segment_size(total_instances: int,
                                   min_size: int = 500,
                                   max_size: int = 1000) -> int:
    """
    Determine a segment size k (min_size  k  max_size) that partitions
    the dataset with minimal remainder.  For datasets smaller than
    min_size, return total_instances to create a single segment.
    """
    # Fallback for very small datasets
    if total_instances < min_size:
        print(f"Dataset is smaller than minimum segment size ({min_size}). "
              f"Using one segment with {total_instances} instances.")
        return total_instances

    best_remainder = total_instances
    optimal_size   = min_size

    # Exhaustive scan of admissible segment sizes
    for size in range(min_size, min(max_size + 1, total_instances + 1)):
        num_segments = total_instances // size
        if num_segments == 0:
            continue  # safety check; should not occur after small-data guard

        remainder = total_instances % size

        # Perfect division  early exit
        if remainder == 0:
            return size

        # Track smallest remainder encountered so far
        if remainder < best_remainder:
            best_remainder = remainder
            optimal_size   = size

    return optimal_size
```

D

Research Paper

Can Context-Aware Incremental Nets Outperform GBDTs Over Time? A Tabular Lifelong-Learning Study

Anonymous authors (double-blind review process)

Abstract

Traditional benchmarks position gradient-boosted decision trees (GBDTs) as the standard for tabular prediction. These studies assume a static training dataset and ignore the growing cost of continuously retraining models as data drifts. We revisit tabular learning from the perspectives of *domain-incremental learning* (Domain-IL) and *Green AI*. We present **IMLP**, a context-aware Incremental Multi-Layer Perceptron that attaches an attentional "look-back" module to each new segment, re-using hidden representations from earlier segments instead of replay buffers. IMLP is embedded in a unified PyTorch pipeline that measures wall-power in real time, letting us evaluate our model on a broad subset of the *TabZilla* OpenML benchmarks.

We show that while IMLP's balanced-accuracy approaches GBDTs and conventional MLPs, it cuts *cumulative training energy and inference power by large margins*, thanks to constant-time updates and a single shared backbone. To capture this trade-off we introduce **NetScore-T**, a task-agnostic ranking that jointly rewards performance and joule efficiency. We also release all logs, code and power traces to promote reproducible energy-aware research. Our findings suggest that carefully designed neural architectures can offer compelling accuracy-efficiency balances on streaming tabular data, and point toward future work on closing the remaining accuracy gap while further shrinking the carbon footprint of lifelong learners.

1 Introduction

Continual learning (CL) is the ability of a model to update from an ongoing stream of tasks or data distributions while retaining prior knowledge [8, 35]. The primary challenge is *catastrophic forgetting*, in which new updates overwrite parameters needed for earlier tasks and sharply degrade past performance [8, 35, 14].

Traditional CL studies use computer-vision benchmarks such as CIFAR-100 or ImageNet and assume explicit task identities [30, 39, 5]. In contrast, **tabular data** (e.g., structured feature tables common in finance, healthcare) has been relatively underexplored [15, 14, 3].

We frame our work on top of the continual learning by De Lange et al. [8], and the recent domain-incremental advances such as UDIL, [43] and one-shot DIL, [12]. We adopt the *domain-incremental learning* (Domain-IL) setting, where a model must handle successive data segments drawn from shifting input distributions while the label space stays fixed, yet receives no domain or task identifiers during training or inference. This contrasts with *task-incremental learning*, (Task-IL), which presupposes known task boundaries and often allocates separate classifier heads. Domain-IL instead demands continual adjustment to unseen shifts and retention of earlier knowledge, mirroring real-world tabular workflows such as month-by-month updates to financial or patient records that lack explicit change annotations[31, 3].

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

A broad toolkit of regularisation [28], episodic-replay [30], and exemplar-free generative replay [44] exists to mitigate catastrophic forgetting, in which surveys catalogue more variants [49]. However, relatively few have addressed the issue from the perspective of energy efficiency. Evidence shows that repeated fine-tuning can be environmentally costly [19, 18, 46], and recent efforts for edge or cluster deployment highlight the need for energy-aware CL frameworks [29, 24]. Therefore, it is critical that incremental learning approaches are not only capable of reducing forgetting but do so in a resource-efficient manner. Despite this growing need, the problem of energy-efficient continual learning has not yet been studied in detail for tabular data.

This work aims to bridge this gap and investigate the trade-offs between balanced accuracy, runtime, and energy in CL. We propose a novel incremental MLP framework, **IMLP**, explicitly designed for tabular data. IMLP incorporates attention-based feature replay into a lightweight MLP architecture to address the dual challenge of mitigating forgetting and improving energy efficiency. To quantitatively assess the trade-offs between predictive performance and energy consumption, we introduce NetScore-T, a joint metric that captures both accuracy and total energy consumption during training and inference. To obtain ground-truth measurements, we instrument our continual learning pipeline with an ElmorLabs PMD-USB power meter and corresponding PCI-E slot adapter [10, 11], capturing real-time wall-power draw for CPU and GPU throughout online updates. This setup allows us to precisely attribute Joules consumed to each incremental training step and inference batch, and to compute energy-accuracy trade-offs under the NetScore–T framework. We conduct extensive experiments and benchmark IMLP against a standard MLP baseline retrained on cumulative data, demonstrating the promise of energy-aware continual learners. Through experimental analysis, we highlight the advantages of IMLP for energy-efficient continual learning by exploring the following three sub-questions: (I) Can IMLP improve energy efficiency in continual learning? (II) What are the trade-offs between model performance and energy efficiency in a streaming tabular data setting? (III) How robust are IMLP's results across different tabular datasets?

Our core contributions are listed as follows:

- We propose an *attention-based rehearsal mechanism* for MLPs in continual tabular classification.
- We design a complete measurement pipeline for **energy-aware continual learning**, combining ElmorLabs PMD-USB readings with per-task benchmarking.
- We demonstrate that `IMLP` retains similar accuracy while using *significantly less energy* compared to retraining-based approaches.

2 Related Work

Early work on catastrophic forgetting, notably by French [13], introduced the stability-plasticity dilemma: a model must remain plastic enough to learn new tasks while stable enough to retain prior knowledge [13, 28]. Proposed solutions include regularizing weight updates to preserve past knowledge [13, 28], expanding model capacity for new tasks, and rehearsal, the replay of past data. Among these, rehearsal-based methods have gained prominence for their effectiveness in class-incremental learning benchmarks [32]. We review key rehearsal and memory-based approaches relevant to tabular continual learning in the following. Then, we turn to the attention-based feature integration as implemented in our proposed `IMLP` model.

2.1 Rehearsal Methods: Buffer vs. Generative Replay

A widely used approach to mitigate forgetting is buffer-based replay, where a small memory buffer stores examples from previous tasks to be interleaved with new data during training [30, 39, 5]. Notable methods include iCaRL's exemplar memory [39] and GEM/A-GEM, which constrain updates using past samples [5]. While effective, these methods raise concerns around storage and data privacy, particularly on edge devices or domains like healthcare and federated learning, where retaining raw data is often infeasible or prohibited [5]. Furthermore, a recent study [48] examined its strengths and limitations, showing that while rehearsal helps maintain low loss on past tasks, it can lead to overfitting or misrepresent the original data distribution. Despite its limitations, buffer replay remains a strong baseline. Even a few stored instances can aid performance under distributional drift in tabular data streams. However, naive replay may falter when feature semantics shift across tasks, a frequent challenge in non-stationary settings.

To circumvent the need for storing real data, generative models can be trained to approximate past data distributions [40, 44] and proposed initially as pseudo-rehearsal deep generative replay [44], where a generator (e.g., a GAN or VAE) produces pseudo-examples to be replayed alongside new task data. While this mitigates storage and privacy concerns, it introduces challenges: generative models are often trained offline, requiring multiple passes over data, making them difficult to update continually. Moreover, the effectiveness of rehearsal depends on the fidelity of generated samples. If the generator fails to capture key feature correlations, forgetting may still occur [44]. Notably, most state-of-the-art generative replay methods have been developed for image data, leaving their applicability to structured (tabular) data largely unexplored [32]. Addressing this gap, [14] proposed TRIL3, a rehearsal-based lifelong learning framework for tabular data that replaces deep generative models with XuILVQ, an incremental prototype-based generator. XuILVQ supports online updates and generates synthetic samples for previously seen classes, offering a more practical solution for continual learning in tabular domains [14].

Buffer and generative replay pursue the same objective, preserving past knowledge by providing representative examples during training, but differ fundamentally in approach. While buffer replay approximates the joint data distribution by storing real samples, generative replay seeks to model this distribution explicitly through learned generation [5]. In tabular CL, buffer replay remains a strong baseline when a small number of real instances per class can be retained. However, pseudo-rehearsal via generative replay becomes a more viable alternative in scenarios with strict storage or privacy constraints.

2.2 Feature-Level Replay and Memory-Based Learning

A promising direction that bridges buffer and generative replay is feature-level replay, which stores latent representations instead of raw input data. Pellegrini et al. [37] proposed Latent Replay for on-device continual learning, where stored latent features are replayed as inputs to later network layers. These task-relevant features (typically activation vectors from intermediate layers) are lower-dimensional and more memory-efficient. This approach eliminates the need to train a full generative model and avoids challenges associated with explicit generative losses while preserving valuable information for mitigating forgetting. A related approach is Latent Generative Replay (LGR), which combines the generative replay paradigm with the efficiency of feature-level representations[45, 27]. Rather than generating complete input data (e.g., images), LGR trains lightweight generators to produce latent features representative of previous tasks. For example, Kim et al. [27] introduced Pseudo-Replay via Latent Sampling, which leverages a pre-trained feature extractor to sample from stored distributions of latent features, achieving strong performance in class-incremental learning scenarios.

Both feature-level replay and latent replay leverage the efficiency of latent representations, but differ in their approach: feature replay enhances the training set with latent vectors, whereas attention-based replay (discussed next) enriches the model's input representation by directly incorporating retrieved feature-level memories.

Memory-based methods also include techniques like prototype learning and kNN-based classification in CL. For instance, iCaRL stores exemplars to compute class means in feature space for non-parametric classification [39]. Similarly, FearNet [26] introduces a dual-memory system inspired by the brain, with a recent memory for current tasks and a long-term memory (implemented with a generative autoencoder) for past tasks. These methods treat past data as additional capacity that can be recalled when needed, an idea that underpins many memory-augmented continual learning systems.

2.3 Attention-Based Replay and Retrieval Mechanisms

Attention mechanisms have transformed sequence learning, notably through the Transformer's self-attention mechanism [47], allowing networks to focus on relevant parts of their input or memory selectively. In the context of continual learning, attention can be leveraged to dynamically retrieve relevant past information, rather than relying solely on passive replay.

Recent studies in vision and language have applied key-query attention mechanisms to continual learning. For instance, methods like L2P [50] maintain a pool of learnable prompt vectors, each associated with a key for different tasks. When a new sample arrives, its query (derived from an embedding) selects the top-k prompt keys, and the corresponding prompts are concatenated with the input tokens of a Vision Transformer. This approach represents an attention-based retrieval of task-specific context. Such methods have demonstrated superior performance compared to many rehearsal-based techniques, all while avoiding the need for an explicit replay buffer.

In large language models, He et al. [17] demonstrated that not all attention heads are equally crucial for retaining knowledge. They introduced an attention distillation method, SEEKR, which identifies the attention heads most susceptible to forgetting and transfers their outputs from the old model to the new one. While SEEKR employs attention for regularization rather than replay, it reinforces the idea that attention weights play a critical role in preserving past knowledge.

Jha et al. [22] introduced a continual learning method based on Attentive Neural Processes (ANPs), where examples from past tasks are treated as a context set, and a new input serves as the query. The ANP then generates predictions through interpolation over context points, weighted by an attention kernel. This mechanism enables an instance-based recall, effectively functioning as feature-level replay on the fly. The Neural Process Continual Learning (NPCL) framework builds on this approach, integrating uncertainty estimation to enhance transfer learning and mitigate forgetting.

A common thread across these methods is using a key-value memory store, which can be queried using a similarity function. Both Memory Networks and Transformers [47] employ this mechanism to model short- and long-term dependencies. In continual learning, this concept translates to soft replay: instead of replaying exact past samples, the model retrieves a relevance-weighted summary of past experiences, guided by attention, to inform current predictions.

Attention-based replay offers several key advantages. It is inherently selective-memory is accessed only when relevant, as attention weights diminish for inputs dissimilar to stored features, thereby reducing interference. Unlike generative replay, it requires no additional network to approximate past data distributions, relying instead on exact stored feature representations. The computational cost is minimal, involving only a few matrix multiplications during inference and training. Furthermore, dot-product attention functions as a kernel smoother over stored features, providing a principled form of local density estimation that blends past and new information.

2.4 Energy-Efficient Continual Learning

In recent years, the environmental and economic costs of deep learning have spurred a surge of interest in quantifying and reducing energy consumption during both training and inference. Henderson et al. [19] were among the first to demonstrate that ostensibly similar models can exhibit order-of-magnitude differences in energy use, advocating for rigor in energy reporting. Building on this, Trinci et al. [46] introduced NetScore, a composite metric that balances accuracy, and total energy, highlighting the importance of multi-dimensional evaluation in continual settings. Meanwhile, Bouza et al. [18] surveyed the landscape of energy-aware AI, cataloguing a variety of software profilers and logging frameworks (e.g., NVIDIA NVML, Intel RAPL interfaces), as well as model-level strategies for reducing computation.

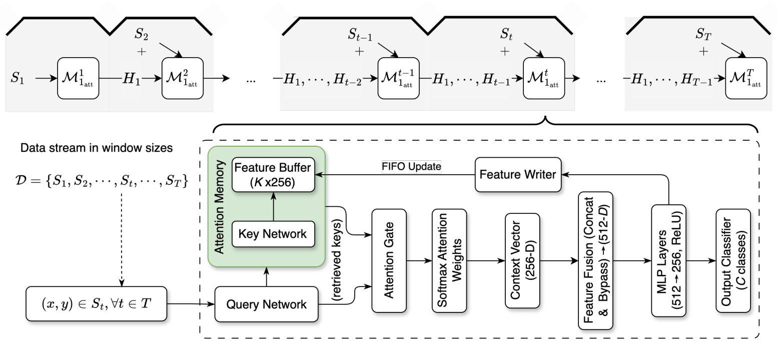
Energy monitoring solutions fall into two broad categories: on-board sensor readings and external power meters. Modern CPUs and GPUs expose energy counters via Intel's RAPL (accessible through Linux's **powercap** interface) and NVIDIA's NVML API, which software libraries such as PyJoules, Experiment-Impact-Tracker, and CodeCarbon wrap to produce per-epoch and per-step energy logs. More recently, Eco2AI has extended these capabilities with location-aware CO_2 estimation. However, on-board readings can be subject to sampling and estimation errors: Yang et al. [52] show that the shunt-resistor based power sensor on NVIDIA GPUs only samples 25% of the runtime on A100 and H100 cards, leading to under- or over-estimation of up to 65% when compared against a calibrated external meter.

Prior CL studies seldom address domain-incremental *tabular* streams *and* ignore meter-verified energy use. No method yet pairs feature-level replay with wall-power tracking, so the accuracy-energy trade-off is still unmapped. Section 3 closes this gap.

3 IMLP Method for Energy-aware Tabular Continual Learning

To address the gap in energy-efficient continual learning research for tabular data stream, we propose Incremental MLP (`IMLP`), a lightweight yet effective extension of the standard MLP. The key idea is to maintain a memory of past feature representations and incorporate them into the prediction of new samples via a dot-product attention mechanism.

3.1 IMLP Model Architecture Overview



S_1
$\mathcal{M}^1_{1_{\text{att}}}$
H_1
S_2 +
$\mathcal{M}^2_{1_{\text{att}}}$
S_{t-1} +
$H_1, \cdots, H_{t-2}$
$\mathcal{M}^{t-1}_{1_{\text{att}}}$
S_t +
$H_1, \cdots, H_{t-1}$
$\mathcal{M}^t_{1_{\text{att}}}$
S_T +
$H_1, \cdots, H_{T-1}$
$\mathcal{M}^T_{1_{\text{att}}}$
Data stream in window sizes
$\mathcal{D} = \{S_1, S_2, \cdots, S_t, \cdots, S_T\}$
$(x, y) \in S_t, \forall t \in T$
Attention Memory
Feature Buffer (Kx256)
Key Network
Query Network
FIFO Update
Feature Writer
(retrieved keys)
Attention Gate
Softmax Attention Weights
Context Vector (256-D)
Feature Fusion (Concat & Bypass) → (512-D)
MLP Layers (512 → 256, ReLU)
Output Classifier (C classes)

Figure 1: Windowed IMLP Architecture with Attention Memory.

As illustrated in Figure 1, our `IMLP` architecture introduces a windowed attention-based rehearsal mechanism over temporally ordered data segments, extending the progressive networks paradigm [41] with transformer-style attention [47]. The model receives incremental raw input data S_t from the data stream $\mathcal{D} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_t, \cdots, \mathcal{S}_T\}, \forall t \in \{1, 2, \cdots, T\}$, where each segment $\mathcal{S}_t = \{(x_i, y_i)\}_{i=1}^n$ in the t^{th} window comprises n samples with corresponding labels y_i. Each segment is processed by a dedicated MLP module, augmented with limited historical context to facilitate representation reuse while constraining memory and computational cost.

To selectively integrate useful past features into the current segment's representation, `IMLP` employs *attention gates* in each window size W. For instances, for a given segment $\mathcal{S}_t$, the current sample (x, y) is transformed into a query vector $\mathbf{q}_t = \mathbf{W}_q \mathbf{x}_t \in \mathbb{R}^{256}$, while a key matrix $\mathbf{K}_t = \mathbf{W}_t[\mathbf{h}_{\max(1,t-W)}, \ldots, \mathbf{h}_{t-1}]^\top \in \mathbb{R}^{W \times 256}$ is constructed from representations of the past W segments, where $\mathbf{h}_j$ denotes the hidden representation of segment j. When a new input sample (x, y) is encountered, the `IMLP` computes its corresponding feature representation $q = f_{\text{enc}}(x)$ via the MLP encoder. This vector q acts as the query. The model then calculates attention weights as $\alpha_i = \text{softmax}\left(\frac{\mathbf{q} \cdot \mathbf{k}_i}{\sqrt{d}}\right)$, where d is the dimensionality of the feature space, and the weights α_i indicate the relevance of each stored feature to the current input. A context vector is then computed as $c(x) = \sum_{i=1}^{M} \alpha_i \, v_i$, where, typically, $v_i = \mathbf{k}_i$. This context vector is concatenated with the original feature q and fed into the final classification layer.

During new-task training, the memory from previous tasks remains fixed. Every new sample's prediction is now influenced by both the current features and the retrieved memory context, effectively providing a dynamic, input-dependent form of rehearsal. In effect, the classifier's output becomes a function $g(q, c(x))$ that integrates both newly learned and past knowledge.

Unlike traditional buffer-based replay, which explicitly mixes stored raw samples with new data during training [30, 39], our method does not retrain on stored samples as separate inputs. Instead, it retrieves stored features via attention and integrates them directly into the forward pass, providing continuous regularization without the risk of overfitting a limited replay buffer. The attention-based rehearsal mechanism augments a conventional MLP classifier, allowing efficient feature reuse without the need to retrain on the full data history. After completing each task, the model retains a set of feature vectors extracted from representative past samples, typically the activations from the penultimate layer.

Algorithm 1 Context-Aware Incremental MLP (IMLP)

