IMLP: AN ENERGY-EFFICIENT CONTINUAL LEARNING METHOD FOR TABULAR DATA STREAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tabular data streams are rapidly emerging as a dominant modality for real-time decision-making in healthcare, finance, and the Internet of Things (IoT). These applications commonly run on edge and mobile devices, where energy budgets, memory, and compute are strictly limited. Continual learning (CL) addresses such dynamics by training models sequentially on task streams while preserving prior knowledge and consolidating new knowledge. While recent CL work has advanced in mitigating catastrophic forgetting and improving knowledge transfer, the practical requirements of energy and memory efficiency for tabular data streams remain underexplored. In particular, existing CL solutions mostly depend on replay mechanisms whose buffers grow over time and exacerbate resource costs.

We propose a *context-aware incremental Multi-Layer Perceptron (IMLP)*, a compact continual learner for tabular data streams. IMLP incorporates a windowed scaled dot-product attention over a sliding latent feature buffer, enabling constant-size memory and avoiding storing raw data. The attended context is concatenated with current features and processed by shared feed-forward layers, yielding lightweight per-segment updates. To assess practical deployability, we introduce *NetScore-T*, a tunable metric coupling balanced accuracy with energy for Pareto-aware comparison across models and datasets. IMLP achieves up to $27.6\times$ higher energy efficiency than TabNet and $85.5\times$ higher than TabPFN, while maintaining competitive average accuracy. Overall, IMLP provides an easy-to-deploy, energy-efficient alternative to full retraining for tabular data streams.

1 INTRODUCTION

Tabular data, structured as a collection of features and instances, is one of the most common and practical data types in practical machine learning applications, such as in fields of healthcare (Lee & Lee, 2020; Amrollahi et al., 2022), finance (Ramjattan et al., 2024; Li et al., 2024a), and IoT (Li et al., 2025b). As such domains increasingly rely on streaming data sources, tabular data streams are gaining significant attention due to their ability to capture continuous, real-time updates rather than static snapshots (Borisov et al., 2022). In particular, most such scenarios often occur on edge devices, IoT systems, and mobile platforms, where energy budgets, battery life, and computational resources are severely constrained Chang et al. (2021).

To tackle those real-world dynamics, Continual Learning (CL) (Wang et al., 2024a), also referred to as lifelong learning (Lee & Lee, 2020), enables models to incrementally acquire, update, accumulate, and exploit knowledge over time. While significant progress has been made on overcoming catastrophic forgetting (Kemker et al., 2018; Li et al., 2019; Bhat et al., 2022) and knowledge transfer (Ke et al., 2021; Li et al., 2024b; Shi et al., 2024a), much less is known about their computational analysis and energy efficiency (Li et al., 2023; Trinci et al., 2024).

Energy-efficient continual learning has become a practical necessity for real-world applications that need to adapt in real time on resource-constrained platforms (Chavan et al., 2023; Shi et al., 2024b; Trinci et al., 2024; Xiao et al., 2024). Meanwhile, most CL progress to date targets image (Trinci et al., 2024; Chavan et al., 2023; Shi et al., 2024b) and language tasks (Li et al., 2025a; Wang et al., 2024b). In contrast, tabular data streams remain underexplored. Tabular models that excel on static datasets do not transfer directly to non-stationary streams with tight memory, compute, and energy budgets. Existing CL methods rarely target these constraints. In particular, replay-based strategies rely on

buffers that grow over time, increasing storage and compute, and hindering on-device deployment. This gap motivates methods for tabular streaming CL that sustain accuracy under distribution shift while operating at low energy cost, with fixed memory, and without storing raw examples. Achieving this under strict resource budgets while mitigating catastrophic forgetting remains a central challenge for Green AI (Henderson et al., 2020; Bouza et al., 2023; Trinci et al., 2024; Różycki et al., 2025).

This paper introduces *Incremental Multi-Layer Perceptron (IMLP)*, a novel method for energy-efficient continual learning, particularly focusing on tabular data streams. IMLP augments a simple MLP with self-attention capabilities, while maintaining efficiency in compute, memory, and energy usage. To be specific: 1) IMLP employs a windowed scaled dot-product attention with a sliding feature buffer, enabling the model to adaptively attend to the most relevant parts of the stream while storing only latent features without needing to revisit raw historical data. 2) The resulting attended representation is concatenated and passed through two shared feed-forward layers followed by a classifier head, serving as the MLP learner for classification tasks. This design avoids the unbounded growth in memory inherent to replay baselines (Rebuffi et al., 2017; Li & Hoiem, 2017; Lopez-Paz & Ranzato, 2017), while remaining computationally lightweight on resource-constrained devices. To evaluate hardware-grounded accuracy–energy trade-offs in continual learning on tabular data streams, we introduce *NetScore-T*, a stream-aware aggregate that couples per-segment performance with a logarithmic energy penalty.

We evaluate IMLP on 36 benchmark tabular datasets designed to assess models under temporal distribution shifts, which provides a systematic comparison across diverse algorithms. IMLP is benchmarked against state-of-the-art (SOTA) tabular models, with results showing that it provides an efficient and competitive neural network alternative. While gradient-boosting methods, such as LightGBM (Ke et al., 2017), still achieve competitive overall accuracy with shorter training time, IMLP demonstrates a favorable trade-off between performance and energy efficiency.