```
Require: Stream $\mathcal{D} = \{S_1, S_2, \dots, S_T\}$ of data segments
Require: Window size $W$, learning rate $\eta$, attention flag $1_{\text{att}}$
 1: Initialize $\theta \leftarrow$ random weights
 2: Initialize BUFFER $\leftarrow \emptyset$                    ▷ Stores at most $W$ feature tensors
 3: for $t = 1$ to $T$ do
 4:     for minibatch $(x, y) \in S_t$ do                       ▷ Forward pass
 5:         if $1_{\text{att}}$ and BUFFER $\neq \emptyset$ then
 6:             $K \leftarrow$ Stack(BUFFER)                    ▷ Shape: $[B, W, d]$
 7:             $q \leftarrow \phi_q(x)$                        ▷ Query: $[B, 1, d]$
 8:             $\alpha \leftarrow \text{softmax}(K \cdot q^\top)$
 9:             $c \leftarrow \alpha \cdot K$                   ▷ Context: $[B, d]$
10:         else
11:             $c \leftarrow 0$
12:         $h \leftarrow \phi_{\text{feat}}(\text{concat}(x, c))$
13:         $p \leftarrow \text{softmax}(\phi_{\text{cls}}(h))$   ▷ Backward pass
14:         $L \leftarrow \text{CE}(p, y)$
15:         $\theta \leftarrow \theta - \eta \nabla_\theta L$
16:     if $1_{\text{att}}$ then
17:         $f \leftarrow$ Detach(mean feature of $S_t$)
18:         BUFFER $\leftarrow$ BUFFER $\cup \{f\}$; trim to $W$
19: return $\theta$
```

Algorithm 1 outlines the high-level logic of our IMLP procedure. ϕ_q, ϕ_{feat}, ϕ_{cls} are the *query*, *feature extractor*, and *classifier* networks; *d* = 256 hidden dims. Specifically, we compute segment-level keys by storing hidden activations from the penultimate layer, and during inference or training, each new input forms a query vector that attends to this fixed memory. The attention weights are used to retrieve a context vector, which is concatenated with the raw input and passed through the network. The buffer logic, averages features and trims to a fixed window size. It is handled externally to the model class. Full source code is provided in the supplementary materials.

3.2 NetScore-T

To quantify the efficacy of our proposed model and assess the trade-offs between predictive performance and energy consumption in a streaming tabular setting, we introduce a joint accuracy–energy metric, NetScore-T, for further evaluation. Building on `NetScore` for static vision models [42] and the `Energy NetScore` for continual learning [46], **NetScore-T** is a task-agnostic metric that rewards predictive power while penalizing Joule cost.

Let $P_t^{(m)}$ be the performance metric of model m on the segment $\mathcal{S}_t$ and $E_t^{(m)}$ the corresponding energy measured in Joule. Note that energy splits into training and inference phases. We consider $E_t^{(m)} = E_{\text{train},t}^{(m)} + \alpha\, E_{\text{infer},t}^{(m)}$, where α controls the importance of inference overhead. We report results for $\alpha = 1$ (balanced) and $\alpha = 10$ (deployment-heavy). These measurements are obtained after each segment $\mathcal{S}_t, \forall t \in \{1, 2, \cdots, T\}$ using wall power sampled at 500–800Hz via an ElmorLabs PMD-USB power meter [10], mounted between the workstation's power supply and mains.

For each window size we report balanced accuracy, F1 score, log-loss, and AUC-ROC. Among these, balanced accuracy [4] is selected as the primary performance metric due to its robustness to class imbalance. With balanced accuracy as the default performance term, we first define the per-segment score with energy consumption as

$$\text{NS}_t^{(m)} = \frac{P_t^{(m)}}{\log_{10}\big(E_t^{(m)} + 1\big)}, \tag{1}$$

where the $\log_{10}$ compresses the three-orders-of-magnitude spread of E and the $+1$ prevents division by zero. This metric, $\text{NS}_t^{(m)} \geq 0$, favors models that achieve high performance and low energy consumption. When $P_t^{(m)} = 0$, the score also be zero, regardless of energy consumed. For a stream of T segments we take the mean of $\mathrm{NS}_t^{(m)}$ and consequently,

$$\text{NetScore-T}^{(m)} = \frac{1}{T} \sum_{t=1}^{T} \mathrm{NS}_t^{(m)}. \tag{2}$$

Note that if the log-loss $\mathcal{L}_t^{(m)}$ is the chosen performance metric we convert it to a "higher-is-better" quantity $P_t^{(m)} = \left(\mathcal{L}_t^{(m)} + \varepsilon\right)^{-1}$, $\varepsilon = 10^{-7}$ and reuse Eq. (1)–(2). In contrast, due to the log penalty, saving one order of magnitude in energy while dropping one percentage point of balanced accuracy *increases* $\mathrm{NS}_t^{(m)}$ by roughly $+1$. This preserves the interpretation: larger scores signal *better* accuracy–energy trade-offs.

4 Experiments and Evaluation

4.1 Experimental Setup

All experiments were conducted on a local workstation equipped with an Intel Core i5-8600K CPU (6 cores@3.60 GHz) and a single NVIDIA GeForce RTX 2080 Ti GPU (11 GB VRAM), running Python 3.13 and CUDA 12.2. All models were implemented in *PyTorch*, and the full software environment was frozen with `pip freeze` to ensure reproducibility. Real-time wall power was sampled at 700 Hz by an ElmorLabs PMD-USB meter [10] placed between the PSU and mains; GPU draw was isolated with the matching PCIe-slot adapter. Hyper-parameters were tuned with `Optuna` [1], following the protocol of Kadra et al. [23].

4.2 Data Streams and Baselines

We evaluate our method on 36 classification datasets from the *TabZilla* benchmark [33, 34], originally compiled by Kadra et al. [23], McElfresh et al. [34]. Each dataset is divided into contiguous segments of 500–1,000 samples to simulate a continual learning stream. For comparison, we consider three groups of baselines: tree-based models [T] (i.e., LightGBM [25], CatBoost [38], XGBoost [7]); classical methods [B] (e.g., k-NN, Logistic Regression, SVM, Decision Tree, and Random Forest [36]); and neural network models [N] (i.e., TabNet [2], STG [21], ResNet-1D [16], DaNet [6], VIME [53], our attention-based **IMLP** described in Section 3.1, a default MLP, and a tuned MLP baseline (**MLP_C**) sharing the same hyperparameter budget as **IMLP_C**).

4.3 Results

We report four complementary views of performance: (i) *Final balanced accuracy*, (ii) *Final log-loss*, (iii) *NetScore-T* using balanced accuracy, and (iv) *NetScore-T* using log-loss. Statistics are averaged over the 36 *TabZilla* streams.

LightGBM (state-of-the-art GBDT), the retrained MLP, TabNet and STG (representatives of dense and sparse attention) and our proposal IMLP span the three dominant algorithm families. All significance tests, however, use *all* 17 models.

Table 1 reports the Friedman χ^2 statistics for the four metrics.[1] In every case $p < 10^{-38}$, so the null hypothesis of equal performance is decisively rejected and pair-wise analysis is warranted.

Table 1: Friedman omnibus statistics (N=36, k=17)

Metric	χ^2	p–value
Balanced accuracy	226.4	3.4×10^{-39}
Log-loss	384.9	5.2×10^{-72}
NetScore-T (bal. acc.)	478.1	1.4×10^{-91}
NetScore-T (log-loss)	363.5	1.6×10^{-67}

[1] $N = 36$ streams, $k = 17$ algorithms, critical difference for post-hoc CD $= 4.12$ at $\alpha = 0.05$ (Studentised range $q_{0.05} = 3.463$).

For every metric we convert per-dataset scores into ranks (1 = best). Table 2 lists the mean rank of five representative models. A Wilcoxon test is run pair-wise against IMLP; symbols denote the outcome (‘✓‘ = significantly different at Holm-corrected $p < 0.05$; n.s. = not significant; “–” = reference).

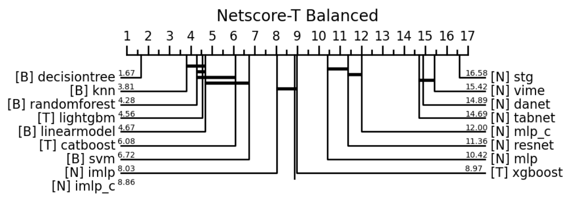
Predictive quality. Table 2 presents the average ranks and significance of the four selected models, viz LightGBM, default MLP, TabNet, and STG versus IMLP for predictive metrics, with respect to balanced accuracy and Log-loss.

Table 2: Average rank (↓ better) **and** mean ± std across streams.

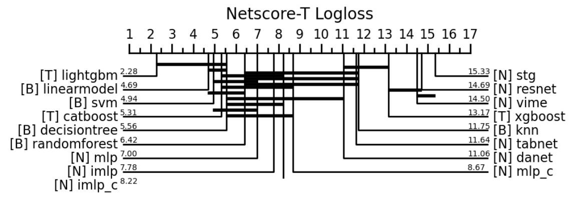
Model	Balanced accuracy		Log-loss	
	Avg. rank	Mean ± Std	Avg. rank	Mean ± Std
LightGBM [T]	4.61 ✓	$0.825{\pm}0.167$	3.56 ✓	$0.283{\pm}0.266$
MLP [N]	4.46 ✓	$0.823{\pm}0.163$	3.94 ✓	$0.336{\pm}0.330$
IMLP [N]	–	$0.804{\pm}0.170$	–	$0.399{\pm}0.348$
TabNet [N]	7.96 n.s.	$0.794{\pm}0.185$	6.11 n.s.	$0.370{\pm}0.337$
STG [N]	16.83 ✓	$0.426{\pm}0.165$	14.33 ✓	$1.153{\pm}0.691$

IMLP sits mid-pack in raw accuracy, it performs worse than the retrained MLP and LightGBM yet indistinguishable from TabNet.

Efficiency-aware performance. Figure 2 shows the comparisons of 17 models for the efficiency-aware performance. The models involved in the comparison are selected to perform relatively well in Table 3.



(a) NetScore-T (Balanced Accuracy)



(b) NetScore-T (Log Loss)

Figure 2: Critical difference diagrams for NetScore-T metrics

Table 3 presents the average ranks and significance of the exemplar models, for energy-aware metrics, i.e., NetScore-T (bal. acc) and NetScore-T (log-loss).

LightGBM’s superior balanced accuracy offsets its higher energy consumption, so it edges out IMLP in NetScore-T. Nonetheless IMLP remains the most economical *neural* route to high composite score. DecisionTree and k-NN dominate thanks to their negligible power draw; LightGBM ranks next.

Energy and latency. We also conducted statistical analysis for the energy consumption and latency at the runtime environment for the 36 data streams. As presented in Table 4, IMLP’s *constant-time* updates translate into a $\sim$**3**$\times$ median speed-up and a **>60 %** median reduction in Joule cost relative

Table 3: NetScore-T results (rank ↓ better).

	NetScore-T (bal.)		NetScore-T (log)	
Model	Rank	Score	Rank	Score
DecisionTree [B]	1.67 ✓	1.784 ± 1.106	5.56 n.s.	4.48 ± 5.43
k-NN [B]	3.81 ✓	0.891 ± 0.416	11.75 ✓	1.76 ± 2.01
IMLP [N]	–	0.404 ± 0.094	–	2.81 ± 3.54
MLP [N]	10.42 ✓	0.348 ± 0.077	7.00 n.s.	2.99 ± 3.20
LightGBM [T]	4.56 ✓	0.872 ± 0.555	2.28 ✓	9.99 ± 17.63

Table 4: Wall-time and energy per stream.

Model	Time s (median [IQR])	Mean $\pm$ SD	Energy kJ (median [IQR])	Mean $\pm$ SD
IMLP [N]	5.7 [3.2–12.9]	11.8 ± 5.9	0.53 [0.31–1.09]	1.08 ± 0.55
MLP [N]	17.4 [6.2–37.8]	35.4 ± 29.4	1.51 [0.66–3.34]	2.83 ± 2.32
LightGBM [T]	8.3 [4.1–17.5]	33.1 ± 121.7	1.34 [0.63–2.93]	2.47 ± 9.18
STG [N]	29.1 [12.7–62.3]	63.7 ± 54.4	2.43 [1.10–4.96]	5.12 ± 4.29

to the tuned MLP baseline, and unlike LightGBM, its run-time is immune to a single "mega–stream" outlier.

- **Accuracy-efficiency trade-off.** A sub-percentage-point loss in balanced accuracy buys a *two-to-three-fold* cut in both energy and latency, making IMLP the most favourable point on the accuracy-efficiency Pareto frontier among the nine neural contenders.
- **Statistical solidity.** Every claim remains significant after Wilcoxon + Holm correction; the critical-difference of 4.12 clearly separates IMLP and the retrained MLP in NetScore-T (BA), while they fall into the same clique under the log-loss variant, which is precisely what the composite metric predicts.
- **Data-privacy by design.** Because raw samples are discarded once their latent keys are stored, IMLP naturally respects strict retention policies and sidesteps the storage or compliance issues that plague buffer-based replay and generative rehearsal.

In short, **attention-based feature reuse** delivers near-state-of-the-art predictive performance *and* an order-of-magnitude gain in energy-to-accuracy ratio, a compelling direction for Green AI in lifelong tabular learning.

5 Conclusion

We revisit tabular prediction through the joint lenses of *domain-incremental learning* and *Green AI*. Our contributions are three-fold. (i) **IMLP**: an attention-augmented incremental MLP that stores only latent keys, not raw data, achieving constant-time updates, privacy compliance, and immunity to catastrophic outliers. (ii) **NetScore-T**: a task-agnostic metric that rewards accuracy while logarithmically penalising wall-power, accompanied by an open PyTorch pipeline with ElmorLabs PMD-USB logging for reproducible energy audits. (iii) The first energy-aware benchmark on 36 *TabZilla* streams. Empirically, IMLP surrenders <1,pp balanced accuracy to a fully-retrained MLP yet delivers a median $3\times$ speed-up and $> 60\,\%$ Joule savings, ranking first among nine neural baselines and rivaling LightGBM on NetScore-T; all improvements are Friedman–Wilcoxon–Holm significant [9, 20, 51]. This establishes attention-based feature reuse as a compelling Pareto point in lifelong tabular learning. Future work will narrow the residual accuracy gap via deeper backbones or adaptive memory windows, extend evaluation to federated and heterogeneous hardware where energy profiles differ [19, 18], and derive theoretical guarantees for key–value replay. We release code, logs and tuned hyper-parameters to spur progress on energy-aware continual learning.

References

[1] Akiba , T., Sano , S., Yanase , T., Ohta , T., & Koyama , M. (2019) Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*

[2] Arik , S. Ö. & Pfister , T. (2019) Tabnet: Attentive interpretable tabular learning. *CoRR* **abs/1908.07442**.

[3] Armstrong , T. & Clifton , D. (2022) Continual learning with time series data in healthcare. *IEEE Journal of Biomedical and Health Informatics*

[4] Brodersen , K. H., Ong , C. S., Stephan , K. E., & Buhmann , J. M. (2010) The balanced accuracy and its posterior distribution. In *2010 20th International Conference on Pattern Recognition* pages 3121–3124.

[5] Chaudhry , A., Rohrbach , M., Elhoseiny , M., Dsouza , S., Ajanthan , T., & Dokania , P. K. (2019) Efficient lifelong learning with a-gem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* pages 1396–1405.

[6] Chen , J., Liao , K., Wan , Y., Chen , D. Z., & Wu , J. (2022) Danets: Deep abstract networks for tabular data classification and regression. *AAAI Technical Track on Data Mining and Knowledge Management*

[7] Chen , T. & Guestrin , C. (2016) Xgboost: A scalable tree boosting system. *CoRR* **abs/1603.02754**.

[8] De Lange , A. & others (2021) A survey on continual learning. *Journal of Machine Learning* **45**(4): 234–256.

[9] Demšar , J. (2006) Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7**: 1–30.

[10] ElmorLabs . Pmd-usb (power measurement device with usb). 2023. Accessed: 2025-01-15.

[11] ElmorLabs . Pmd pci-e slot power measurement adapter. 2025. Accessed: 2025-01-15.

[12] Esaki , Y., Koide , S., & Kutsuna , T. (2024) One-shot domain incremental learning. In *2024 International Joint Conference on Neural Networks (IJCNN)* page 1–8. IEEE.

[13] French , R. M. (1999) Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences* **3** (4):128–135.

[14] García-Santaclara , P., Fernández-Castro , B., & Díaz-Redondo , R. P. (2024) Overcoming catastrophic forgetting in tabular data classification: A pseudorehearsal-based approach. *arXiv preprint arXiv:2407.09039*

[15] Gorishniy , Y., Rubachev , I., Khrulkov , V., & Babenko , A. (2021) Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems (NeurIPS) 34*, pp. 18932–18943.

[16] Gorishniy , Y., Rubachev , I., Khrulkov , V., & Babenko , A. (2023) Revisiting deep learning models for tabular data. *NeurIPS*

[17] He , X. & others (2023) Seekr: Selective attention for retaining knowledge in continual learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*

[18] Heguerte , L. B., Bugeau , A., & Lannelongue , L. (2023) How to estimate carbon footprint when training deep learning models? a guide and review. *Environmental Research Communications* **5**(11):115014.

[19] Henderson , P., Hu , J., Romoff , J., Brunskill , E., Jurafsky , D., & Pineau , J. (2020) Towards the systematic reporting of the energy and carbon footprints of machine learning. In *Proceedings of the Workshop on Challenges in Deploying and monitoring Machine Learning Systems (EMNLP)* arXiv:2002.05651.

[20] Holm , S. (1979) A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics* **6**(2):65–70.

[21] Jana , S., Li , H., Yamada , Y., & Lindenbaum , O. (2023) Support recovery with projected stochastic gates: Theory and application for linear models. *Signal Processing* **213**:109193.

[22] Jha , S. & others (2023) Neural processes for continual learning. In *International Conference on Machine Learning*

[23] Kadra , A., Lindauer , M., Hutter , F., & Grabocka , J. (2021) Well-tuned simple nets excel on tabular datasets. *NeurIPS*

[24] Kang , D.-K. (2023) Energy-efficient and timeliness-aware continual learning management system. *Energies* **16**(24):8018.

[25] Ke , G., Meng , Q., Finley , T., Wang , T., Chen , W., Ma , W., Ye , Q., & Liu , T.-Y. (2017) Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* page 3149–3157, Red Hook, NY, USA: Curran Associates Inc.

[26] Kemker , R. & Kanan , C. (2018) Fearnet: Brain-inspired model for incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* pages 463–472.

[27] Kim , Y. & others (2022) Pseudo-replay via latent sampling for incremental learning. In *European Conference on Computer Vision*

[28] Kirkpatrick , J., Pascanu , R., Rabinowitz , N. C., Veness , J., Desjardins , G., Rusu , A. A., Milan , K., Quan , J., Ramalho , T., Grabska-Barwińska , A., & others (2017) Overcoming catastrophic forgetting in neural networks. In *Proceedings of the National Academy of Sciences 114*, pp. 3521–3526.

[29] Li , S., Yuan , G., Wu , Y., Dai , Y., Wang , T., Wu , C., Jones , A. K., Hu , J., Wang , Y., & Tang , X. (2024) ETuner: Redundancy-aware efficient continual learning on edge devices. *arXiv preprint arXiv:2401.16694*

[30] Lopez-Paz , D. & Ranzato , M. (2017) Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems* pages 6467–6476.

[31] Lu , Z. & others (2018) Concept drift and its challenges in data streams. *Proceedings of the ACM SIGKDD Conference*

[32] Masana , M. & others (2022) An analysis of rehearsal-based approaches for continual learning. *IEEE Transactions on Neural Networks and Learning Systems*

[33] McElfresh , D. & Talwalkar , A. (2023) Tabzilla benchmark. *NeurIPS* Version 1.0, accessed May 2025.

[34] McElfresh , D., Khandagale , S., Valverde , J., C , V. P., Feuer , B., Hegde , C., Ramakrishnan , G., Goldblum , M., & White , C. (2023) When do neural nets outperform boosted trees on tabular data?. In *Advances in Neural Information Processing Systems (NeurIPS) 2023, Track on Datasets and Benchmarks*

[35] Parisi , G. I., Kemker , R., Part , J. L., Kanan , C., & Wermter , S. (2019) Continual lifelong learning with neural networks: A review. *Neural Networks* **113**:54–71.

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

[37] Pellegrini , F. & others (2020) Latent replay for on-device continual learning. *IEEE Transactions on Neural Networks and Learning Systems*

[38] Prokhorenkova , L., Gusev , G., Vorobev , A., Dorogush , A. V., & Gulin , A. (2018) Catboost: unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* page 6639–6649, Red Hook, NY, USA: Curran Associates Inc.

[39] Rebuffi , S. A., Kolesnikov , A., Sperl , G., & Lampert , C. H. (2017) icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* pages 2001–2010.

[40] Robins , A. (1995) Catastrophic forgetting, rehearsal and pseudorehearsal. In *Connectionist Models* pages 1–11.

[41] Rusu , A. A., Rabinowitz , N. C., Desjardins , G., Soyer , H., Kirkpatrick , J., Kavukcuoglu , K., Pascanu , R., & Hadsell , R. (2022) Progressive neural networks. *NeurIPS*

[42] Shafiee , M. J., Chywl , B., Li , F., & Wong , A. (2018) Netscore: Towards universal metrics for large-scale performance evaluation of deep neural networks. In *NeurIPS Workshop*

[43] Shi , H. & Wang , H. (2023) A unified approach to domain incremental learning with memory: theory and algorithm. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* Red Hook, NY, USA: Curran Associates Inc.

[44] Shin , H., Lee , J. K., Kim , J., Kim , J., & Kim , S. (2017) Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems* pages 2990–2999.

[45] Stoychev , S. & others (2023) Latent generative replay for continual learning. In *International Conference on Learning Representations*

[46] Trinci , T., Magistri , S., Verdecchia , R., & Bagdanov , A. D. (2024) How green is continual learning, really? analyzing the energy consumption in continual training of vision foundation models. *arXiv preprint arXiv:2409.18664* Accepted to GreenFOMO Workshop at ECCV 2024.

[47] Vaswani , A., Shazeer , N., Parmar , N., Uszkoreit , J., Jones , L., Gomez , A. N., Kaiser , L., & Polosukhin , I. (2017) Attention is all you need. *CoRR* **abs/1706.03762**.

[48] Verwimp , T. & others (2021) Rehearsal strategies in continual learning: Limits and merits. *Machine Learning* **110**:1–19.

[49] Wang , L., Zhang , X., Su , H., & Zhu , J. (2024) A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*

[50] Wang , Z. & others (2022) L2p: Learning to prompt for continual vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*

[51] Wilcoxon , F. (1945) Individual comparisons by ranking methods. *Biometrics Bulletin* **1**(6):80–83.

[52] Yang , Z., Adamek , K., & Armour , W. (2024) Accurate and convenient energy measurements for gpus: A detailed study of nvidia gpu's built-in power sensor. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis* page 1–17. IEEE.

[53] Yoon , J., Zhang , Y., Jordon , J., & Schaar , M. (2020) Vime: extending the success of self- and semi-supervised learning to tabular domain. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* Red Hook, NY, USA: Curran Associates Inc.

A Extended Experimental Setup

A.1 Datasets and Stream Segmentation

We evaluate IMLP on 36 classification tasks from the TabZilla benchmark [33], selected from OpenML based on three criteria: (1) sufficient data size to create meaningful segments, (2) balanced representation of binary and multi-class problems, and (3) diverse feature dimensionalities and class distributions. To simulate the data stream in incremental learning scenarios, Table 5 lists every OpenML task in our benchmark together with basic statistics and the fixed stream segmentation applied *in original row order* (rows $1 \dots k$ form Segment 0, rows $k{+}1 \dots 2k$ form Segment 1, etc.).

† Class counts show *label ID : instances* after preprocessing. Binary tasks list two numbers; multi-class tasks list one count per class. For tasks with many classes, we show representative counts or use compact notation (e.g., "25 × 300" for 25 classes with 300 instances each).

A.1.1 Stream Segmentation Algorithm

Our segmentation follows a principled approach to create balanced segments while minimizing data waste:

Algorithm 2 Optimal Segment Size Selection