2 Related Work

Traditional tabular data models can be roughly categorized into three main groups: Gradient-Boosted Decision Trees (GBDTs) (Friedman, 2001), Neural Networks (NNs) (Goodfellow et al., 2016), and classic models (e.g., SVMs (Cortes & Vapnik, 1995), k-NN (Cover & Hart, 1967), linear model (Cox, 1958), and simple decision trees (Loh, 2011)).

GBDTs and their variants for CL. Traditional GBDTs such as XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2019) remain strong baselines for tabular classification due to their efficiency and robustness, especially on large or irregular static datasets. However, they are not naturally suited for continual learning: (1) new data typically requires retraining from scratch, since tree splits and boosting weights depend on the full dataset (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2019); (2) without access to past data, models trained only on new samples overwrite previous knowledge, causing catastrophic forgetting (Wang et al., 2024a); and (3) unlike neural networks, GBDTs lack mechanisms for knowledge transfer across tasks (Ke et al., 2021; Parisi et al., 2019; De Lange et al., 2021). Extensions such as online bagging and boosting (Oza & Russell, 2001) or warm-starting (Pedregosa et al., 2011), and adaptive XGBoost (Montiel et al., 2020), partially mitigate these issues, but remain limited in long-term knowledge retention due to the lack of representation reuse, especially when compared to neural continual learning methods.

Classic models in CL. Both standard SVMs (Cortes & Vapnik, 1995) and decision trees (Loh, 2011) are batch learners, requiring retraining on the full dataset when new tasks arrive. SVMs can be extended to continual learning through incremental or online variants such as incremental SVM (Cauwenberghs & Poggio, 2000), LASVM (Bordes et al., 2005), and NORMA (Kivinen et al., 2004), which handle streaming updates but still face challenges with scalability, memory growth, and forgetting. k-NNs (Cover & Hart, 1967) trivially avoid forgetting if all data is stored, but this violates the constraint of no access to past raw inputs and is impractical under resource limits. Linear models (Cox, 1958) are efficient but prone to forgetting under distribution shifts, as updates overwrite prior knowledge. Incremental decision trees, such as Hoeffding Trees (Domingos & Hulten, 2000), and streaming ensembles (Bifet et al., 2010; Gomes et al., 2017) can adapt to data streams without full retraining, but their accuracy degrades under severe drift, they lack strong representation learning, and ensemble methods can be computationally expensive.

Neural models in CL. Recent studies demonstrate that advanced NNs (Zabërgja et al., 2024; Arik & Pfister, 2021; Kadra et al., 2021a; Gorishniy et al., 2023a; Hollmann et al., 2025b; Ye et al., 2024; Gorishniy et al., 2024) can surpass GBDTs on static tabular data in certain regimes, e.g., with well-regularized MLPs (Kadra et al., 2021a), attention-based models such as SAINT (Somepalli et al., 2021), or meta-learned foundation models like TabPFN and its variants (Hollmann et al., 2025b). While their training is typically computationally intensive than that of GBDTs unless carefully tuned (Kadra et al., 2021a), NNs are generally better suited for streaming data, owing to their rich representations, incremental updates via stochastic gradient descent, and flexible architectures. However, vanilla NNs still suffer from catastrophic forgetting in the absence of CL strategies (Wang et al., 2024a).

CL strategies with neural models. In NNs, CL strategies are commonly categorized into regularization-based approaches (Kirkpatrick et al., 2017; Zenke et al., 2017), replay-based strategies (Rebuffi et al., 2017; Shin et al., 2017), attention-based retrieval mechanisms (Chaudhry et al., 2019; Aljundi et al., 2017), and architectural methods (Rusu et al., 2016). Regularization-based methods, such as EWC (Kirkpatrick et al., 2017), SI (Zenke et al., 2017), MAS (Aljundi et al., 2017), and LwF (Li & Hoiem, 2016), mitigate forgetting by constraining updates to parameters deemed important for previously learned tasks. Replay-based strategies, including iCaRL (Rebuffi et al., 2017) and generative replay (Shin et al., 2017), maintain past knowledge by rehearsing stored samples or synthetic data. Attention-based retrieval mechanisms, such as A-GEM with attention (Chaudhry et al., 2019) and attentive experience replay (Aljundi et al., 2017), employ attention to prioritize and retrieve relevant past experiences. Architectural methods, exemplified by PNNs (Rusu et al., 2016), expand model capacity by freezing previously trained components and introducing new modules for incoming tasks.

Despite recent progress, energy-efficient CL for tabular data streams remains largely unexplored (Chavan et al., 2023; Trinci et al., 2024). Real-world tables frequently undergo domain drift (e.g., quarterly finance transactions, evolving sensor logs, healthcare data) without changes to the label space, yet no standardized Domain-Incremental Learning benchmark currently exists for tabular streams. Moreover, pre-trained transformers for tabular data (Gorishniy et al., 2023b; Hollmann et al., 2025b) and feature-level or attention-based CL strategies (Pellegrini et al., 2020; Vaswani et al., 2017a; Jha et al., 2023) show promise for low-storage, privacy-preserving CL, but their effectiveness under domain drift has not been systematically evaluated. Here, we bridge this gap by introducing our method, establishing fair comparisons, and quantifying energy–performance trade-offs.

3 PRELIMINARIES

Continual learning. Owing to the general difficulty and diversity of challenges in continual learning, we focus on a simplified task incremental learning setting (Parisi et al., 2019; De Lange et al., 2021). In this setting, a model is trained on a sequence of tasks $\{\mathcal{T}_t\}_{t=1}^{T}$, where the training data for each task arrives incrementally at time t. Training continues until convergence on each task. Each task $\mathcal{T}_t$ is associated with data $(\mathcal{X}_t, \mathcal{Y}_t)$ randomly drawn from distribution $\mathcal{D}_t$, where $\mathcal{X}_t$ denotes the set of data samples and $\mathcal{Y}_t$ is the corresponding ground truth labels. The key objective is to acquire new knowledge from the current task while maintaining performance on previously learned tasks.

Formally, given a model $f_t(\theta)$ with parameters θ for task $\mathcal{T}_t$, loss function $\ell(\cdot)$ (e.g., cross-entry), the number of tasks seen so far T, the learner is updated sequentially by minimizing the expected risk across all observed tasks, with limited or no access to the data from earlier tasks $t < T$,

$$\sum_{t=1}^{T} \mathbb{E}_{(\mathcal{X}_t, \mathcal{Y}_t) \sim \mathcal{D}_t} [\ell(f_t(\mathcal{X}_t; \theta), \mathcal{Y}_t)] \tag{1}$$

Multi-layer perceptron (MLP). A standard MLP with one hidden layer computes hidden activations as $\mathbf{h} = g\big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big)$ and produces class probabilities $p_\theta(y)$ through $\text{softmax}(W^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$, where $g(\cdot)$ is a pointwise nonlinearity (e.g., ReLU (Daubechies et al., 2022)). The model parameter consists of weight matrices $W^{(1)} \in \mathbb{R}^{d_h \times d_{\text{in}}}$ and $W^{(2)} \in \mathbb{R}^{C \times d_h}$, along with bias vectors $\mathbf{b}^{(1)} \in \mathbb{R}^{d_h}$, $\mathbf{b}^{(2)} \in \mathbb{R}^{C}$ (Goodfellow et al., 2016). In the task incremental learning setting, a pre-trained MLP is then sequentially trained on a sequence of T tasks, typically tabular classification tasks, each defined by a disjoint set of input-label distributions.

Dot-Product Attention. The scaled dot-product attention mechanism forms the basis of most modern attention-based architectures. Given queries $Q \in \mathbb{R}^{n_q \times d_k}$, keys $K \in \mathbb{R}^{n_k \times d_k}$, and values $V \in \mathbb{R}^{n_k \times d_v}$, the attention output (Vaswani et al., 2017b) is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{2}$$

Here, the similarity between a query $q_i \in Q$ and a key $k_j \in K$ is computed via their inner product, scaled by $\sqrt{d_k}$ to mitigate the effect of large dot products when the dimensionality d_k is high. More explicitly, the normalized weight α_{ij} assigned to value v_j for query q_i, and the resulting attended representation z_i are given by

$$\alpha_{ij} = \frac{\exp\left(\frac{q_i^\top k_j}{\sqrt{d_k}}\right)}{\sum_{j'=1}^{n_k} \exp\left(\frac{q_i^\top k_{j'}}{\sqrt{d_k}}\right)}, \qquad z_i = \sum_{j=1}^{n_k} \alpha_{ij}\, v_j. \tag{3}$$

Trade-off measures. In many optimization problems, objectives are inherently conflicting; for instance, improving the accuracy of a neural network increases energy consumption or latency. To systematically evaluate such trade-offs, Trinci et al. (2024) proposed the Energy NetScore metric, which balances predictive accuracy against energy consumption, originally derived from NetScore (Wong, 2019). For a model $\mathcal{M}$, the Energy NetScore is computed as $20 \log\left(\frac{A(\mathcal{M})^\alpha}{E(\mathcal{M})^\beta}\right)$, where $A(\mathcal{M})$ is predictive accuracy and $E(\mathcal{M})$ is the total energy consumption; the exponents α and β weight the trade-off between accuracy and energy.

A classical way to study such trade-offs is through Pareto front analysis (Giagkiozis & Fleming, 2014). Let $x = (p, E)$ denote the pair of model performance p and energy consumption E measured during training and inference. Consider a set of candidate solutions $\mathcal{S} = \{x_i = (p_i, E_i) \mid i = 1, 2, \ldots, m\}$. For any two solutions $x_a = (p_a, E_a)$ and $x_b = (p_b, E_b)$, we say that x_a *dominates* x_b if $p_a \geq p_b \quad \text{and} \quad E_a \leq E_b$, with at least one inequality being strict. The Pareto front $\mathcal{P} \subseteq \mathcal{S}$ is then defined as the set of all non-dominated solutions:

$$\mathcal{P} = \{x^* \in \mathcal{S} \mid \nexists\, x' \in \mathcal{S} \text{ such that } x' \text{ dominates } \vec{x}\}. \tag{4}$$

The Pareto front thus provides a set of optimal trade-offs. By examining its shape and Pareto efficiency, one can assess how much performance must be sacrificed to achieve energy savings.