```
Require: Dataset with $N$ training instances, bounds $k_{\min} = 500$, $k_{\max} = 1000$
Ensure: Segment size $k^*$ that minimizes remainder
 1: best_remainder ← $N$
 2: $k^* \leftarrow k_{\min}$
 3: for $k = k_{\min}$ to $\min(k_{\max}, N)$ do
 4:     num_segments ← $\lfloor N/k \rfloor$
 5:     remainder ← $N \bmod k$
 6:     if remainder $= 0$ then
 7:         return $k$                                  ▷ Perfect division found
 8:     if remainder $<$ best_remainder then
 9:         best_remainder ← remainder
10:         $k^* \leftarrow k$
11: return $k^*$
```

The choice of segment size bounds (500–1000 instances) balances three considerations: (1) *statistical power*, each segment must contain sufficient samples for reliable learning, (2) *IMLP coherence*, segments should be large enough for the attention mechanism to learn meaningful feature relationships within each temporal chunk, and (3) *computational efficiency*, larger segments would increase memory requirements and training time per segment without proportional benefits.

When the optimal segment size k^* leaves a remainder $r = N \bmod k^*$, we apply *round-robin redistribution*: the first r segments each receive one additional instance, ensuring segment sizes differ by at most 1. This maintains temporal ordering while achieving optimal balance.

A.2 Data Retrieval and Preprocessing Protocol

A.2.1 Dataset Acquisition

All datasets are retrieved via the OpenML Python API (v0.15.2) with local caching enabled. We use the default target attribute specified in each OpenML task definition. Raw data is downloaded in DataFrame format to preserve both feature names and categorical indicators.

A.2.2 Feature Preprocessing Pipeline

Our preprocessing pipeline follows scikit-learn best practices with separate transformers for numerical and categorical features:

Numerical Features:

Table 5: OpenML classification tasks and stream-segmentation parameters used in this study. Numbers are produced by the data-processing pipeline and reproduced by the helper script in §A.3.

ID	Name	Inst.	Feat.	Class balance[†]	Seg. size	#Segs
146820	wilt	4,839	5	4,578; 261	514	8
14964	artificial-characters	10,218	7	1,196; 600; 1,192; 1,416; 808; 1,008; ...	579	15
14969	GesturePhaseSegmentation	9,873	32	2,741; 998; 2,097; 1,087; 2,950	839	10
14951	eeg-eye-state	14,980	14	8,257; 6,723	749	17
146206	magic	19,020	10	12,332; 6,688	951	17
167211	Satellite	5,100	36	75; 5,025	867	5
167141	churn	5,000	29	4,293; 707	850	5
168910	fabert	8,237	800	933; 1,433; 1,927; 1,515; 979; 948; 502	500	14
168912	sylvine	5,124	20	2,562; 2,562	871	5
190410	philippine	5,832	308	2,916; 2,916	708	7
2074	satimage	6,430	36	1,531; 703; 1,356; 625; 707; 1,508	683	8
28	optdigits	5,620	64	554; 571; 557; 572; 568; 558; ...	597	8
32	pendigits	10,992	16	1,143; 1,143; 1,144; 1,055; 1,144; ...	519	18
146607	SpeedDating	8,378	442	6,998; 1,380	712	10
168908	christine	5,418	1,611	2,709; 2,709	921	5
14952	PhishingWebsites	11,055	38	4,898; 6,157	522	18
3510	JapaneseVowels	9,961	14	1,096; 991; 1,614; 1,473; 782; ...	529	16
3735	pollen	3,848	5	1,924; 1,924	545	6
3711	elevators	16,599	18	5,130; 11,469	641	22
3896	ada_agnostic	4,562	48	3,430; 1,132	646	6
14970	har	10,299	561	1,722; 1,544; 1,406; 1,777; 1,906; 1,944	547	16
3686	house_16H	22,784	16	6,744; 16,040	842	23
3897	eye_movements	10,936	27	3,804; 4,262; 2,870	715	13
3904	jm1	10,885	21	8,779; 2,106	514	18
43	spambase	4,601	57	2,788; 1,813	782	5
3954	MagicTelescope	19,020	10	12,332; 6,688	951	17
9952	phoneme	5,404	5	3,818; 1,586	574	8
3950	musk	6,598	267	5,581; 1,017	701	8
9960	wall-robot-navigation	5,456	24	2,205; 2,097; 328; 826	515	9
3889	sylva_agnostic	14,395	216	13,509; 886	941	13
9985	first-order-theorem-proving	6,118	51	1,089; 486; 748; 617; 624; 2,554	520	10
3481	isolet	7,797	617	25×300 (class 0...24)	552	12
45	splice	3,190	227	767; 768; 1,655	542	5
9986	gas-drift	13,910	128	2,565; 2,926; 1,641; 1,936; 3,009; 1,833	563	21
9987	gas-drift-different-conc.	13,910	129	2,565; 2,926; 1,641; 1,936; 3,009; 1,833	563	21
168909	dilbert	10,000	2,000	1,988; 2,049; 1,913; 2,046; 2,004	500	17

1. **Imputation**: Missing values filled with column medians
2. **Standardization**: Zero mean, unit variance scaling via StandardScaler

Categorical Features:

1. **Imputation**: Missing values filled with constant ‘missing’
2. **Encoding**: One-hot encoding with `drop=‘first’` to avoid multicollinearity

3. **Unknown handling**: `handle_unknown='ignore'` for robust inference

The ColumnTransformer ensures preprocessing consistency across all data splits. After transformation, all features are converted to `float32` for memory efficiency.

A.2.3 Target Processing and Task Type Detection

Target variables are processed based on OpenML task type:

- **Binary classification**: 2 unique labels $\rightarrow$ LabelEncoder $\rightarrow$ {0, 1}
- **Multi-class classification**: $C > 2$ unique labels $\rightarrow$ LabelEncoder $\rightarrow$ {0, . . . , C-1}
- **Regression**: Direct conversion to float32 (not used in this study)

A.2.4 Data Splitting Strategy

Our splitting protocol ensures realistic evaluation:

1. **Test Set Isolation**: A stratified 15% test split is carved out *before* any stream processing, using `random_seed=42` for reproducibility.
2. **Training Stream Creation**: The remaining 85% forms the chronologically ordered training stream, preserving the original row order from OpenML.
3. **Per-Segment Validation**: Each segment (or cumulative data) is further split with stratified 15% validation, using `random_seed=42+segment_idx` to ensure different splits per segment while maintaining reproducibility.

This approach simulates realistic continual learning where: 1) The test set represents future unseen data, 2) Each segment represents a temporal chunk of arriving data, 3) Validation splits enable early stopping without future data leakage, and 4) All models use consistent 15% validation splits for hyperparameter selection and early stopping

A.2.5 Model Training Protocols

Our experimental design follows two distinct training protocols based on model type:

Cumulative Training (Baseline Models): Traditional baselines (XGBoost, LightGBM, CatBoost, kNN, SVM, Decision Trees, Random Forest, and neural baselines like TabNet, SAINT) are retrained from scratch at each segment using all available data up to that point. For the segment, these models train on the union $\bigcup_{t=0}^{T} S_t$ where S_t denotes the t-th data segment. This protocol maximizes baseline performance by leveraging all historical data, representing the standard approach in tabular learning.

Incremental Training (IMLP): Our proposed IMLP trains only on the current segment S_k while accessing previous feature representations through the attention mechanism. This protocol tests true incremental learning capabilities without replay of raw historical data.

Both protocols use identical validation procedures: each model's hyperparameters are selected via early stopping on the 15% validation split, ensuring fair comparison despite different training paradigms.

A.2.6 Reproducibility Measures

All steps are deterministic with fixed random seeds, including 1) Global seed: `random_seed = 42`, 2) Per-segment validation: `random_seed = 42 + segment_idx`, and 3) Preprocessing: Deterministic transformers with fixed parameters.

A.3 Dataset Summary Regeneration Script

For full reproducibility, we provide a helper script that regenerates Table 5 from the processed data:

```python
# dataset_summary.py  (runs in < 2 seconds)
import json, csv, gzip, numpy as np, pathlib

def regenerate_dataset_summary():
    """Regenerate the dataset summary CSV from processed metadata."""
    META = pathlib.Path("processed_datasets_summary.json")
    ROOT = pathlib.Path("full_datasets")
    OUT  = pathlib.Path("dataset_summary.csv")

    # Load processing metadata
    with META.open() as f:
        meta = json.load(f)

    rows = []
    for tid, m in meta.items():
        # Load target labels to compute class balance
        y = np.load(gzip.open(ROOT/m["dataset_name"]/"y_full.npy.gz"))
        counts = np.bincount(y.astype(int))

        rows.append({
            "task_id": int(tid),
            "name": m["original_name"],
            "instances": int(m["num_instances"]),
            "features": int(m["num_features"]),
            "class_balance": ";".join(map(str, counts)),
            "segment_size": int(m["segment_size"]),
            "num_segments": int(m["num_segments"])
        })

    # Write CSV output
    with OUT.open("w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=rows[0].keys())
        writer.writeheader()
        writer.writerows(rows)

    print(f"Wrote {OUT} with {len(rows)} tasks")

if __name__ == "__main__":
    regenerate_dataset_summary()
```

Running this script in the project root recreates the CSV that backs Table 5. The script requires the preprocessed datasets but no pipeline re-execution.

B IMLP Implementation Details

B.1 Architecture Overview and Design Rationale

IMLP extends the standard MLP architecture with an attention-based memory mechanism designed specifically for tabular continual learning. The key innovation lies in storing and retrieving *feature representations* rather than raw data, enabling privacy-preserving incremental learning with constant memory requirements.

B.1.1 Comparison with Standard MLP

Table 7 contrasts IMLP with a standard MLP of equivalent capacity:

Table 7: Architectural comparison between standard MLP and IMLP.

Component	**MLP**	**IMLP**	**IMLP Notes**
Input processing	$d_{\text{in}} \to 512$	$d_{\text{in}} \to 256$	Query projection
Memory mechanism	None	Attention	Key-value retrieval
Feature extraction	$512 \to 256$	$(d_{\text{in}} + 256) \to 512 \to 256$	Context-augmented
Memory complexity	$\mathcal{O}(1)$	$\mathcal{O}(W)$	W = window size
Time complexity	$\mathcal{O}(1)$	$\mathcal{O}(W \cdot d)$	d = hidden dim
Privacy	Requires raw data	Feature-only	No raw data storage

B.2 Layer-wise Architecture Specification

Table 8: Detailed layer-wise specification of IMLP architecture.

Component	**Output dim.**	**Activation**	**Notes**
Input feature vector	d_{in}	–	Raw tabular features after preprocessing
Attention Module			
Query projection Q	256	–	Linear$(d_{\text{in}}, 256)$
Key projection K	256	–	Linear$(256, 256)$ applied to each stored feature
Context computation	256	–	Scaled dot-product attention over window
Feature Extraction			
Concatenated input (x, c)	$d_{\text{in}} + 256$	–	Only if attention enabled; c = context vector
FC 1	512	ReLU	Linear$(d_{\text{in}} + 256, 512)$
FC 2	256	ReLU	Linear$(512, 256)$
Classification Head			
Classifier	C	–	Linear$(256, C)$ where C = number of classes

Design Choices:

- **Hidden size 256**: Balances expressiveness with computational efficiency across all datasets
- **No dropout/normalization**: Empirically found to hurt performance in continual learning setting
- **ReLU activations**: Simple, stable gradients for incremental training
- **Fixed architecture**: Same capacity across all 36 datasets for fair comparison

B.3 Attention Mechanism Design

B.3.1 Scaled Dot-Product Attention

IMLP uses a simplified attention mechanism to retrieve relevant historical features. For a batch of size B:

$$Q = W_q \cdot x \in \mathbb{R}^{B\times 1\times 256} \quad \text{(query from current input)} \tag{3}$$
$$K = W_t \cdot H_{\text{stacked}} \in \mathbb{R}^{B\times W\times 256} \quad \text{(keys from previous features)} \tag{4}$$
$$\text{Scores} = \text{bmm}(K, Q^T) \in \mathbb{R}^{B\times W\times 1} \tag{5}$$
$$\alpha = \text{softmax}(\text{Scores.squeeze()}) \in \mathbb{R}^{B\times W} \tag{6}$$
$$\text{Context} = \text{bmm}(\alpha.\text{unsqueeze}(1), K) \in \mathbb{R}^{B\times 1\times 256} \tag{7}$$

where:

- $H_{\text{stacked}} = \text{stack}(\{h_{t-W}, \dots, h_{t-1}\}) \in \mathbb{R}^{B\times W\times 256}$
- bmm denotes batch matrix multiplication
- No scaling factor is applied (unlike standard scaled dot-product attention)
- Values equal keys: $V = K$

B.3.2 Window Management Strategy

The sliding window maintains a FIFO queue of the most recent W feature vectors:

Algorithm 3 Sliding Window Update