4 IMLP: AN INCREMENTAL MLP FOR TABULAR DATA STREAMS

Problem Statement. Let $\{\mathcal{T}_t\}_{t=1}^T$ denote a stream of T segments, where the training data for each task arrives incrementally at time t. The raw inputs are real-valued feature vectors $x_i \in \mathcal{X}_t \subseteq \mathbb{R}^{d_{\text{in}}}$, and the corresponding labels are categorical $y_i \in \mathcal{Y}_t$, sampled from distribution $\mathcal{D}_t$. Our goal is to train an incremental MLP model using data $\mathcal{T}_t$ available at time t, where the learner may scan the current data multiple times but cannot revisit raw data from earlier tasks. The training objective is to minimize the cross-entropy loss. We aim to train the model without storing raw inputs from past tasks, and to evaluate its performance during inference on a stratified test set.

Architecture Overview. For efficient learning from the current task while maintaining performance on previously learned tasks, we consider two strategies: (1) processing each task with an augmented MLP module that incorporates limited historical context through a variant of scaled dot-product attention. 2) maintaining an FIFO feature buffer with fixed memory over time, which facilitates representation reuse while keeping memory and computation cost constrained as new data evolves.

We aim to reuse a compact, input-dependent summary of recent experience without retaining raw data, while preventing unbounded growth in memory and computation. To this end, we employ a sliding window of size W to cache 256-dimensional (256-D) feature vectors. By attending to latent features, the model enables privacy-preserving rehearsal with a constant memory footprint. The architecture of IMLP is presented in Figure 1. Each incoming sample $\mathbf{x} \in \mathcal{T}t$ is mapped from the dimensional input d_{in} to a 256-D query $\mathbf{Q}$ within a windowed attention module. This query retrieves a context vector $\mathbf{c}$ from a *fixed-size* feature memory that stores the most recent W segment embeddings, $\mathbf{H}_{\text{stacked}} = \{h_{t-1}, \ldots, h_{t-W}\}$. When the attention gate is active, the concatenated pair

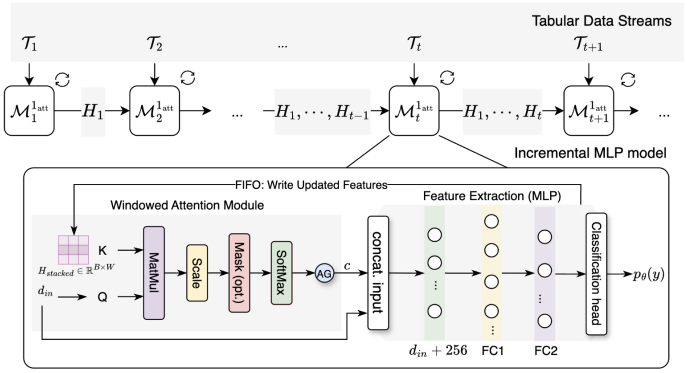


Figure 1: IMLP architecture. IMLP sequentially takes $\mathcal{T}_t$ as input and outputs predictive performance $p_\theta(y)$.

$(\mathbf{x}, \mathbf{c})$ is passed through two shared feed-forward layers followed by a classification head. Next, the model $\mathcal{M}_t^{1\text{att}}$ yields the model's predictions at step t and updates the feature representations.

In the following, we detail these two key component designs to achieve energy-efficient continual learning over tabular data streams.

4.1 Windowed Self-Attention Mechanism

To selectively integrate useful past features into the current segment's representation, IMLP employs *attention gates* (AG) in each window size W, as depicted in Figure 1.

Unlike standard scaled dot-product attention, our windowed variant departs in two controlled ways: (i) the *sequence* is a FIFO buffer of latent features $[h_{t-1}, \ldots, h_{t-W}]$ rather than input tokens, and (ii) we *tie values to keys*, i.e., $V = K$, to reduce parameters and latency while preserving the dot-product inductive bias.

Algorithm 1 Sliding Window Update

Require: Current input x, H_{prev}, W
Ensure: Updated feature H_{new}
$h_{\text{current}} \leftarrow \text{FeatureExtractor}(x, \text{Context}(x))$
$H_{\text{new}} \leftarrow H_{\text{prev}} \cup \{h_{\text{current}}\}$
if $|H_{\text{new}}| > W$ **then**
  $H_{\text{new}} \leftarrow H_{\text{new}}[1:]$ ▷ Remove oldest feature
return H_{new}

Let $d_h = 256$ and B denote the batch size. Stacking the last W penultimate features along the temporal dimension yields $H \in \mathbb{R}^{B\times W\times d_h}$. For each input $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$, we compute $Q = W_q x \in \mathbb{R}^{B\times 1\times d_h}$, $K = W_k H \in \mathbb{R}^{B\times W\times d_h}$, $\alpha = \text{softmax}(\frac{\text{scores}}{\sqrt{d_h}}) \in \mathbb{R}^{B\times W\times 1}$, where $\text{scores} = \text{bmm}(K, Q^\top) \in \mathbb{R}^{B\times W\times 1}$. The context vector is then computed as $c = \text{bmm}(\alpha^\top, K) \in \mathbb{R}^{B\times 1\times d_h}$ through the attention module.

Next, squeezing the singleton dimension yields $\tilde{c} \in \mathbb{R}^{B\times d_h}$, which is concatenated with the input $z = [x; \tilde{c}] \in \mathbb{R}^{B\times(d_{\text{in}}+d_h)}$. This representation is then fed into an MLP consisting of a two-layer feature extractor followed by a linear classification head as

$$h = \sigma\big(W_2\,\sigma(W_1 z + b_1)\big) \in \mathbb{R}^{B\times d_h}, \qquad o = W_c h + b_c \in \mathbb{R}^{B\times C}, \qquad p = \text{softmax}(o). \quad (5)$$

After each forward pass, the detached penultimate feature $\bar{h}_t$ is appended to a FIFO buffer, and the oldest entry is discarded once the buffer holds W items.

4.2 Sliding Window Management

The model retains a set of feature vectors extracted from representative past samples, typically the activations from the penultimate layer. IMLP maintains a FIFO feature buffer update so that it continually captures updated latent features from previous tasks, as shown in Algorithm 1.

Let F_t denote the FIFO buffer of the last W *segment prototypes*, each of which is a detached penultimate feature vector. After consuming all minibatches of $\mathcal{T}_t$, we compute

$$\vec{f}_t = \mathrm{Detach}\Big(\frac{1}{|\mathcal{T}_t|}\sum\nolimits_{(\mathcal{X}_t,\mathcal{Y}_t)\in\mathcal{T}_t} h(x)\Big), \qquad F_t \leftarrow \mathrm{truncate\_last}\big(F_{t-1}\cup\{\vec{f}_t\},\, W\big). \tag{6}$$

Notably, IMLP stores only latent features instead of raw samples. Optionally, stored features can be ℓ_2-normalized before enqueueing, i.e., $\tilde{h} = \frac{h}{\|h\|_2+\varepsilon}$, which stabilizes attention weights by focusing on feature directions. Likewise, we enqueue normalized prototypes $\tilde{f}_t = f_t/(\|f_t\|_2+\varepsilon)$ with $\varepsilon = 10^{-8}$.

4.3 Memory and Computational Efficiency

In terms of memory complexity, the only additional cost arises from the feature buffer used by the attention mechanism, given by $O(Wd_h)$. Since the oldest feature is discarded once the window is full, this overhead remains constant with respect to the number of tasks T. Unlike replay-based methods, our approach avoids memory growth with the number of segments, thereby preventing excessive storage costs. Moreover, this design also contributes to lightweight computation.

Given a mini-batch of B samples and a sliding window of W cached 256-D feature vectors, the total computational efficiency in our model can be formally stated by,

$$\underbrace{O(Bd_{\mathrm{in}}d_h)}_{\text{query}} + \underbrace{O(BWd_h^2)}_{\text{key projection}} + \underbrace{O(BWd_h)}_{\text{scores}} + \underbrace{O(BWd_h)}_{\text{aggregation}} + \underbrace{O(B(d_{\mathrm{in}}+d_h)512)}_{\text{feature MLP}} = O\big(B(d_{\mathrm{in}}d_h + Wd_h^2)\big) \tag{7}$$

where 512 is the feature dimension of the FC1, FC2 layer in the feature extraction module using MLP. Hence, with W and d_h fixed, IMLP exhibits (empirically) linear cumulative energy growth with the number of segments, in contrast to the quadratic growth of cumulative retraining. Further analysis is provided in Appendix B.

Summary of IMLP's strengths. IMLP offers several notable advantages over related tabular methods: (1) it is simple and inherently suitable for streaming tabular learning without replaying past raw inputs; (2) it is lightweight in both computation and memory, with costs independent of task size, yielding an energy-efficient solution; (3) it is straightforward to deploy on hardware; and (4) it aligns well with privacy-sensitive applications by avoiding storage of raw data.

5 Energy Efficiency Analysis

Measuring energy consumption. To obtain ground-truth measurements, we instrument our CL pipeline with an ElmorLabs PMD-USB power meter and corresponding PCI-E slot adapter (ElmorLabs, 2023; 2025). This device captures fine-grained voltage/current sampling with millisecond-level resolution that is drawn for CPU and GPU throughout online updates, enabling precise monitoring of transient spikes, including the training and inference phases. Different from software-based measurements introduced by Trinci et al. (2024), this hardware-level setup provides real-life wall power and vendor-independent measurements of actual power draw without introducing runtime overhead. The reported energy values will always refer to the total energy that integrates power readings over the training and inference duration for each segment t, that is $E(\mathcal{M}_t) = \int_{t_{\text{start}}}^{t_{\text{end}}} P_{\text{total}}(t)\,dt$, where $P_{\text{total}}(t)$ includes both CPU and GPU power consumption during training and inference.