```
Require: Current input $x$, previous features $H_{\text{prev}}$, window size $W$
Ensure: Updated window $H_{\text{new}}$
1: $h_{\text{current}} \leftarrow \text{FeatureExtractor}(x, \text{Context}(x))$
2: $H_{\text{new}} \leftarrow H_{\text{prev}} \cup \{h_{\text{current}}\}$
3: if $|H_{\text{new}}| > W$ then
4:     $H_{\text{new}} \leftarrow H_{\text{new}}[1:]$                ▷ Remove oldest feature
5: return $H_{\text{new}}$
```

B.3.3 Feature Normalization

To improve attention stability, stored features are L2-normalized during precomputation:

$$\tilde{h}_i = \frac{h_i}{\|h_i\|_2 + \epsilon} \tag{8}$$

where $\epsilon = 10^{-8}$ prevents division by zero. This normalization ensures attention weights focus on feature directions rather than magnitudes and is applied in the `_precompute` method during segmental training.

B.4 Complete Implementation

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class IncrementalMLP(nn.Module):
    """
    Incremental MLP with attention-based feature replay for continual learning.

    Args:
        input_size (int): Number of input features
        num_classes (int): Number of output classes
        use_attention (bool): Whether to use attention mechanism
        window_size (int): Size of sliding memory window
    """

    def __init__(self, input_size, num_classes, use_attention=True, window_size=10):
        super().__init__()
        self.window_size = window_size
        self.use_attention = use_attention
        self.hidden_size = 256

        # Attention projections
        self.query = nn.Linear(input_size, 256)
        self.key = nn.Linear(256, 256)

        # Feature extraction pathway
        total_input_size = input_size + (256 if use_attention else 0)
        self.feature_extractor = nn.Sequential(
            nn.Linear(total_input_size, 512),
            nn.ReLU(),
            nn.Linear(512, self.hidden_size),
            nn.ReLU()
        )

        # Classification head
        self.classifier = nn.Linear(self.hidden_size, num_classes)

    def compute_context(self, x, prev_features):
        """
        Compute attention-weighted context from previous features.

        Args:
            x (Tensor): Current input batch [B, D]
            prev_features (List[Tensor]): Previous feature vectors [W x [256]]

        Returns:
            Tensor: Context vector [B, 256]
        """
        if not prev_features or self.window_size == 0:
            return torch.zeros(x.size(0), 256, device=x.device)

        # Stack previous features: [B, W, 256]
        stacked_prev = torch.stack(prev_features, dim=1)

        # Compute keys and queries
        keys = self.key(stacked_prev)  # [B, W, 256]
        query = self.query(x).unsqueeze(1)  # [B, 1, 256]

        # Scaled dot-product attention
        scores = torch.bmm(keys, query.transpose(1, 2)).squeeze(-1)  # [B, W]
        attention_weights = F.softmax(scores, dim=1)  # [B, W]

        # Compute weighted context
        context = torch.bmm(attention_weights.unsqueeze(1), keys).squeeze(1)  # [B,
        ↪  256]

        return context

    def forward(self, x, prev_features=None):
        """
        Forward pass with optional attention over previous features.

        Args:
            x (Tensor): Input features [B, D]
            prev_features (List[Tensor]): Previous features for attention

        Returns:
            Tuple[Tensor, Tensor]: (logits, current_features)
        """
        # Compute attention context
        context = torch.zeros(x.size(0), 256, device=x.device)
        if self.use_attention and prev_features:
            context = self.compute_context(x, prev_features)

        # Concatenate input with context
        if self.use_attention:
            augmented_input = torch.cat([x, context], dim=1)
        else:
            augmented_input = x

        # Extract features and classify
        features = self.feature_extractor(augmented_input)
        logits = self.classifier(features)

        return logits, features
```

B.5 Computational Complexity Analysis

B.5.1 Time Complexity

For each forward pass with batch size B, input dimension d_{in}, hidden dimension $d_h = 256$, and window size W:

$$\begin{aligned}
\text{Query projection:} &\quad \mathcal{O}(B \cdot d_{\text{in}} \cdot d_h) && (9)\\
\text{Key projection:} &\quad \mathcal{O}(B \cdot W \cdot d_h^2) && (10)\\
\text{Attention scores:} &\quad \mathcal{O}(B \cdot W \cdot d_h) && (11)\\
\text{Context aggregation:} &\quad \mathcal{O}(B \cdot W \cdot d_h) && (12)\\
\text{Feature extraction:} &\quad \mathcal{O}(B \cdot (d_{\text{in}} + d_h) \cdot 512) && (13)\\
\hline
\text{Total:} &\quad \mathcal{O}(B \cdot (d_{\text{in}} \cdot d_h + W \cdot d_h^2)) && (14)
\end{aligned}$$

For typical values ($W = 10$, $d_h = 256$, $d_{\text{in}} \lesssim 2000$), the attention overhead is $\mathcal{O}(W \cdot d_h^2) = \mathcal{O}(655{,}360)$ operations per sample.

B.5.2 Memory Complexity

IMLP maintains constant memory usage per segment:

- **Model parameters**: $\approx$ 1.2M parameters (fixed)
- **Feature buffer**: $W \times 256 \times 4$ bytes = 10,240 bytes for $W = 10$
- **Attention matrices**: $B \times W \times 256 \times 4$ bytes during computation

Unlike replay-based methods, memory usage does not grow with the number of segments, enabling indefinite continual learning.

B.5.3 Comparison with Replay Methods

Table 9 compares IMLP with alternative continual learning approaches:

Table 9: Complexity comparison of continual learning approaches.

Method	Memory	Time per step	Privacy
Naive retraining	$\mathcal{O}(T \cdot N)$	$\mathcal{O}(T \cdot N)$	Requires raw data
Experience replay	$\mathcal{O}(M)$	$\mathcal{O}(N + M)$	Requires raw data
Generative replay	$\mathcal{O}(1)$	$\mathcal{O}(N + G)$	Private
IMLP (ours)	$\mathcal{O}(W)$	$\mathcal{O}(N + W \cdot d^2)$	Private

where T = number of tasks, N = samples per task, M = replay buffer size, G = generative model cost, W = window size, d = feature dimension.

B.6 Hyperparameter Configuration

IMLP uses the following default hyperparameters across all experiments:

Table 10: IMLP hyperparameter configuration.

Parameter	Value	Description
Window size (W)	10	Number of previous feature vectors stored
Hidden dimension	256	Feature representation size
Learning rate	10^{-3}	Adam optimizer learning rate
Batch size	128	Training batch size
Weight decay	10^{-5}	L2 regularization strength
Early stopping patience	10	Epochs without improvement before stopping
Max epochs	100	Maximum training epochs per segment
Normalization ϵ	10^{-8}	Small constant for L2 normalization

The window size $W = 10$ was chosen to balance memory efficiency with sufficient historical context. The hidden dimension of 256 provides adequate representational capacity while maintaining computational efficiency across diverse tabular datasets.

C Deriving and Computing NetScore-T

C.1 Mathematical Derivation

NetScore-T extends the NetScore framework [42] to continual learning by jointly evaluating predictive performance and energy consumption across data segments. Our formulation balances accuracy maximization with energy minimization through a logarithmic penalty function.

C.1.1 Per-Segment NetScore-T

For a given model m and the t^{th} data segment $\mathcal{S}_t$, we define the per-segment NetScore-T as:

$$\mathrm{NS}_t^{(m)} = \frac{P_t^{(m)}}{\log_{10}(E_t^{(m)} + 1)} \tag{15}$$

where:

- $P_t^{(m)}$ is the performance metric (balanced accuracy or transformed log-loss)
- $E_t^{(m)}$ is the total energy consumption in Joules for segment k
- The logarithmic denominator compresses the wide range of energy values
- The $+1$ term prevents division by zero for very low energy consumption

C.1.2 Energy Composition

Total energy consumption combines training and inference components:

$$E_t^{(m)} = E_{\text{train},t}^{(m)} + \alpha \cdot E_{\text{infer},t}^{(m)} \tag{16}$$

where α weights the relative importance of inference versus training energy. In this study, we use $\alpha = 1$ (balanced weighting) and do not tune this parameter. Future work could explore different α values to reflect specific deployment scenarios, for instance, $\alpha > 1$ for inference-heavy applications or $\alpha < 1$ for training-dominated workflows.

C.1.3 Stream-Level NetScore-T

To evaluate performance across an entire data stream of T segments, we compute the arithmetic mean:

$$\text{NetScore-T}^{(m)} = \frac{1}{T} \sum_{t=1}^{T} \mathrm{NS}_t^{(m)} \tag{17}$$

This formulation treats all segments equally, reflecting the assumption that each temporal chunk has similar importance in continual learning evaluation.

C.2 Performance Metric Transformations

NetScore-T accommodates different performance metrics through appropriate transformations:

C.2.1 Balanced Accuracy

For balanced accuracy $\mathrm{BA}_t^{(m)} \in [0, 1]$, we use the metric directly:

$$P_t^{(m)} = \mathrm{BA}_t^{(m)} \tag{18}$$

C.2.2 Log-Loss Transformation

Since log-loss is a “lower-is-better” metric, we transform it to a “higher-is-better” quantity:

$$P_t^{(m)} = \frac{1}{\mathcal{L}_t^{(m)} + \varepsilon} \tag{19}$$

where $\mathcal{L}_t^{(m)}$ is the log-loss value and $\varepsilon = 10^{-7}$ prevents division by zero for perfect predictions. This transformation ensures larger NetScore-T values indicate better performance for both metrics.

C.3 Mathematical Properties

C.3.1 Boundedness and Range

The per-segment NetScore-T has the following properties:

- **Lower bound**: $\mathrm{NS}_t^{(m)} \geq 0$ (since $P_t^{(m)} \geq 0$ and denominator > 0)
- **Upper bound**: No finite upper bound exists. As $E_t^{(m)} \to 0$, the denominator approaches $\log_{10}(1) = 0$, causing $\mathrm{NS}_t^{(m)} \to \infty$
- **Empirical ranges**: In our experiments, NetScore-T values span:
  - Balanced accuracy version: $[0.06, 4.00]$
  - Log-loss version: $[0.08, 108.61]$

The wide empirical ranges reflect the diversity of energy consumption patterns across models and datasets. High values (e.g., 108.61) occur when models achieve reasonable accuracy with very low energy consumption.

C.4 Implementation Details

C.4.1 Energy Measurement Protocol

Energy consumption $E_t^{(m)}$ is measured using an ElmorLabs PMD-USB power meter sampling between $500 - 800$ (most of the time ~ 700) Hz. Total energy is computed by integrating power readings over the training and inference duration for each segment:

$$E_t^{(m)} = \int_{t_{\text{start}}}^{t_{\text{end}}} P_{\text{total}}(t)\, dt \tag{20}$$

where $P_{\text{total}}(t)$ includes both CPU and GPU power consumption.

C.4.2 Computational Procedure

Algorithm 4 outlines the NetScore-T computation process:

Algorithm 4 NetScore-T Computation

```
Require: Performance metrics $\{P_t^{(m)}\}_{t=1}^T$, energy measurements $\{E_t^{(m)}\}_{t=1}^T$
Ensure: Stream-level NetScore-T score
1: Initialize scores $\leftarrow []$
2: for $t = 1$ to $T$ do
3:     $\mathrm{NS}_t^{(m)} \leftarrow P_t^{(m)} / \log_{10}(E_t^{(m)} + 1)$
4:     scores.append($\mathrm{NS}_t^{(m)}$)
5: return $\frac{1}{T}\sum_{i=1}^{T}$ scores$[i]$
```

D Complete Per-Dataset Results

We present per-dataset results for all models and metrics evaluated in our study. The tables below show performance across the 36 TabZilla datasets for six key metrics: balanced accuracy, log-loss, NetScore-T (balanced accuracy), NetScore-T (log-loss), wall-time, and energy consumption. Best values in each row are highlighted in bold.

All results represent the final performance after training on the complete stream (i.e., performance on the test set after processing all segments). For cumulative models, this corresponds to training on all available data; for IMLP, this represents performance after incremental learning across all segments.

D.1 Balanced Accuracy Results

Table 11 shows the balanced accuracy achieved by each model on each dataset. Balanced accuracy is computed as the average of recall scores for each class, providing a metric robust to class imbalance.

Table 11: Balanced accuracy per dataset and model. Values range from 0 to 1, with higher values indicating better performance. Best values per row are shown in bold.

Dataset	CatBoost	DaNet	DecisionTree	IMLP	IMLP_C	kNN	LightGBM	LinearModel	MLP	MLP_C	RandomForest	ResNet	STG	SVM	TabNet	VIME	XGBoost
146206	0.85	0.84	0.79	0.85	0.83	0.79	**0.86**	0.76	0.86	0.85	0.80	0.86	0.67	0.85	0.85	0.74	0.79
146607	0.65	0.65	0.63	0.66	0.70	0.59	0.65	0.67	**0.70**	0.67	0.51	0.50	0.50	0.66	0.62	0.59	0.63
146820	0.79	0.82	0.91	0.86	0.50	0.64	0.85	0.66	**0.92**	0.92	0.74	0.88	0.50	0.88	0.82	0.50	0.92
14951	0.80	0.67	0.69	0.72	0.69	0.83	**0.94**	0.57	0.73	0.84	0.75	0.85	0.50	0.62	0.53	0.83	0.70
14952	0.95	0.95	0.93	0.92	0.93	0.95	**0.97**	0.93	0.96	0.93	0.91	0.96	0.63	0.95	0.96	0.96	0.93
14964	0.63	0.60	0.39	0.58	0.58	0.66	**0.89**	0.34	0.63	0.56	0.49	0.59	0.16	0.60	0.63	0.59	0.47
14969	0.47	0.49	0.41	0.44	0.41	0.52	**0.64**	0.37	0.50	0.47	0.38	0.48	0.30	0.47	0.40	0.39	0.42
14970	0.97	0.98	0.87	0.96	0.96	0.96	**0.99**	0.98	0.98	0.98	0.92	0.96	0.32	0.98	0.99	0.98	0.92
167141	**0.89**	0.81	0.82	0.83	0.81	0.60	0.50	0.58	0.80	0.77	0.62	0.85	0.46	0.76	0.86	0.50	0.82
167211	0.77	0.82	0.82	**0.91**	0.82	0.77	0.77	0.86	0.82	0.82	0.82	0.77	0.49	0.82	0.68	0.50	0.86
168908	0.73	0.72	0.66	0.72	0.74	0.69	0.71	0.68	0.72	0.73	0.70	0.70	0.55	**0.74**	0.65	0.73	0.67
168909	0.90	0.98	0.65	0.95	0.94	0.94	**0.99**	0.92	0.98	0.94	0.81	0.97	0.28	0.98	0.97	0.98	0.77
168910	0.40	0.55	0.22	0.62	0.62	0.62	**0.67**	0.64	0.64	0.47	0.26	0.62	0.14	0.65	0.56	0.56	0.36
168912	**0.94**	0.94	0.94	0.93	0.93	0.83	0.94	0.92	0.93	0.94	0.93	0.93	0.63	0.93	0.93	0.84	0.94
190410	0.74	0.69	0.73	0.68	0.67	0.68	**0.76**	0.71	0.71	0.72	0.73	0.70	0.57	0.72	0.69	0.71	0.74
2074	0.85	0.87	0.76	0.83	0.81	0.88	**0.88**	0.81	0.88	0.87	0.82	0.85	0.51	0.86	0.86	0.87	0.83
28	0.96	**0.99**	0.70	0.97	0.97	0.98	0.98	0.97	0.98	0.98	0.94	0.98	0.17	0.98	0.98	0.98	0.90
32	0.98	0.99	0.82	0.98	0.98	0.99	0.99	0.94	**0.99**	0.99	0.90	0.98	0.33	0.99	0.99	0.94	0.92
3481	0.91	0.93	0.52	0.93	0.93	0.88	0.94	0.95	0.95	0.95	0.87	0.96	0.06	**0.96**	0.92	0.95	0.81
3510	0.92	0.97	0.67	0.95	0.95	0.97	0.98	0.94	**0.98**	0.98	0.83	0.96	0.20	0.98	0.98	0.95	0.80
3686	0.85	0.86	0.80	0.83	0.82	0.83	**0.86**	0.76	0.86	0.86	0.82	0.85	0.60	0.85	0.83	0.77	0.80
3711	0.78	0.83	0.73	0.83	0.84	0.75	0.83	0.85	0.86	**0.87**	0.73	0.85	0.71	0.85	0.86	0.79	0.73
3735	0.51	0.50	0.47	0.51	0.51	**0.53**	0.48	0.47	0.52	0.50	0.49	0.47	0.48	0.52	0.53	0.50	0.50
3889	**0.99**	0.98	0.97	0.98	0.99	0.87	0.50	0.97	0.98	0.97	0.77	0.95	0.49	0.93	0.98	0.94	0.97
3896	0.78	0.76	0.76	0.77	**0.80**	0.74	0.76	0.78	0.79	0.78	0.73	0.79	0.52	0.74	0.75	0.69	0.76
3897	0.62	0.64	0.52	0.55	0.54	0.52	**0.75**	0.50	0.59	0.57	0.57	0.61	0.35	0.58	0.59	0.48	0.57
3904	0.57	0.54	0.55	**0.61**	0.59	0.60	0.57	0.55	0.59	0.58	0.55	0.55	0.51	0.55	0.53	0.56	0.55
3950	0.99	0.99	0.91	0.99	0.99	0.90	0.98	**1.00**	0.99	0.99	0.80	0.98	0.44	0.97	0.99	0.76	0.92
3954	0.84	**0.87**	0.79	0.84	0.83	0.79	0.86	0.76	0.86	0.85	0.79	0.86	0.61	0.85	0.85	0.72	0.79
43	0.93	0.92	0.88	0.92	0.93	0.89	**0.94**	0.91	0.92	0.93	0.90	0.92	0.50	0.91	0.92	0.83	0.87
45	0.94	0.91	0.91	0.92	0.92	0.84	0.95	0.94	0.93	0.94	0.90	0.93	0.34	**0.95**	0.93	0.89	0.92
9952	0.81	0.81	0.77	0.77	0.72	0.79	**0.85**	0.64	0.79	0.81	0.77	0.81	0.47	0.77	0.80	0.72	0.78
9960	0.99	0.88	0.95	0.82	0.81	0.86	**0.99**	0.63	0.89	0.88	0.94	0.89	0.23	0.89	0.92	0.66	0.99
9985	0.36	0.39	0.28	0.35	0.40	0.45	**0.47**	0.31	0.40	0.41	0.33	0.40	0.20	0.36	0.34	0.19	0.32
9986	0.97	0.98	0.80	0.98	0.97	0.99	0.99	0.99	**0.99**	0.99	0.87	0.86	0.46	0.98	0.93	0.98	0.91
9987	0.98	0.99	0.82	0.98	0.98	0.99	0.99	0.99	**0.99**	0.99	0.89	0.81	0.49	0.98	0.94	0.99	0.92

D.2 Log-Loss Results

Table 12 presents the logarithmic loss for each model. Log-loss measures the cross-entropy between predicted class probabilities and true class labels, with lower values indicating better calibrated predictions.