NetScore-T. NetScore-T extends the NetScore framework (Shafiee et al., 2018) to CL by jointly assessing the model's predictive performance and energy consumption across data segments. Let $P(\mathcal{M}_t)(\geq 0)$ denote a performance measure (e.g., balanced accuracy) of the model $\mathcal{M}$ on segment $t \in \{\mathcal{T}_t\}_{t=1}^{T}$ and its total energy consumed $E(\mathcal{M}_t)$. We define the per-segment score as $\mathrm{NS}(\mathcal{M}_t)$, and as consequence, its stream aggregate

$$\text{NetScore-T}(\mathcal{M}) = \tfrac{1}{T}\sum_{t=1}^{T}\mathrm{NS}(\mathcal{M}_t), \quad \text{where} \quad \mathrm{NS}(\mathcal{M}_t) = \frac{P(\mathcal{M}_t)}{\log_{10}(E(\mathcal{M}_t)+1)} \tag{8}$$

High NetScore-T values indicate models that combine strong accuracy with low energy usage. Wide empirical ranges can reflect the diversity of energy consumption patterns across models and datasets.

Logarithmic energy scaling. NetScore-T penalizes energy via a base-10 logarithm (see equation 8), which compresses the wide dynamic range of E_t across hardware and datasets, preventing any

outlier segment from dominating the stream average. The $+1$ term ensures the expression remains well-defined as $E \to 0$. For fixed performance $P(\mathcal{M}_t)$, the mapping $E \mapsto \log_{10}(E+1)$ is strictly decreasing with diminishing penalties: additional Joules always reduce the score, but with progressively smaller marginal impact at higher energy levels.

Additionally, rankings remain invariant to the choice of logarithm base (a constant rescaling) and are practically unaffected by energy unit rescaling ($E \mapsto kE$), because $\log_{10}(kE+1) = \log_{10} k + \log_{10}(E + \frac{1}{k})$, which behaves nearly as an additive shift in the typical regime $E \gg 1$.

Unlike scalarizations that require exponent tuning, the logarithmic transform introduces no additional hyperparameters.

6 EXPERIMENTS

Setup and Configuration. All experiments were conducted on a single workstation equipped with an Intel® Core™ i5-8600K Processor, a NVIDIA GeForce RTX 2080 Ti GPU, 16GB DDR4 RAM, and an NVMe SSD for data and model checkpoints. An ElmorLabs PMD-USB power meter and corresponding PCI-E slot adapter (ElmorLabs, 2023; 2025) were used for real-life energy consumption measurement.

Datasets. We have conducted experiments on 36 classification tasks from the TabZilla benchmarks (McElfresh & Talwalkar, 2023), selected from OpenML to ensure (i) sufficient size for meaningful segmentation, (ii) a balanced mix of binary and multi-class problems, and (iii) diversity in feature dimensionalities and label distributions. To simulate data streams, each dataset is partitioned chronologically into contiguous segments of size 500–1000 instances, determined by an algorithm that minimizes remainder imbalance while preserving temporal order (see Appendix §A.1.1). Any remainder is redistributed across the first segments to maintain near-uniform segment sizes. For fair evaluation, all datasets undergo the same preprocessing pipeline: median imputation and standardization for numerical features, constant imputation and one-hot encoding for categorical features, stratified 85%-15% splits for training, validation, and /or testing.

Baselines. We compare against SOTA networks for tabular data, including TabPFN v2 (Hollmann et al., 2025a), TabM (Gorishniy et al., 2024), Real-MLP (Holzmüller et al., 2024), TabR (Gorishniy et al., 2023a), and ModernNCA (Ye et al., 2024), as well as representative methods such as MLP (Taud & Mas, 2017), TabNet (Arik & Pfister, 2021), DANet (Chen et al., 2022), ResNet (Gorishniy et al., 2021), STG (Jana et al., 2023), VIME (Yoon et al., 2020)). To provide a broader perspective, we also include tree-based gradient boosting methods (XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017), CatBoost (Prokhorenkova et al., 2019)), and classical models (k-NN (Guo et al., 2003), SVM (Jakkula, 2006), Linear Model (Kiebel & Holmes, 2007), Random Forest (Rigatti, 2017), Decision Tree (Rokach & Maimon, 2005)), although they require replay and are not directly suited to our problem setting. Since the recent baselines we consider were not originally designed for stream learning, we adopt a best-effort comparison: at each segment step t, all baselines are retrained from scratch on the cumulative data available so far to mitigate catastrophic forgetting and maximize their performance. In contrast, our method operates in a true incremental mode without replay of past data. Appendix §A.2.5 provides details on model training protocols.

Evaluation. To assess whether performance differences among algorithms are statistically significant across multiple datasets, we first conduct the classic Friedman test (Friedman, 1937), $\chi_F^2 = \frac{12N}{k(k+1)}\big[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\big]$, where N is the number of datasets, i.e., 36, k the number of algorithms and R_j the average rank of the j-th algorithm. If the null hypothesis is rejected, we perform post-hoc analyses using the Wilcoxon signed-rank test (Wilcoxon, 1945) with Holm correction (Holm, 1979), as well as critical difference analysis (Nemenyi, 1963). All models are evaluated on six key metrics: balanced accuracy, log-loss, energy consumption, execution time, and the composite NetScore-T metrics capturing accuracy–efficiency trade-offs. Furthermore, we plot 2D Pareto fronts of performance versus energy efficiency to examine the trade-offs among models. More details have been provided in Appendix C.