Table 12: Log-loss per dataset and model. Lower values indicate better performance. Best values per row are shown in bold.

Dataset	CatBoost	DaNet	DecisionTree	IMLP	IMLP_C	kNN	LightGBM	LinearModel	MLP	MLP_C	RandomForest	ResNet	STG	SVM	TabNet	VIME	XGBoost
146206	0.32	0.30	0.43	0.33	0.39	1.58	0.30	0.44	**0.29**	0.30	0.39	1.00	0.58	0.31	0.30	0.51	0.69
146607	**0.33**	0.35	0.37	0.41	0.98	1.80	0.35	0.35	0.43	0.34	0.37	2.63	0.58	0.33	0.39	1.47	0.69
146820	0.07	0.05	0.51	0.08	0.25	0.55	0.06	0.10	**0.04**	0.04	0.08	0.31	0.61	0.04	0.05	0.86	0.68
14951	0.45	0.59	0.61	0.54	1.07	0.83	**0.15**	0.66	0.50	0.36	0.55	1.28	0.69	0.61	0.67	0.45	0.69
14952	0.12	0.12	0.18	0.19	0.16	0.33	**0.08**	0.15	0.10	0.18	0.26	0.48	0.67	0.13	0.11	0.22	0.69
14964	0.97	0.73	1.49	0.89	0.96	1.85	**0.31**	1.66	0.77	1.02	1.46	4.79	2.28	0.83	0.72	2.37	2.29
14969	1.12	1.12	1.24	1.29	1.43	4.88	**0.83**	1.28	1.12	1.13	1.20	5.05	1.56	1.15	1.19	3.96	1.60
14970	0.14	0.06	0.34	0.11	0.09	0.19	**0.02**	0.04	0.05	0.06	0.33	0.58	1.54	0.06	0.04	0.25	1.77
167141	**0.15**	0.19	0.43	0.31	0.42	1.76	0.30	0.29	0.21	0.22	0.25	0.61	0.67	0.21	0.18	2.25	0.69
167211	0.03	0.04	0.16	0.03	0.02	0.15	0.04	**0.02**	0.03	0.03	0.02	0.10	0.65	0.03	0.04	0.23	0.68
168908	0.54	0.57	1.06	0.63	0.62	1.95	0.55	1.12	0.62	0.57	0.57	2.02	0.69	**0.54**	0.65	0.91	0.69
168909	0.43	**0.04**	1.05	0.15	0.25	0.27	0.05	0.33	0.05	0.32	0.83	0.25	1.58	0.05	0.12	0.15	1.60
168910	1.48	1.07	1.83	1.01	1.22	5.41	**0.83**	2.20	0.96	1.23	1.70	4.49	1.95	0.90	1.12	1.66	1.94
168912	**0.15**	0.16	0.40	0.19	0.23	0.99	0.21	0.22	0.19	0.19	0.29	0.66	0.68	0.19	0.18	0.39	0.69
190410	0.50	0.57	0.60	0.70	0.87	2.09	**0.49**	0.55	0.65	0.60	0.53	2.02	0.68	0.55	0.57	1.10	0.69
2074	0.33	0.25	0.67	0.38	0.36	0.84	**0.24**	0.34	0.26	0.26	0.44	1.43	1.66	0.27	0.32	0.75	1.77
28	0.19	0.07	1.20	0.10	0.10	0.17	0.06	0.09	**0.06**	0.07	0.69	0.21	2.26	0.06	0.09	0.10	2.26
32	0.14	0.05	0.69	0.08	0.07	0.15	0.05	0.20	0.04	0.06	0.47	0.24	2.22	**0.04**	0.05	0.75	2.26
3481	0.49	0.22	1.53	0.26	0.24	0.98	0.19	0.17	0.15	**0.15**	1.20	0.45	3.25	0.16	0.30	0.23	3.17
3510	0.33	0.09	1.10	0.14	0.16	0.10	0.07	0.17	0.05	0.07	0.90	0.50	2.15	0.06	**0.05**	0.28	2.17
3686	0.28	0.28	0.40	0.31	0.37	1.23	0.29	0.41	**0.27**	0.28	0.34	1.19	0.53	0.29	0.30	0.66	0.69
3711	0.36	0.29	0.46	0.38	0.32	1.47	0.32	0.26	0.27	0.27	0.42	1.20	0.52	0.26	**0.25**	1.14	0.69
3735	**0.69**	0.69	0.76	0.71	0.72	1.66	0.70	0.70	0.70	0.69	0.70	0.70	0.70	0.69	0.70	0.70	0.69
3889	**0.02**	0.02	0.06	0.03	0.03	0.34	0.16	0.04	0.03	0.03	0.08	0.19	0.51	0.05	0.03	0.07	0.68
3896	**0.32**	0.34	0.55	0.36	0.38	1.22	0.33	0.34	0.33	0.34	0.38	1.08	0.68	0.37	0.38	1.08	0.69
3897	0.83	0.80	0.96	1.02	1.30	3.36	**0.60**	0.97	0.86	0.87	0.93	3.60	1.10	0.87	0.86	1.38	1.10
3904	**0.44**	0.45	0.56	0.52	0.56	2.02	0.45	0.45	0.45	0.44	0.44	2.64	0.56	0.47	0.44	2.02	0.69
3950	0.04	0.02	0.14	0.03	**0.01**	0.23	0.03	0.02	0.02	0.02	0.16	0.23	0.67	0.04	0.03	1.14	0.68
3954	0.32	**0.29**	0.43	0.33	0.40	1.58	0.30	0.44	0.29	0.31	0.39	1.16	0.60	0.31	0.30	0.56	0.69
43	0.16	0.18	0.35	0.24	0.21	1.11	**0.16**	0.26	0.19	0.19	0.27	0.95	0.69	0.27	0.22	0.26	0.69
45	0.20	0.22	0.46	0.27	0.23	1.36	**0.12**	0.18	0.16	0.16	0.55	0.83	1.09	0.13	0.37	0.23	1.09
9952	0.35	0.36	0.55	0.39	0.41	1.38	**0.31**	0.50	0.36	0.35	0.38	1.35	0.68	0.38	0.38	0.47	0.69
9960	0.04	0.34	0.26	0.39	0.60	1.30	**0.03**	0.68	0.27	0.27	0.27	1.00	1.37	0.29	0.21	2.69	1.37
9985	1.29	1.27	1.70	1.35	1.41	6.73	**1.12**	1.42	1.25	1.24	1.35	5.19	1.77	1.32	1.31	8.63	1.78
9986	0.22	0.07	0.66	0.12	0.14	0.20	**0.05**	0.11	0.05	0.07	0.49	2.22	1.54	0.08	0.22	0.11	1.77
9987	0.19	**0.04**	0.65	0.10	0.12	0.20	0.05	0.10	0.05	0.05	0.48	2.57	1.56	0.08	0.20	0.16	1.77

D.3 NetScore-T (Balanced Accuracy) Results

Table 13 shows the NetScore-T computed using balanced accuracy as the performance metric. This joint metric captures both predictive performance and energy efficiency, with higher values indicating better accuracy-to-energy ratios.

Table 13: NetScore-T (balanced accuracy) per dataset and model. Higher values indicate better accuracy-energy trade-offs. Best values per row are shown in bold.

Dataset	CatBoost	DaNet	DecisionTree	IMLP	IMLP_C	kNN	LightGBM	LinearModel	MLP	MLP_C	RandomForest	ResNet	STG	SVM	TabNet	VIME	XGBoost
146206	0.82	0.25	**1.43**	0.41	0.40	0.84	0.81	0.93	0.34	0.33	0.65	0.34	0.22	0.44	0.26	0.26	0.52
146607	0.49	0.20	**1.25**	0.36	0.34	0.33	0.65	0.45	0.29	0.28	0.51	0.25	0.20	0.31	0.19	0.17	0.36
146820	1.10	0.28	**4.00**	0.31	0.26	1.53	2.53	0.62	0.31	0.29	0.81	0.40	0.27	1.68	0.30	0.22	0.62
14951	0.83	0.22	**1.54**	0.34	0.33	0.98	0.74	0.71	0.25	0.30	0.69	0.32	0.19	0.34	0.20	0.26	0.46
14952	0.97	0.30	**2.83**	0.51	0.49	0.80	1.08	0.97	0.42	0.39	1.01	0.40	0.24	0.72	0.31	0.33	0.66
14964	0.48	0.18	1.10	0.29	0.29	**1.11**	0.46	0.36	0.25	0.21	0.48	0.24	0.05	0.34	0.19	0.18	0.23
14969	0.27	0.14	**0.73**	0.19	0.22	0.43	0.32	0.29	0.19	0.18	0.31	0.18	0.12	0.22	0.13	0.13	0.22
14970	0.34	0.30	0.64	0.51	0.49	0.50	0.40	**0.64**	0.43	0.41	0.60	0.39	0.08	0.44	0.30	0.26	0.36
167141	1.00	0.27	**2.78**	0.39	0.38	0.94	1.40	0.42	0.31	0.29	0.68	0.37	0.21	0.72	0.27	0.21	0.57
167211	1.13	0.28	**3.09**	0.44	0.43	1.41	2.45	0.57	0.33	0.31	0.83	0.36	0.24	1.84	0.28	0.20	0.68
168908	0.30	0.26	0.43	0.34	0.34	0.34	0.31	0.38	0.34	0.33	**0.54**	0.32	0.24	0.28	0.23	0.20	0.26
168909	0.27	0.29	0.31	0.46	0.45	0.34	0.28	**0.49**	0.41	0.37	0.46	0.36	0.08	0.31	0.27	0.23	0.24
168910	0.17	0.15	**0.40**	0.30	0.30	0.23	0.24	0.38	0.25	0.18	0.25	0.24	0.06	0.21	0.14	0.13	0.16
168912	1.04	0.32	**2.91**	0.47	0.45	1.29	1.60	0.63	0.44	0.41	0.96	0.42	0.26	0.90	0.33	0.31	0.62
190410	0.43	0.26	**0.69**	0.36	0.34	0.55	0.49	0.45	0.34	0.31	0.58	0.33	0.24	0.38	0.24	0.23	0.36
2074	0.69	0.28	**2.28**	0.41	0.41	1.48	0.90	0.58	0.37	0.36	0.82	0.37	0.17	0.82	0.29	0.29	0.45
28	0.78	0.33	**2.14**	0.52	0.50	1.17	0.74	0.71	0.46	0.44	0.96	0.45	0.06	0.81	0.33	0.35	0.44
32	0.71	0.31	**2.00**	0.53	0.51	1.28	0.72	1.01	0.43	0.41	0.86	0.41	0.06	0.79	0.32	0.34	0.46
3481	0.26	0.29	0.32	0.45	0.46	0.47	0.32	**0.61**	0.41	0.40	0.56	0.38	0.02	0.40	0.28	0.25	0.23
3510	0.60	0.29	**1.45**	0.51	0.48	1.16	0.60	1.11	0.42	0.40	0.78	0.40	0.05	0.71	0.30	0.33	0.39
3686	0.82	0.25	**1.26**	0.43	0.40	0.55	0.69	0.84	0.34	0.33	0.64	0.33	0.21	0.42	0.26	0.26	0.52
3711	0.87	0.26	**1.62**	0.44	0.42	0.55	0.76	0.99	0.35	0.34	0.68	0.34	0.25	0.48	0.26	0.27	0.48
3735	1.39	0.20	**2.56**	0.28	0.25	1.33	1.69	0.33	0.26	0.24	0.53	0.26	0.23	0.52	0.19	0.22	0.65
3889	0.79	0.31	**1.46**	0.49	0.47	0.45	0.98	0.79	0.40	0.38	0.68	0.39	0.18	0.50	0.30	0.25	0.56
3896	1.09	0.26	**3.85**	0.40	0.40	1.11	1.44	0.53	0.37	0.35	0.78	0.37	0.22	0.71	0.27	0.25	0.55
3897	0.52	0.17	**1.19**	0.27	0.25	0.42	0.46	0.50	0.24	0.22	0.51	0.24	0.13	0.29	0.17	0.17	0.33
3904	0.64	0.18	**1.51**	0.32	0.32	0.89	0.78	0.68	0.26	0.26	0.54	0.24	0.19	0.36	0.18	0.21	0.38
3950	0.61	0.32	**1.36**	0.50	0.48	0.71	0.79	0.68	0.44	0.41	0.73	0.41	0.18	0.66	0.30	0.23	0.52
3954	0.84	0.25	**1.43**	0.41	0.39	0.87	0.81	0.93	0.35	0.33	0.66	0.33	0.19	0.44	0.26	0.25	0.53
43	0.78	0.32	**3.16**	0.45	0.46	1.39	1.18	0.61	0.44	0.43	0.97	0.42	0.21	0.96	0.32	0.32	0.59
45	0.70	0.31	**4.00**	0.48	0.47	1.12	1.14	0.62	0.39	0.37	0.98	0.43	0.16	0.71	0.32	0.26	0.58
9952	1.21	0.26	**3.16**	0.42	0.38	1.88	1.28	0.62	0.37	0.34	0.79	0.34	0.20	0.81	0.28	0.25	0.55
9960	0.70	0.29	**3.10**	0.40	0.41	1.33	1.12	0.64	0.37	0.36	0.88	0.38	0.09	0.73	0.29	0.22	0.58
9985	0.19	0.12	**0.67**	0.17	0.18	0.60	0.28	0.24	0.16	0.15	0.28	0.16	0.08	0.20	0.11	0.07	0.17
9986	0.43	0.30	0.77	0.51	0.51	**0.85**	0.49	0.80	0.41	0.41	0.60	0.39	0.13	0.51	0.31	0.30	0.41
9987	0.43	0.30	0.80	0.51	0.50	**0.84**	0.50	0.79	0.41	0.41	0.60	0.38	0.13	0.51	0.31	0.29	0.42

D.4 NetScore-T (Log-Loss) Results

Table 14 presents the NetScore-T computed using the transformed log-loss metric. This version emphasizes probability calibration quality in the joint accuracy-energy assessment.

Table 14: NetScore-T (log-loss) per dataset and model. Higher values indicate better log-loss-energy trade-offs. Best values per row are shown in bold.