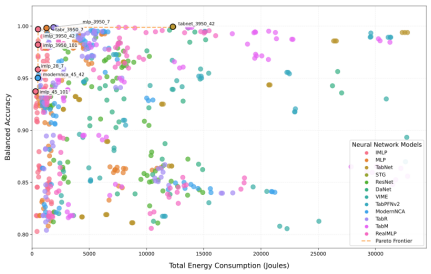
IMLP excels on the energy–accuracy trade-off under no replay. Table 1 summarizes the energy consumption, time cost, and performance among neural methods. Neural methods generally demand higher energy and runtime, particularly for large networks, TabPFN v2 (72,319J), DANet (32,382J),

Table 1: Performance statistics across all neural network baselines on 36 TabZilla datasets, reported as mean±standard deviation. Energy is measured in Joules, and Time in seconds. Trade-offs between balanced accuracy and energy consumption are measured by NetScore-T (see Eq. 8).

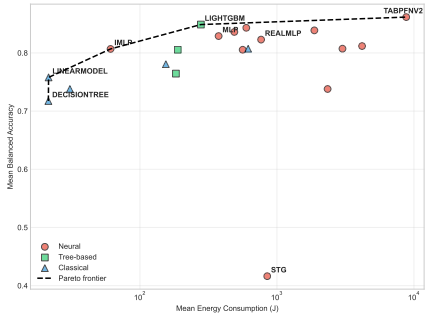
Model	Energy Consumed (J) (↓)	Time (s) (↓)	Balanced Accuracy (↑)	Log Loss (↓)	NetScore-T (↑)
STG	6747 ± 5671	85.6 ± 71.8	0.416 ± 0.166	1.162 ± 0.689	0.151 ± 0.073
VIME	21705 ± 34212	261.8 ± 416.9	0.738 ± 0.197	1.098 ± 1.544	0.238 ± 0.060
DANet	32382 ± 26865	406.3 ± 336.4	0.812 ± 0.172	0.349 ± 0.334	0.248 ± 0.055
TabNet	23312 ± 18268	285.3 ± 226.3	0.807 ± 0.177	0.357 ± 0.337	0.250 ± 0.059
TabPFN v2	72319 ± 100877	291.1 ± 405.3	**0.862** ± 0.151	**0.240** ± 0.265	0.252 ± 0.052
TabM	15558 ± 12321	96.6 ± 76.8	0.839 ± 0.161	0.288 ± 0.307	0.280 ± 0.056
ResNet	5422 ± 3672	66.1 ± 45.1	0.805 ± 0.160	1.489 ± 1.469	0.312 ± 0.065
Real-MLP	6243 ± 4735	62.8 ± 47.4	0.823 ± 0.159	0.342 ± 0.314	0.313 ± 0.065
MLP	3241 ± 2581	41.4 ± 32.9	0.829 ± 0.162	0.329 ± 0.326	0.341 ± 0.073
TabR	4125 ± 3625	46.1 ± 41.2	0.836 ± 0.155	0.554 ± 0.656	0.345 ± 0.072
ModernNCA	4829 ± 4630	54.9 ± 53.1	0.843 ± 0.145	1.298 ± 1.344	0.346 ± 0.077
IMLP (Ours)	**845** ± 386	**9.9** ± 4.5	0.807 ± 0.164	0.399 ± 0.365	**0.430** ± 0.095

TabNet (23,312J), and TabM (15,558J) on average, though they still achieve competitive accuracy, namely 0.862, 0.812, 0.807, and 0.839, respectively. Among them, IMLP stands out for its superior energy–accuracy trade-off under the no-replay setting. On average, it requires only 845J, which is $27.6\times$ lower than TabNet (23,312J) at comparable accuracy and $85.5\times$ lower than TabPFN v2 (72,319J), with only a 0.055 drop in balanced accuracy. Compared to standard MLP, IMLP achieves a $4.2\times$ speedup on average, along with a 73.9% reduction in energy usage.

The energy-accuracy trade-off, as measured by NetScore-T, shows that IMLP ranks as the most energy-efficient method under the no-replay setting. Among all evaluated methods, GBDTs such as LightGBM consistently deliver strong performance with relatively low runtime. However, IMLP beats CatBoost and XGBoost in terms of both average energy consumption and runtime (see Appendix C). Classic methods such as DecisionTree, RandomForest, and LinearModel use very little energy and time, but their balanced accuracy is generally low (< 0.8). SVM and k-NN are useful for reference; however, they require more computation and energy usage compared to IMLP. Specifically, while the average balanced accuracy of SVM is the same as that of IMLP, it requires about $4.4\times$ more energy and takes $8.5\times$ longer on average (see Appendix C).



(a) Pareto optimal trade-offs for NNs.



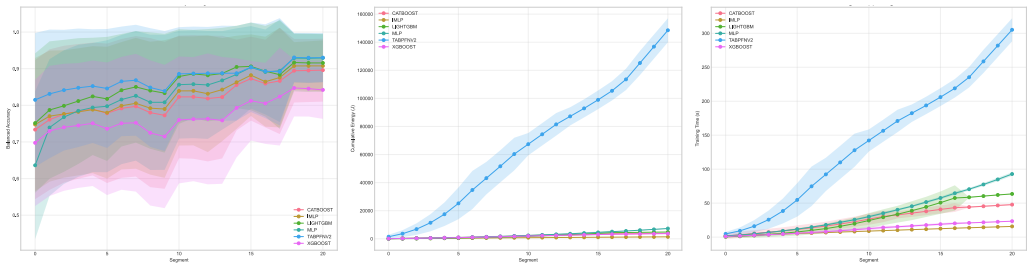
(b) Pareto optimal trade-offs for all family.