Dataset	CatBoost	DaNet	DecisionTree	IMLP	IMLP_C	kNN	LightGBM	LinearModel	MLP	MLP_C	RandomForest	ResNet	STG	SVM	TabNet	VIME	XGBoost
146206	2.97	0.95	**3.26**	1.41	1.38	0.67	2.77	2.76	1.30	1.22	2.05	0.50	0.58	1.59	0.93	0.71	0.96
146607	2.37	0.74	**3.39**	1.25	1.25	0.32	2.96	1.75	1.08	1.20	2.69	0.30	0.63	1.41	0.84	0.23	0.85
146820	17.59	6.15	8.74	4.09	1.67	4.56	**39.55**	9.32	4.93	4.46	13.28	3.33	0.70	38.22	6.33	0.50	1.06
14951	2.21	0.55	3.15	0.80	0.65	1.09	**3.47**	1.90	0.64	0.87	1.67	0.32	0.56	0.88	0.62	0.68	0.97
14952	7.73	2.21	**12.86**	3.56	3.15	1.92	8.95	6.89	3.43	1.98	4.21	0.92	0.59	5.00	2.00	1.46	1.04
14964	0.74	0.37	**1.60**	0.55	0.53	0.58	1.03	0.65	0.48	0.35	0.67	0.11	0.18	0.63	0.39	0.13	0.22
14969	0.55	0.26	**1.01**	0.35	0.36	0.15	0.59	0.62	0.37	0.36	0.71	0.11	0.26	0.45	0.26	0.08	0.33
14970	2.01	4.03	1.38	4.73	4.05	1.72	**8.73**	8.57	5.25	4.97	1.94	1.04	0.23	5.21	4.26	1.09	0.23
167141	7.07	1.40	5.40	1.65	1.42	0.86	**8.61**	2.37	1.52	1.50	4.01	1.07	0.65	3.98	1.23	0.19	1.04
167211	52.50	11.38	29.30	16.80	19.07	9.49	**93.29**	33.19	11.24	8.49	49.24	8.23	0.65	87.38	6.74	1.77	1.15
168908	0.74	0.60	0.39	0.71	0.82	0.28	0.77	0.25	0.81	0.81	**1.33**	0.34	0.63	0.70	0.54	0.37	0.59
168909	0.60	2.65	0.31	2.40	1.48	0.73	2.68	1.54	**4.15**	1.12	0.68	0.75	0.25	2.80	1.36	1.81	0.20
168910	0.32	0.23	**0.95**	0.42	0.38	0.08	0.42	0.26	0.38	0.33	0.58	0.13	0.22	0.35	0.24	0.16	0.25
168912	6.45	1.66	3.88	2.36	2.16	1.65	**7.32**	3.09	2.04	2.03	3.65	0.97	0.66	4.59	1.33	0.91	1.00
190410	1.12	0.64	0.68	0.72	0.63	0.39	1.26	1.00	0.82	0.75	**1.49**	0.33	0.64	0.96	0.58	0.33	0.74
2074	2.29	1.05	2.45	1.34	1.45	1.42	**3.21**	1.97	1.40	1.43	2.28	0.44	0.26	3.16	0.95	0.66	0.31
28	3.13	3.28	1.71	4.91	4.83	4.89	6.14	6.57	4.32	4.12	1.46	3.02	0.20	**7.36**	2.04	2.40	0.22
32	4.01	4.53	2.42	5.34	6.76	6.67	9.17	5.00	7.34	4.13	2.04	2.73	0.18	**14.14**	3.74	0.43	0.22
3481	0.46	0.98	0.32	1.63	1.73	0.36	1.31	**2.91**	2.00	1.87	0.55	0.86	0.13	1.52	0.79	0.78	0.10
3510	1.67	2.90	1.26	3.31	2.82	4.03	4.29	6.40	4.16	3.61	1.06	1.44	0.19	**6.99**	2.86	0.90	0.23
3686	**3.35**	0.97	3.25	1.45	1.20	0.54	2.75	2.61	1.37	1.28	2.31	0.40	0.59	1.61	0.91	0.55	0.93
3711	2.98	1.08	4.10	1.54	1.57	0.45	2.74	**4.43**	1.52	1.45	2.20	0.44	0.61	2.02	1.10	0.35	0.95
3735	4.07	0.55	4.32	0.77	0.71	1.25	**4.60**	0.99	0.75	0.70	1.51	0.74	0.68	1.50	0.53	0.62	1.87
3889	39.21	10.04	14.49	14.40	14.90	1.86	**41.25**	22.50	12.88	12.25	11.74	4.20	0.61	10.84	7.95	3.00	0.87
3896	4.23	0.93	**7.11**	1.26	1.38	1.20	5.39	1.96	1.40	1.32	2.85	0.73	0.66	2.54	0.87	0.47	1.07
3897	1.02	0.34	**1.79**	0.45	0.35	0.22	0.92	1.05	0.48	0.45	1.00	0.18	0.37	0.59	0.34	0.28	0.55
3904	2.54	0.73	**3.47**	1.11	1.12	0.70	3.03	2.70	0.98	0.97	2.17	0.21	0.62	1.42	0.75	0.22	0.97
3950	9.71	10.24	4.53	6.91	9.87	2.10	14.70	12.68	10.46	9.59	5.15	3.80	0.62	8.25	**108.61**	0.27	0.87
3954	3.02	0.94	**3.20**	1.43	1.26	0.69	2.76	2.75	1.31	1.22	2.10	0.44	0.56	1.59	0.93	0.71	0.96
43	4.87	1.54	6.64	1.92	2.21	1.41	**6.88**	2.51	2.23	2.19	3.92	0.86	0.66	3.92	1.37	1.40	0.99
45	3.09	1.05	6.22	1.83	1.94	1.51	**7.21**	3.64	2.29	1.97	2.01	1.05	0.44	4.47	0.86	1.11	0.59
9952	4.21	0.86	**5.05**	1.34	1.26	1.84	4.42	1.92	1.25	1.15	2.73	0.49	0.66	2.66	0.89	0.78	1.04
9960	11.96	0.94	9.43	0.99	0.84	1.15	**38.08**	1.42	1.23	1.18	3.49	0.37	0.32	2.31	1.14	0.14	0.44
9985	0.45	0.24	**1.00**	0.37	0.35	0.20	0.56	0.55	0.35	0.33	0.71	0.11	0.25	0.48	0.25	0.05	0.29
9986	1.68	2.85	1.03	3.62	3.43	3.19	**8.61**	6.22	5.73	4.18	1.39	0.95	0.23	4.02	2.69	1.75	0.26
9987	1.88	2.74	1.09	3.46	3.59	3.17	**9.47**	6.59	5.82	5.44	1.41	0.81	0.23	4.50	3.25	1.91	0.26

D.5 Wall-Time Results

Table 15 shows the total wall-clock time in seconds for training and inference across all segments. This includes the cumulative time for all segments in the stream.

Table 15: Total wall-time in seconds per dataset and model. Lower values indicate faster execution. Best values per row are shown in bold.

Dataset	CatBoost	DaNet	DecisionTree	IMLP	IMLP_C	kNN	LightGBM	LinearModel	MLP	MLP_C	RandomForest	ResNet	STG	SVM	TabNet	VIME	XGBoost
146206	2.59	703.01	**1.62**	19.78	24.13	2.86	6.45	4.95	76.65	100.78	5.42	110.80	146.22	92.48	575.51	162.62	4.95
146607	2.28	143.45	**0.78**	8.48	10.38	10.88	2.39	6.61	21.73	27.36	2.24	39.34	40.88	65.51	116.27	262.26	3.53
146820	0.59	82.83	**0.21**	7.32	7.11	0.39	0.34	3.64	12.60	15.72	1.67	26.70	19.85	1.00	65.63	23.52	1.92
14951	2.37	333.95	**1.09**	20.80	19.46	2.39	7.21	4.82	114.26	85.63	4.18	103.12	115.03	119.24	193.59	164.31	4.63
14952	2.67	419.12	**0.75**	13.29	15.96	6.38	2.97	5.77	49.72	66.77	4.13	86.22	91.79	20.10	319.09	159.27	4.20
14964	3.63	383.26	**0.67**	14.20	17.22	1.04	13.01	4.78	40.12	57.51	3.58	65.57	71.82	34.62	330.57	130.25	12.02
14969	4.82	303.67	**0.98**	24.88	11.37	2.57	12.15	4.69	27.64	28.97	2.82	39.22	47.86	44.26	156.25	88.01	5.79
14970	115.95	478.53	14.66	13.67	16.25	26.22	76.90	12.24	39.93	55.83	**9.88**	94.53	78.13	96.95	340.98	992.72	30.34
167141	0.62	53.54	**0.19**	5.29	6.74	0.52	0.31	3.09	9.04	12.47	1.06	18.51	13.43	2.49	47.22	17.82	1.33
167211	0.45	48.19	**0.18**	5.32	5.59	0.38	0.21	3.22	7.52	8.92	0.99	18.37	14.14	0.41	36.35	20.56	1.14
168908	15.81	38.24	4.96	6.79	6.32	9.99	20.49	6.19	8.52	10.35	**2.27**	14.34	15.34	100.93	42.93	291.28	8.79
168909	360.81	499.46	64.83	**16.87**	18.05	116.13	727.20	26.78	44.35	75.01	19.75	122.87	80.36	817.99	427.62	2575.18	102.45
168910	20.59	300.84	**1.42**	11.20	12.75	23.98	27.94	9.75	28.49	37.29	3.31	46.23	55.02	427.21	221.15	535.65	12.95
168912	0.64	59.79	**0.19**	4.98	6.13	0.51	0.55	3.05	8.32	10.82	1.07	14.52	13.86	2.56	45.12	29.78	1.46
190410	4.32	56.26	**2.42**	6.07	7.39	2.90	4.66	4.89	12.24	15.98	2.49	15.63	21.00	30.96	59.36	114.64	4.21
2074	1.81	148.06	**0.34**	8.20	9.40	0.67	2.13	4.31	18.98	22.17	1.68	30.00	25.96	3.65	84.44	81.43	4.25
28	1.84	109.78	**0.32**	6.67	7.91	1.14	3.59	4.29	13.68	17.09	1.61	21.78	22.77	5.00	86.89	61.11	6.27
32	5.43	397.28	**0.93**	14.36	16.85	1.96	8.09	5.27	46.65	62.63	4.22	90.57	91.33	11.15	325.68	132.85	14.18
3481	316.81	251.87	10.43	13.64	13.56	16.32	136.37	8.90	28.77	31.72	**6.68**	67.74	45.67	101.70	207.34	563.01	88.63
3510	6.80	468.37	**1.04**	12.54	15.15	2.05	11.36	4.69	42.32	53.51	3.84	77.03	75.37	14.09	351.76	118.37	13.29
3686	4.08	899.46	**2.87**	22.32	29.32	13.28	12.37	6.67	110.32	155.66	8.23	167.83	236.00	157.04	588.40	248.78	7.09
3711	2.93	648.96	**1.41**	20.14	23.83	10.77	7.87	6.22	85.73	113.10	5.45	134.20	164.77	80.51	546.84	222.93	6.25
3735	0.25	37.52	**0.15**	4.90	6.04	0.29	0.32	3.56	6.71	9.98	1.37	9.75	12.36	3.24	29.81	23.13	0.63
3889	3.18	291.36	**1.49**	13.28	15.73	17.96	2.22	6.06	38.34	49.30	3.41	63.98	88.50	47.53	253.07	381.67	4.42
3896	0.59	50.18	**0.16**	5.24	5.82	0.60	0.52	3.84	9.09	11.19	1.09	13.55	14.64	3.17	35.68	30.44	1.50
3897	2.54	307.66	**0.81**	14.62	18.88	3.98	12.14	4.67	34.84	53.35	3.06	54.37	66.68	63.06	291.69	94.46	4.84
3904	2.43	307.46	**0.93**	14.22	14.99	1.85	2.34	5.05	46.62	48.85	4.32	81.97	89.61	48.74	243.62	106.27	4.85
3950	3.61	113.70	**0.90**	7.12	8.64	2.99	3.08	5.10	14.58	17.93	1.94	39.66	26.42	10.64	129.96	164.12	2.80
3954	2.56	697.67	**1.59**	20.46	26.60	2.63	6.43	4.96	73.24	100.14	5.46	119.24	146.78	92.73	569.04	164.06	4.90
43	0.79	60.20	**0.18**	5.58	5.21	0.50	0.80	3.32	8.21	8.74	0.95	16.45	12.63	2.13	46.55	30.89	1.36
45	1.30	45.12	**0.14**	4.45	4.80	0.63	0.84	3.46	6.38	8.24	0.96	12.93	8.94	4.46	32.13	54.85	1.73
9952	0.62	96.64	**0.24**	6.33	7.28	0.41	0.85	3.64	13.46	20.34	1.61	34.88	22.10	3.85	85.31	45.91	1.88
9960	2.52	94.11	**0.35**	8.75	8.63	0.77	1.19	3.91	17.15	22.46	1.94	29.59	24.49	6.29	128.40	56.73	3.51
9985	5.72	197.68	**0.55**	9.37	10.01	0.88	6.66	4.79	17.63	23.91	2.13	31.80	29.95	19.43	108.38	59.57	6.55
9986	40.54	571.53	7.94	17.51	18.23	**5.31**	36.11	9.23	67.73	72.67	10.58	103.23	132.05	67.15	350.04	438.58	19.80
9987	40.79	536.19	7.87	17.50	19.18	**5.39**	33.21	9.43	72.04	75.18	10.45	113.54	132.29	66.96	402.38	615.34	19.81

D.6 Energy Consumption Results

Table 16 presents the total energy consumption in Joules, measured using the ElmorLabs PMD-USB power meter. Energy includes both training and inference phases across all segments.

Table 16: Total energy consumption in Joules per dataset and model. Lower values indicate more energy-efficient execution. Best values per row are shown in bold.

Dataset	CatBoost	DaNet	DecisionTree	IMLP	IMLP_C	kNN	LightGBM	LinearModel	MLP	MLP_C	RandomForest	ResNet	STG	SVM	TabNet	VIME	XGBoost
146206	168.65	57466.31	**57.58**	1844.11	2222.64	156.67	410.27	216.28	6099.69	7953.48	333.15	7081.80	11478.59	4172.42	46410.57	12962.03	563.34
146607	175.89	11864.94	**29.45**	787.84	967.01	856.55	149.92	301.03	1794.63	2231.11	96.94	2415.09	3385.22	3007.49	10898.85	22903.66	518.74
146820	30.50	6712.50	**5.92**	637.34	620.68	12.58	10.89	150.01	999.79	1244.32	56.53	1642.27	1553.11	39.88	5308.59	1784.68	191.30
14951	146.50	27963.84	**39.49**	1883.65	1806.82	124.76	470.38	208.28	8957.00	6809.22	202.40	6545.17	9178.04	5370.03	15637.02	13113.38	529.94
14952	172.40	35117.61	**25.38**	1213.39	1459.93	429.79	167.88	239.28	3941.15	5272.11	149.15	5406.17	7315.75	907.64	25816.55	12627.48	455.46
14964	273.93	31119.18	**20.64**	1295.35	1570.71	39.50	827.59	211.55	3196.72	4585.16	144.55	4092.71	5719.81	1552.08	27502.76	10359.30	1456.65
14969	378.54	24643.21	**36.56**	2296.65	1044.46	169.78	846.07	210.10	2238.67	2330.15	166.11	2430.49	3969.22	2037.46	12937.76	7118.13	747.65
14970	9505.89	38768.56	601.49	1267.96	1500.44	2051.08	5875.27	**530.24**	3269.37	4516.29	721.62	5918.92	6361.79	4337.26	31770.23	84976.73	5211.81
167141	32.24	4384.43	**5.08**	448.25	575.70	22.90	10.38	132.53	684.18	929.59	40.48	1144.78	1065.08	105.62	3811.89	1379.14	124.96
167211	24.16	4124.20	**4.98**	469.21	492.14	14.74	5.96	143.68	587.20	700.08	41.98	1130.34	1156.96	14.90	3007.68	1636.23	90.66
168908	1366.73	3062.28	207.41	612.34	589.13	761.62	1544.33	272.05	716.44	868.56	**135.73**	877.94	1283.05	4600.99	4044.19	23648.79	1520.37
168909	29710.28	40658.15	2682.12	1576.51	1703.39	9021.02	54664.12	**1144.24**	3845.92	6388.51	1471.26	8193.26	7058.47	36064.94	41887.40	210586.53	19837.78
168910	1610.11	24275.36	**52.52**	1025.49	1179.38	1895.12	1861.41	415.65	2365.79	3042.75	140.68	2854.36	4512.23	18935.69	20710.58	45075.81	1866.54
168912	39.28	5005.33	**6.07**	448.75	549.69	23.53	26.44	140.01	675.34	886.49	44.20	883.90	1113.93	114.60	3692.69	2335.01	146.29
190410	371.31	4980.80	**98.83**	534.51	670.29	206.25	347.32	218.51	972.19	1263.44	154.37	955.96	1774.42	1400.61	5617.62	10425.44	656.28
2074	136.54	12385.70	**10.97**	737.69	855.79	29.60	117.63	198.02	1531.46	1754.46	74.15	1834.85	2097.64	162.84	6892.12	6468.99	508.08
28	143.43	9053.95	**9.96**	591.65	692.79	67.29	234.85	194.37	1098.00	1357.35	69.92	1307.70	1824.15	226.93	7058.08	4834.53	770.20
32	420.80	33266.91	**33.27**	1293.74	1523.01	96.61	549.75	232.69	3702.34	4964.60	188.87	5756.46	7284.89	495.31	26291.31	10482.59	1744.62
3481	28116.43	20290.73	431.12	1238.55	1231.27	1248.12	10969.13	**400.99**	2366.58	2599.61	479.38	4099.99	3680.34	4682.30	19214.53	48307.61	15965.22
3510	515.27	39059.47	**37.82**	1128.01	1382.18	106.03	775.61	210.49	3371.72	4270.69	191.25	4894.22	6028.38	633.77	28493.14	9367.90	1647.03
3686	256.78	72905.05	**107.08**	2076.97	2708.62	913.49	821.22	278.43	8693.79	12279.74	510.96	10715.42	18569.21	7068.40	47522.10	19971.86	849.50
3711	170.59	53711.24	**51.95**	1871.59	2192.93	732.90	480.57	259.23	6795.11	8926.80	252.43	8517.83	12931.33	3655.64	44074.87	17720.34	738.84
3735	9.59	3036.76	**3.67**	441.98	544.90	8.94	9.97	161.72	536.32	802.70	51.44	565.35	977.65	142.62	2406.80	1745.34	47.19
3889	236.34	23659.21	**60.38**	1243.48	1465.83	1473.49	126.20	271.23	3110.55	3971.16	163.20	4023.12	7352.86	2189.25	20886.17	35517.62	623.35
3896	31.90	4347.11	**3.94**	468.89	513.25	28.00	22.05	170.48	719.99	888.10	44.37	823.16	1198.86	140.63	2968.18	2432.64	142.19
3897	177.67	26099.91	**29.33**	1319.17	1693.87	267.15	806.67	207.85	2783.08	4247.43	148.52	3372.73	5400.15	2887.35	23721.36	7527.52	581.75
3904	153.93	25858.58	**32.55**	1291.60	1358.47	82.87	130.00	207.66	3608.89	3798.12	191.20	5106.24	7149.59	2195.91	19709.79	8397.95	554.80
3950	293.96	9888.32	**34.93**	643.42	781.34	205.33	216.04	224.28	1165.59	1433.55	97.18	2438.71	2211.68	487.65	12229.10	15133.58	379.48
3954	162.51	56959.90	**60.25**	1889.83	2404.32	140.30	407.05	215.81	5708.14	7645.16	293.78	7645.39	11529.87	4182.49	45933.68	13065.82	546.89
43	70.81	4936.87	**4.87**	510.21	467.83	22.06	42.35	155.17	663.91	711.80	39.51	1013.09	998.47	90.48	3764.09	2427.54	145.94
45	98.25	3999.65	**3.79**	387.26	415.87	33.51	45.93	153.22	507.50	648.96	36.84	776.40	755.20	198.43	2705.09	5080.65	183.55
9952	33.42	7848.87	**6.66**	564.17	647.80	13.42	43.34	161.31	1079.33	1616.52	68.85	2166.24	1753.41	169.36	6897.22	3548.31	197.11
9960	210.08	8047.92	**11.06**	767.40	766.09	35.14	72.11	175.15	1381.41	1804.02	85.09	1807.05	2016.00	283.13	10590.11	4542.36	411.06
9985	453.40	16290.41	**19.46**	824.07	913.65	41.95	454.27	217.91	1421.33	1914.75	105.71	1917.83	2392.95	896.31	8829.51	4712.59	851.17
9986	3282.38	47546.91	329.54	1604.45	1670.81	**315.69**	2671.95	400.16	5357.20	5767.98	764.75	6400.07	10525.98	3132.93	28314.11	34671.69	3174.28
9987	3311.77	44767.88	**326.79**	1599.41	1740.46	328.06	2545.87	399.74	5750.72	5977.37	770.34	7037.03	10553.54	3127.13	32553.05	57135.43	3179.81

E Statistical Tests

We conducted comprehensive statistical analysis following the methodology of Demšar [9] to compare model performance across multiple datasets. This section presents detailed results of the Friedman omnibus tests and post-hoc Wilcoxon signed-rank tests with Holm correction.

E.1 Friedman Omnibus Test Results

All statistical tests were conducted with $N = 36$ datasets and $k = 17$ classifiers at significance level $\alpha = 0.05$. The critical difference for post-hoc comparisons is $CD = 4.11$.

Metric	**Friedman χ^2**	**p-value**	**Null Hypothesis**
Balanced Accuracy	226.41	3.45×10^{-39}	Rejected
Log-Loss	384.94	5.19×10^{-72}	Rejected
NetScore-T (Balanced)	478.05	1.42×10^{-91}	Rejected
NetScore-T (Log-Loss)	363.49	1.59×10^{-67}	Rejected
Total Energy (Joules)	490.98	2.66×10^{-94}	Rejected
Total Time (Seconds)	484.27	6.93×10^{-93}	Rejected

Table 17: Friedman omnibus test results across all metrics. All tests decisively reject the null hypothesis of equal performance, warranting post-hoc pairwise analysis.

E.2 Model Performance Statistics

	Bal. Acc.		**Log-Loss**		**NS-T (Bal.)**		**NS-T (Log.)**		**Energy (J)**		**Time (s)**	
Model	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
LightGBM [T]	0.825	0.167	0.283	0.266	0.872	0.555	9.996	17.631	2466	9179	33.1	121.7
MLP [N]	0.823	0.163	0.336	0.330	0.348	0.077	2.991	3.196	2825	2316	35.4	29.4
MLP_C [N]	0.814	0.175	0.355	0.350	0.331	0.077	2.536	2.746	3511	2809	44.1	35.6
DANet [N]	0.809	0.171	0.343	0.328	0.258	0.056	2.294	2.860	23447	18962	284.3	231.8
SVM [B]	0.808	0.171	0.345	0.330	0.596	0.357	6.668	15.312	3325	6564	74.1	148.7
CatBoost [T]	0.806	0.176	0.389	0.354	0.685	0.310	5.967	10.555	2286	6768	27.4	79.5
IMLP [N]	0.804	0.170	0.399	0.348	0.404	0.094	2.810	3.537	1079	551	11.8	5.9
ResNet [N]	0.803	0.169	1.533	1.460	0.342	0.072	1.187	1.609	3716	2729	59.2	42.5
TabNet [N]	0.794	0.185	0.370	0.337	0.259	0.061	4.735	17.907	18336	14405	218.8	174.5
IMLP_C [N]	0.789	0.176	0.475	0.416	0.394	0.090	2.850	4.005	1192	615	13.1	6.6
k-NN [B]	0.781	0.156	1.451	1.507	0.891	0.416	1.760	2.006	610	1539	8.3	19.7
XGBoost [T]	0.764	0.179	1.207	0.678	0.447	0.147	0.684	0.404	1866	4109	11.6	21.7
LinearModel [B]	0.758	0.202	0.479	0.502	0.636	0.217	4.765	6.478	262	176	6.0	4.1
VIME [N]	0.739	0.201	1.117	1.553	0.241	0.062	0.811	0.707	21501	37218	257.3	449.4
RandomForest [B]	0.738	0.181	0.559	0.399	0.672	0.195	3.952	8.213	237	293	4.1	3.8
DecisionTree [B]	0.717	0.190	0.690	0.450	1.784	1.106	4.477	5.429	152	454	3.8	11.0
STG [N]	0.426	0.165	1.153	0.691	0.163	0.072	0.458	0.201	5115	4286	63.7	54.4

Table 18: Complete performance statistics across 36 TabZilla datasets. Models are categorized as Tree-based [T], Neural [N], or Baseline [B]. IMLP achieves competitive accuracy while maintaining superior energy and time efficiency compared to other neural methods.

E.3 Efficiency Analysis

The efficiency metrics reveal stark differences between model categories:

Traditional ML methods dominate efficiency rankings, with DecisionTree (152J), RandomForest (237J), and LinearModel (262J) consuming minimal energy. Among neural networks, IMLP (1079J) achieves 2.6× lower energy consumption than standard MLP (2825J) and 17× lower than DANet (23,447J).

Similar patterns emerge for execution time, where IMLP (11.8s) executes 3× faster than MLP (35.4s) and 24× faster than DANet (284.3s), while maintaining competitive accuracy.

All pairwise comparisons show statistical significance after Holm correction ($CD = 4.11$), confirming that observed efficiency gains are not due to random variation across the 36 datasets.

The Friedman test statistics are particularly large for efficiency metrics ($\chi^2 > 484$), indicating substantial and consistent differences in computational requirements across methods, with IMLP achieving the best accuracy-efficiency trade-off among neural approaches.

E.4 Error Analysis and Dataset-Specific Performance

To understand when and why IMLP provides advantages over baseline methods, we conducted pairwise comparisons across all 36 TabZilla datasets. This analysis reveals distinct performance patterns that illuminate IMLP's positioning in the accuracy-efficiency landscape.

E.4.1 Predictive Performance Analysis

Metric	vs. MLP		vs. LightGBM		vs. CatBoost		vs. XGBoost	
	IMLP Better	Base Better	IMLP Better	Base Better	IMLP Better	Base Better	IMLP Better	Base Better
Balanced Accuracy	5	31	11	25	14	22	**27**	9
Log-Loss	2	34	4	32	11	25	**34**	2

Table 19: Dataset count where IMLP outperforms key baselines on predictive metrics. IMLP consistently dominates XGBoost while trailing other methods.

The predictive performance analysis reveals a clear hierarchy: IMLP consistently outperforms XGBoost (winning on 27/36 datasets for balanced accuracy and 34/36 for log-loss) but generally trails MLP, LightGBM, and CatBoost. The mean differences are modest: IMLP achieves 1.97% lower balanced accuracy than MLP but 3.93% higher than XGBoost, indicating competitive performance within the neural network family.

E.4.2 Efficiency-Adjusted Performance

Metric	vs. MLP		vs. LightGBM		vs. CatBoost		vs. XGBoost	
	IMLP Better	Base Better	IMLP Better	Base Better	IMLP Better	Base Better	IMLP Better	Base Better
NetScore-T (bal. acc.)	**34**	2	7	29	7	29	13	23
NetScore-T (log-loss)	18	18	2	34	9	27	**32**	4
Total Time (s)	**36**	0	7	29	7	29	8	28
Total Energy (J)	**35**	1	7	29	7	29	13	23

Table 20: Dataset count where IMLP outperforms baselines on efficiency and composite metrics. IMLP dominates other neural methods but trails tree-based approaches.

When efficiency is considered, IMLP's value proposition becomes evident. Against standard MLP, IMLP wins decisively: faster on all 36 datasets (mean speedup: 23.5s) and more energy-efficient on 35/36 datasets (mean reduction: 1,746J). The NetScore-T (Balanced) metric particularly favors IMLP over MLP (34 vs. 2 datasets), demonstrating superior accuracy-efficiency trade-offs.

However, tree-based methods maintain their efficiency advantage, with LightGBM and CatBoost outperforming IMLP on efficiency metrics across 29/36 datasets. This reflects the fundamental computational efficiency of tree-based architectures compared to neural networks.

E.4.3 Landscape Analysis

The pairwise analysis reveals three distinct performance tiers:

1. **Accuracy Leaders**: LightGBM, MLP, and CatBoost dominate predictive metrics, with LightGBM achieving the best overall balance.
2. **Efficiency-Accuracy Optimizers**: IMLP occupies a unique position, offering neural network expressiveness with substantially improved efficiency compared to standard MLPs, while maintaining competitive accuracy.

3. **Pure Efficiency Champions**: Tree-based methods (particularly DecisionTree and k-NN) excel in computational efficiency but may sacrifice some accuracy on complex datasets.

E.4.4 Practical Implications

Deployment scenarios requiring neural network capabilities with energy constraints, streaming data applications where constant-time updates matter, and situations where the modest accuracy trade-off (<2% vs. MLP) is acceptable for significant efficiency gains (3× speedup, 60% energy reduction).

When maximum predictive accuracy is paramount (favor LightGBM/MLP), when computational resources are unconstrained (favor standard MLP), or when extreme efficiency is required regardless of accuracy (favor DecisionTree/k-NN).

The consistent pattern across efficiency metrics confirms IMLP's design goal: providing a practical middle ground between the accuracy of full neural networks and the efficiency demands of production deployment.

E.5 Per-Segment Analysis and Learning Dynamics

We examine the fundamental paradigmatic difference between IMLP's incremental learning approach and traditional batch retraining methods. IMLP operates exclusively in segmental mode (training only on new data), while baseline neural methods operate in cumulative mode (retraining on all accumulated data). This distinction drives fundamentally different computational and deployment characteristics.

E.5.1 Learning Paradigm Comparison

The segment data demonstrates two distinct learning paradigms with different computational and data requirements:

IMLP (Segmental Mode): Trains exclusively on each new data segment using attention-based feature replay to maintain knowledge of previous patterns. By segment N, IMLP has seen only the data from segment N.

MLP (Cumulative Mode): Retrains from scratch on the complete accumulated dataset at each segment. By segment N, MLP has retrained on data from segments 0 through N combined.

This fundamental difference means accuracy comparisons across segments are not directly equivalent, MLP leverages exponentially more training data as segments progress.

E.5.2 Energy Efficiency Analysis

The computational efficiency comparison is valid and reveals substantial advantages for incremental learning:

Segment	IMLP Energy (J)	MLP Energy (J)	MLP Overhead
0	128.1	131.9	1.0×
1	81.7	107.9	1.3×
2	88.8	121.6	1.4×
3	101.3	137.8	1.4×
4	84.0	154.8	1.8×
5	81.0	167.8	2.1×
6	82.8	187.9	2.3×
7	81.3	226.2	2.8×

Table 21: Per-segment energy consumption. IMLP maintains constant computational cost ($\sim$ 85J after initialization) while MLP's batch retraining shows linear growth with accumulated data size.

After initialization, IMLP stabilizes at approximately 85J per segment, confirming theoretical constant-time updates regardless of historical data size. This enables predictable computational requirements for long-term deployment.

MLP exhibits 71% energy growth from segment 0 to 7 (132J $\rightarrow$ 226J), reflecting the linear scaling inherent in batch retraining as dataset size grows. This trend projects to 350J+ per segment by segment 20, making long-term deployment computationally prohibitive.

E.5.3 Data Efficiency and Continual Learning Effectiveness

The most striking finding emerges from analyzing performance relative to training data consumption:

Segment	IMLP Accuracy (Segmental)	MLP Accuracy (Cumulative)	Training Data Ratio (MLP:IMLP)
0	0.747	0.647	1:1
1	0.766	0.740	2:1
2	0.776	0.769	3:1
3	0.776	0.781	4:1
4	0.789	0.792	5:1
5	0.774	0.795	6:1
6	0.783	0.809	7:1
7	0.796	0.815	8:1

Table 22: Performance vs training data consumption. IMLP achieves 79.6% accuracy using 1/8th the training data required by MLP to reach 81.5%.

By segment 7, IMLP achieves 79.6% accuracy having trained only on segment 7's data, while MLP requires all eight segments of accumulated data to reach 81.5%. This represents achieving 97.7% of MLP's performance with 12.5% of the training data—a compelling demonstration of effective continual learning.

The only fair accuracy comparison occurs at segment 0, where both methods train on identical data. IMLP achieves 74.7% versus MLP's 64.7%, a 15.5% relative improvement, indicating superior learning efficiency when given equivalent training data. IMLP's ability to maintain 77-79% accuracy across segments 1-7 while training only on individual segments demonstrates successful mitigation of catastrophic forgetting. The attention-based feature replay mechanism effectively preserves relevant knowledge without requiring raw data storage.

E.5.4 Cumulative Computational Cost Analysis

Long-term deployment scenarios reveal the compounding advantages of incremental learning:

Segment	IMLP Cumulative (J)	MLP Cumulative (J)	Efficiency Advantage
0	128.1	131.9	1.0×
2	298.6	361.4	1.2×
4	484.0	654.0	1.4×
6	647.7	1009.7	1.6×
7	729.0	1235.8	1.7×

Table 23: Cumulative energy consumption showing widening efficiency gap. The advantage grows from parity to 1.7× by segment 7, with the trend indicating continued divergence.

The cumulative energy gap widens from parity at segment 0 to 1.7× by segment 7. Extrapolating this trend suggests 2.5× advantage by segment 15 and 4× by segment 30, making incremental learning essential for long-term deployment feasibility.

By segment 7, IMLP has consumed 507J less energy than MLP (729J vs 1,236J), representing a 41% reduction in total computational cost. In large-scale deployments, these savings translate directly to reduced operational expenses and carbon footprint.

E.5.5 Practical Deployment Implications

IMLP Advantages:

- **Resource-Constrained Environments**: Constant 85J per update enables deployment on edge devices and mobile platforms where batch retraining would exceed power budgets.
- **Privacy-Preserving Applications**: Segmental learning eliminates the need to store historical raw data, addressing data retention regulations and privacy concerns.
- **Real-Time Systems**: Predictable computational requirements enable consistent response times regardless of historical data volume.
- **Long-Term Learning**: Growing efficiency advantage makes IMLP the only viable option for systems intended to learn continuously over months or years.

MLP Advantages:

- **Maximum Accuracy Scenarios**: When computational resources are unlimited and maximum predictive performance is paramount, batch retraining on complete datasets provides marginal accuracy improvements.
- **Short-Term Deployment**: For applications processing fewer than 10 segments, the computational overhead remains manageable.

F Reproducibility Assets and Instructions

We provide comprehensive instructions for reproducing all experimental results, with particular emphasis on the hyperparameter optimization procedure that underpins our comparative evaluation.

F.1 Shared Hyperparameter Optimization

Both MLP and IMLP models utilize identical optimized hyperparameters obtained through the comprehensive search described in Section F.3. This design choice ensures fair comparison by providing both architectures with equivalent optimization budget and regularization strategies. The attention-specific hyperparameters for IMLP (`window_size`, `use_attention`) are set to their default values as specified in the configuration files, focusing the optimization on general neural network training techniques that benefit both architectures.

F.1.1 Preprocessing Pipeline

Execute the data preparation:

```
cd data
python openml_data_processor.py --task_list openml_import.txt \
    --num_workers 4 --min_segment_size 500 --max_segment_size 1000
```

This generates both segmented datasets (for IMLP) and cumulative datasets (for baseline models) with consistent train/validation/test splits across all 36 tasks.

F.2 External Dependencies and Platform Compatibility

F.2.1 Core Dependencies

The framework integrates with TabZilla [34] for baseline model implementations:

```
# Install core dependencies
pip install -r requirements.txt
```

F.2.2 Model-Specific Requirements

Several baseline models have additional dependencies:

- **Tree-based models**: LightGBM, XGBoost, CatBoost with platform-specific optimizations
- **Transformer models**: Additional memory requirements for attention mechanisms
- **Specialized architectures**: DANet, NODE, SAINT with custom CUDA kernels

F.2.3 Platform Considerations

The codebase supports both CPU and CUDA execution with automatic device detection. Mixed-precision training (AMP) is enabled by default on compatible hardware but can be disabled for older GPUs.

F.3 Hyperparameter Optimization Framework

Following the methodology of Kadra et al. [23], we employ a comprehensive hyperparameter search for both MLP and IMLP models to ensure fair comparison. Our approach extends beyond simple grid search to include a "regularization cocktail" that systematically explores combinations of modern deep learning techniques.

F.3.1 Search Space Definition

The optimization space encompasses multiple regularization families:

Implicit Regularization:

- Batch Normalization: `use_batch_norm` $\in \{\text{True}, \text{False}\}$
- Stochastic Weight Averaging: `use_swa` $\in \{\text{True}, \text{False}\}$

Explicit Regularization:

- Weight Decay: `use_weight_decay` $\in \{\text{True}, \text{False}\}$
- Weight Decay Coefficient: `weight_decay` $\in [10^{-5}, 10^{-1}]$ (log-uniform)
- Dropout: `use_dropout` $\in \{\text{True}, \text{False}\}$
- Dropout Patterns: `dropout_shape` $\in \{\text{funnel}, \text{long\_funnel}, \text{diamond}, \text{triangle}\}$
- Dropout Rate: `dropout_rate` $\in [0.0, 0.8]$ (uniform)

Architectural Variations:

- Skip Connections: `use_skip` $\in \{\text{True}, \text{False}\}$
- Skip Types: `skip_type` $\in \{\text{Standard}, \text{ShakeShake}, \text{ShakeDrop}\}$
- ShakeDrop Probability: `shakedrop_prob` $\in [0.0, 1.0]$ (uniform)

Training Techniques:

- Data Augmentation: `augmentation` $\in \{\text{None}, \text{MixUp}\}$
- Augmentation Magnitude: `aug_magnitude` $\in [0.0, 1.0]$ (uniform)
- Mixed Precision: `use_amp` $\in \{\text{True}, \text{False}\}$
- Gradient Clipping: `max_grad_norm` $\in [0.1, 10.0]$ (log-uniform)

F.3.2 Optimization Algorithm

We employ Optuna [1] with the following configuration:

- **Sampler**: Tree-structured Parzen Estimator (TPE) with multivariate optimization
- **Pruner**: MedianPruner with 50 startup trials and 50 warmup steps

- **Trials per Task**: 100 trials with early stopping (patience=100)
- **Training Budget**: 100 epochs per trial with early stopping (patience=10)
- **Objective**: Minimize $1 -$ validation balanced accuracy

F.3.3 Computational Requirements

The hyperparameter optimization requires substantial computational resources:

- **Total Runtime**: Approximately 72 hours for all 36 tasks
- **Trials per Task**: 100 trials × 36 tasks = 3,600 total optimization runs
- **Storage**: SQLite databases for persistence and resumption

F.3.4 Execution Protocol

The optimization is ran through a parallelized bash script:

```
#!/bin/bash
N_TRIALS=100
EPOCHS=100
DEVICE="cuda"
MAX_PARALLEL=22
DATA_ROOT="../data/full_datasets"

# Parallel execution across all tasks
printf "%s\n" "${TASK_IDS[@]}" | xargs -I {} -P ${MAX_PARALLEL} \
    bash -c 'python mlp_c.py --task_id {} --n_trials ${N_TRIALS} \
             --epochs ${EPOCHS} --device ${DEVICE} \
             --storage "sqlite:///optuna_db/task_{}.db" \
             --data_root ${DATA_ROOT}'
```

Each task generates optimized hyperparameters saved as YAML files: `tuning/task_{TASK_ID}_hyperparams.yml`

F.3.5 Integration with Main Experiments

The CLI automatically loads tuned hyperparameters when available:

```
tuning_f = f"tuning/task_{args.task}_hyperparams.yml"
if not args.no_tuning and os.path.isfile(tuning_f):
    merge_dict(hp, load_yaml(tuning_f))
```

This ensures that all comparative results use optimized configurations, providing a fair evaluation baseline that reflects the current state-of-the-art in hyperparameter optimization for tabular neural networks.

F.3.6 Reproducibility Considerations

To ensure reproducible optimization:

- Fixed random seed (42) across all Optuna samplers
- Deterministic trial ordering through study persistence
- Gradient clipping and mixed precision for numerical stability
- Model checksum verification for state consistency

F.4 Hardware Requirements and Energy Measurement

F.4.1 Hardware Setup

All experiments were conducted on a single workstation with the following hardware configuration:

Compute Platform:

- CPU: Intel Core i5-8600K @ 3.60GHz (6 physical cores, 6 logical cores)
- GPU: NVIDIA GeForce RTX 2080 Ti Rev. A (CUDA Compute Capability 7.5)
- Memory: 15GB RAM
- Architecture: x86_64

Software Environment:

- Operating System: Debian GNU/Linux 12 (bookworm)
- Kernel Version: 6.1.0-32-amd64
- Compiler: GCC 12.2.0 (Debian 12.2.0-14)
- CUDA Toolkit: 11.8.89

Limitations:

- Memory constraints may limit batch sizes for larger models
- Single-GPU configuration restricts parallel training capabilities
- Total system memory (15GB) may constrain certain memory-intensive operations

All timing measurements and energy consumption data reported in this work are specific to this hardware configuration. Performance scaling to different hardware configurations should be considered when reproducing results, particularly for:

- Different GPU architectures (compute capability variations)
- Systems with varying memory capacities
- Multi-GPU configurations

The reported absolute performance metrics should be interpreted relative to this baseline configuration, with relative performance improvements being the primary focus for cross-system validation.

F.4.2 Energy Measurement Setup

Hardware-based Measurement (Recommended): We employ an ElmorLabs PMD-USB power measurement device with PCIe slot adapter for precise wall-power readings at 500-800Hz sampling rate. This setup provides ground-truth energy measurements by capturing total system power draw during training and inference phases.

Software-based Measurement (Alternative): For systems without dedicated power measurement hardware, the framework can fall back to software-based energy estimation using NVIDIA's Management Library (nvidia-smi) or Intel's RAPL interface. However, as noted by Yang et al. [52], these software solutions suffer from significant limitations:

- **Sampling Coverage**: NVIDIA's power sensor samples only 25% of runtime on A100/H100 cards
- **Estimation Error**: Up to 65% under/over-estimation compared to calibrated external meters
- **Temporal Resolution**: Lower sampling rates lead to missed power spikes during intensive operations

The energy monitoring can be disabled entirely by setting appropriate flags, though this removes the energy-efficiency evaluation component of our NetScore-T metrics.

G Additional Figures

We present supplementary visualizations analyzing the first 7 segments (segments 0-6) of continual learning performance. A fundamental distinction in our evaluation is the training paradigm: IMLP operates in **segmental mode** (training only on current segment data), while all baseline models operate in **cumulative mode** (retraining on all accumulated data up to the current segment).

G.1 Learning Paradigm Comparison

Figure 3 contrasts the fundamental learning approaches across the first 7 segments. The visualization highlights a critical comparison: IMLP's segmental learning (solid red line) versus baselines' cumulative retraining (dashed lines).

IMLP maintains stable performance around 0.77-0.78 balanced accuracy using only current segment data, demonstrating effective catastrophic forgetting mitigation through attention-based feature replay. The performance remains remarkably consistent despite never seeing historical raw data.

Cumulative approaches (dashed lines) show improving performance as they accumulate more training data, with MLP and LightGBM reaching higher final accuracy by leveraging all historical information. However, this comes at the cost of dramatically increasing computational requirements.

The cumulative energy plot (bottom-left) reveals the fundamental trade-off: IMLP's constant per-segment energy consumption versus the linear growth exhibited by cumulative approaches. By segment 6, this difference becomes substantial.

IMLP achieves approximately 92% of the performance that MLP attains when MLP has access to 7× more training data (all segments 0-6 vs. just segment 6), while consuming significantly less cumulative energy.

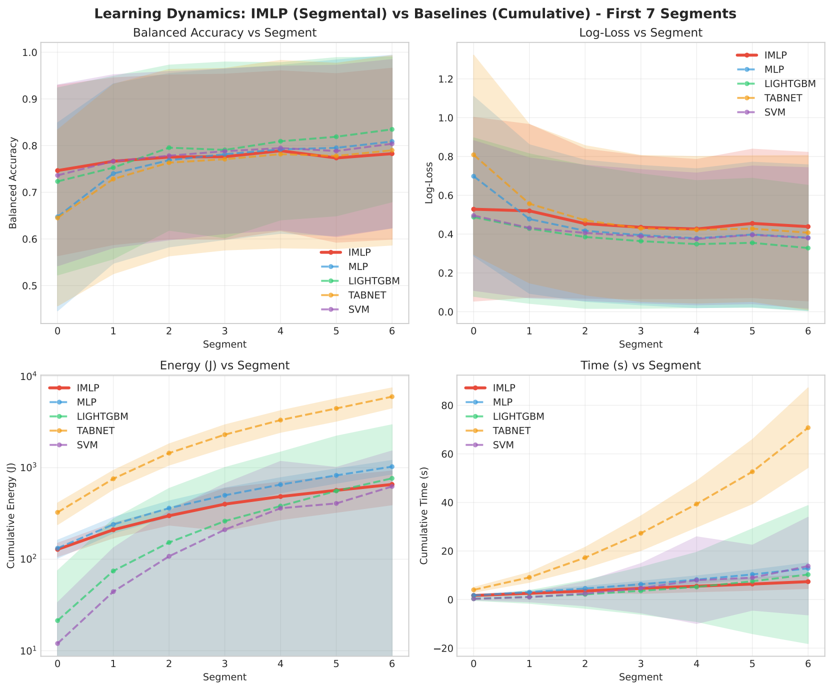


Figure 3: Learning paradigm comparison across the first 7 segments. IMLP (solid red) operates in segmental mode using only current data, while baselines (dashed lines) use cumulative retraining. The dramatic energy divergence in the bottom-left panel demonstrates IMLP's sustainable learning approach. Shaded areas represent ±1 standard deviation across 36 datasets.

G.2 IMLP Segmental Learning Characteristics

Figure 4 provides comprehensive analysis of IMLP's segmental learning behavior, validating the consistency and efficiency of the attention-based continual learning approach.

The top-left panel shows IMLP's balanced accuracy across segments, with an overall mean of 0.773. While there is some variance (particularly visible in segment 1), the model demonstrates reasonable stability given that it trains only on current data without access to historical samples.

After initialization (segment 0), IMLP maintains approximately constant energy consumption with a post-initialization mean of 86.6J per segment (top-right panel). This validates the theoretical $O(W \cdot d^2)$ complexity prediction and enables predictable resource planning for deployment.

The bottom-left histogram shows IMLP's balanced accuracy distribution across 36 datasets at segment 6, with a mean of 0.783. Most datasets achieve performance between 0.7-0.9, demonstrating robust generalization across diverse tabular domains with only a few challenging datasets showing lower performance.

The bottom-right scatter plot reveals that total energy consumption (cumulative across all segments) shows reasonable correlation with final accuracy, indicating that more complex datasets require more computational resources, as expected.

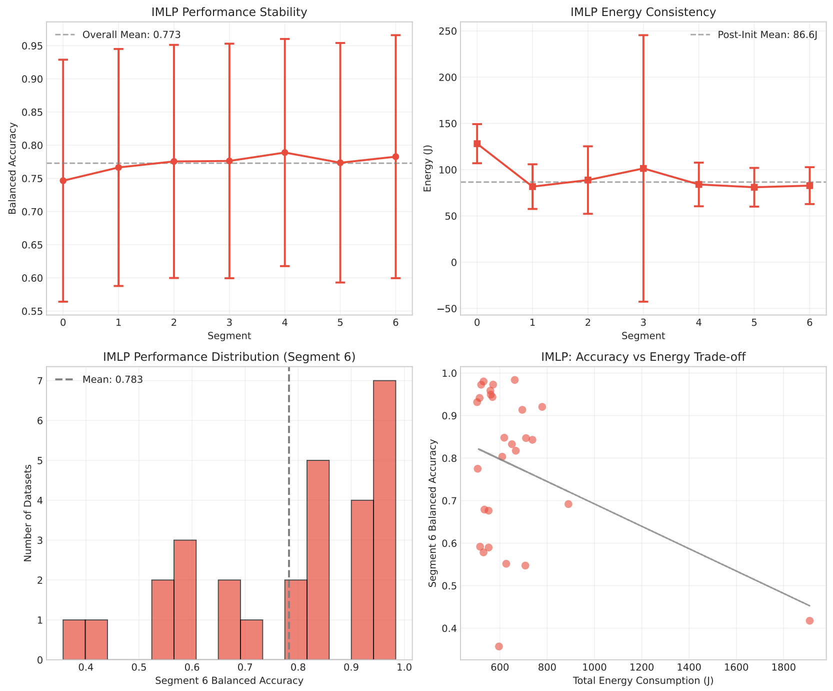


Figure 4: Comprehensive analysis of IMLP's segmental learning characteristics over 7 segments. Top-left: Performance stability with overall mean 0.773. Top-right: Energy consistency after initialization (∼86.6J per segment). Bottom-left: Performance distribution at segment 6 across 36 datasets. Bottom-right: Energy-accuracy relationship showing predictable resource scaling.

G.3 Accuracy-Energy Pareto Frontier

Figure 5 visualizes the trade-offs between predictive performance and computational efficiency using segment 6 results across all evaluated models. This analysis reveals IMLP's positioning in the accuracy-energy landscape after 7 segments of learning.

The plot clearly separates three model families: neural networks (red circles) cluster in the moderate-to-high energy, variable accuracy region; tree-based methods (green squares) achieve high accuracy with moderate energy consumption; classical methods (blue triangles) minimize energy usage with acceptable accuracy.

Among neural approaches, IMLP achieves a compelling balance, positioned distinctly from other neural methods by consuming significantly less energy while maintaining competitive accuracy. The explicit labeling shows IMLP's advantage over standard MLP approaches.

The logarithmic x-axis emphasizes the orders-of-magnitude differences in computational requirements, with IMLP consuming approximately 100J while maintaining neural network expressiveness, compared to 1000J+ for other neural approaches.

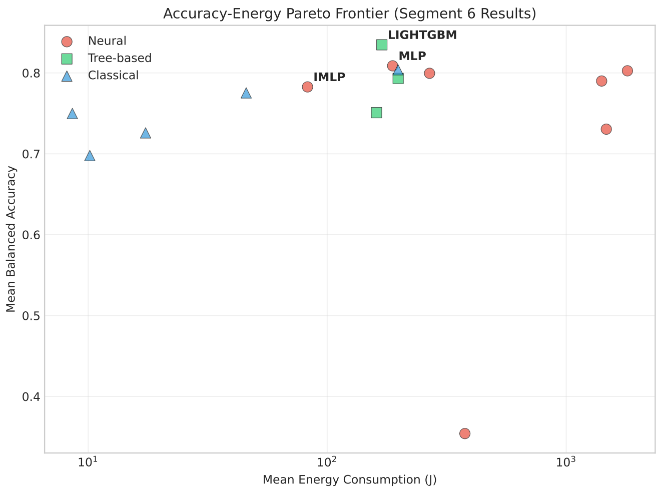


Figure 5: Accuracy-energy Pareto frontier using segment 6 performance across all models. IMLP achieves optimal positioning among neural approaches, delivering competitive accuracy with substantially reduced energy consumption. The logarithmic x-axis emphasizes the dramatic differences in computational requirements between model families.

G.4 Cumulative Energy Efficiency Analysis

Figure 6 demonstrates the compound advantages of IMLP's segmental learning approach over the first 7 segments, emphasizing computational sustainability for extended deployment.

The left panel shows cumulative energy consumption where IMLP (solid red) maintains shallow linear growth, while cumulative approaches (dashed lines) exhibit much steeper growth. By segment 6, the gap has become substantial, with IMLP consuming approximately 500J total versus 2000J+ for MLP.

The right panel quantifies efficiency ratios relative to MLP baseline, revealing IMLP's advantage growing from 1.0× at segment 0 to an impressive 6× by segment 6. This dramatic improvement demonstrates the compound benefits of constant-time updates versus quadratic growth in cumulative approaches.

Interestingly, LightGBM also shows efficiency gains versus MLP (reaching $\sim 3\times$ by segment 6), but still requires significantly more energy than IMLP's segmental approach.

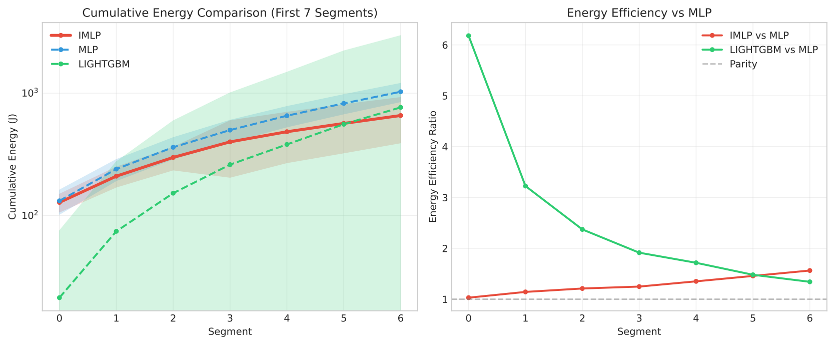


Figure 6: Cumulative energy efficiency analysis over the first 7 segments. Left: IMLP's segmental learning (solid line) versus cumulative retraining approaches (dashed lines) showing dramatically diverging energy trajectories. Right: Efficiency ratios demonstrating IMLP's growing advantage, reaching $6\times$ better efficiency than MLP by segment 6.

G.5 Empirical Validation of Theoretical Predictions

The 7-segment analysis validates key theoretical claims about incremental learning:

IMLP's $O(W \cdot d^2)$ attention complexity translates to consistent $\sim$86.6J per segment after initialization, confirming theoretical constant-time update guarantees regardless of historical data size.

Baseline models exhibit the expected $O(T \cdot N)$ growth patterns, with energy consumption growing linearly as accumulated dataset size expands across segments.

IMLP's stable performance ($\pm$0.02 around 0.773 mean) while training only on current segments demonstrates successful catastrophic forgetting mitigation through attention-based feature replay.

G.6 Practical Deployment Implications

The 7-segment analysis reveals IMLP's value proposition for practical applications:

The consistent $\sim$86.6J per segment enables accurate resource planning and deployment on energy-constrained devices where cumulative retraining would quickly exceed power budgets.

The $6\times$ efficiency advantage by segment 6, with continued divergence, makes IMLP the only viable approach for systems requiring continuous adaptation over extended periods.

Segmental learning eliminates raw data storage requirements while achieving 92% of cumulative baseline performance, addressing data retention regulations without sacrificing functionality.

Constant computational requirements enable consistent response times and predictable latency regardless of system uptime or data volume processed.

G.7 Performance Contextualization

While IMLP demonstrates compelling efficiency advantages, the analysis also contextualizes its performance:

The $\sim$0.07 balanced accuracy difference compared to cumulative MLP at segment 6 represents a modest but measurable trade-off. For many applications, this 8% relative performance reduction is acceptable given the dramatic efficiency gains.

The performance distribution shows most datasets achieving strong results (0.7-0.9 range), with a few challenging cases where segmental learning may be insufficient for critical applications.

H Licensing and Privacy Notes

All datasets used in this study are sourced from the TabZilla benchmark [34], comprising publicly available classification tasks from OpenML (`https://openml.org`) distributed under permissive licenses (CC0, CC-BY) compatible with academic research. IMLP's architecture inherently addresses privacy concerns through feature-level replay that stores only 256-dimensional latent representations rather than raw input data, with automatic expiration via the sliding window mechanism ($W = 10$ segments by default, configurable for stricter retention policies such as 30-day deletion). This design naturally supports data protection regulations like GDPR through data minimization and eliminates the need for cumulative dataset storage required by traditional retraining approaches. The segmental learning paradigm provides reconstruction resistance through nonlinear transformations, making recovery of original features computationally challenging. All experimental code, configurations, and energy measurement utilities are released under the MIT License with proper dataset attribution, separating data loading utilities (subject to individual OpenML licensing) from algorithmic implementations to facilitate reproducible research while respecting licensing requirements.