Figure 2: Pareto frontiers overview. (a) optimal trade-offs for NN models. (b) optimal trade-offs across all families.

NetScore-T is compatible with Pareto efficiency. Figure 2 depicts the optimal trade-offs among neural tabular models using a Pareto-2D visualization. Each point represents a specific model training and inference run on a given dataset indexed by ID (see Appendix A) with a particular random seed (e.g., 7, 42, and 101), providing a detailed view of the Pareto optimal trade-offs in this group. As shown in sub Figure 2a, the frontier line maps the optimal trade-offs, in which most (four out of eight) of the optimal trade-offs in neural tabular models come from IMLP, with one from TabNet, one from MLP, one from ModernNCA, and one from TabR. IMLP points (pink) lie in the low-energy region while still keeping high accuracy. In contrast, TabNet attains relatively high accuracy but at the cost of substantially higher energy consumption, often exceeding 10,000 joules. MLP and TabR fall in the middle ground, showing

balanced trade-offs without the extremes observed in IMLP or TabNet. Some come close to the frontier, but most need more energy than IMLP. Surprisingly, for all model families, as shown in Figure 2b, IMLP also becomes one of the Pareto optimal trade-offs between the average balanced accuracy and average total energy consumption. The others include DecisionTree, LinearModel, LightGDM, and TabPFN v2. Overall, IMLP distinguishes itself as an efficient and competitive neural counterpart.

Furthermore, we compare IMLP with the SOTA methods, namely TabPFN v2, LightGBM, CatBoost, XGBoost, and MLP, about the dynamic performance and efficiency when data arrives in sequence, as illustrated in Figure 3. Figure 3a presents the balanced accuracy averaged across datasets for



(a) Balanced Accuracy. (b) Cumulative energy. (c) Training time.

Figure 3: Dynamic learning performance and efficiency comparisons per segment.

each segment. The accuracy generally increases across segments, showing that all models, including IMLP, improve with more data. Figure 3b displays cumulative energy consumption per segment. Notably, TabPFN v2 exhibits substantially higher energy consumption compared to others, primarily due to its large neural network backbone. The results reveal that as the segment step increases, MLP, LightGBM, CatBoost, and XGBoost consume energy at accelerating rates, while IMLP sustains the lowest and most stable energy usage across all segments. Similarly, Figure 3c shows training time per segment. Its trend closely follows that of cumulative energy consumption, as expected from the strong correlation between training time and energy use.

7 Conclusion

This paper addresses the critical gap of energy-efficient continual learning on tabular data streams by introducing IMLP, a novel incremental MLP model. IMLP employs attention-based feature replay with context retrieval and sliding buffer updates, integrated into a minibatch training loop for streaming tabular learning. We further propose NetScore-T, a new metric that jointly evaluates balanced accuracy and energy consumption and can be compatible with traditional Pareto efficiency. IMLP achieves outstanding energy savings compared to SOTA neural tabular models and excels on energy-accuracy trade-off under no replay, according to hardware-level energy measurement.

Experiments show that IMLP matches the accuracy of neural baselines under no replay while substantially reducing runtime and energy costs. IMLP achieves up to $27.6\times$ higher energy efficiency than TabNet and $85.5\times$ higher than TabPFN, while maintaining competitive average accuracy. Positioned optimally on the neural Pareto frontier, IMLP consistently delivers efficiency gains across diverse datasets.

Limitations and Future Work. Despite these exciting findings, IMLP currently treats baselines on classic TabZilla benchmarks in an experimental setting. A promising next step is to compare the method with up-to-date models on real-life lifelong settings, thereby enriching the benchmarks (e.g., TabRed (Rubachev et al., 2024)). Beyond that, conducting a comprehensive ablation study would shed light on the influence of key parameter choices, such as window size, feature dimensions, scaling, and alternative CL strategies. Ultimately, an important future direction is to extend IMLP toward jointly optimizing the trade-offs between energy efficiency and predictive performance, ideally supported by theoretical guarantees or unified analytical frameworks.

Ethics statement. This work contributes to an easy-to-deploy, energy-efficient alternative to full retraining for tabular data streams. By a windowed scaled dot-product attention over a sliding latent feature buffer, it enables lightweight computation and avoids unbounded memory growth in continual learning, while achieving efficient energy consumption. This method will be beneficial for Green AI, especially in resource-constrained tabular data learning. All experiments are conducted on publicly available benchmark datasets and baselines. Regarding the large language model use, ChatGPTs and Grammarly were used to assist us with writing and editing, retrieving related work, coding improvement, but all the ideas, designs, plots, and analyses are our own.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. *CoRR*, abs/1711.09601, 2017. URL http://arxiv.org/abs/1711.09601.

Fatemeh Amrollahi, Supreeth P Shashikumar, Andre L Holder, and Shamim Nemati. Leveraging clinical data across healthcare institutions for continual learning of predictive risk models. *Scientific reports*, 12(1):8380, 2022.

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 6679–6687, 2021.

Prashant Shivaram Bhat, Bahram Zonooz, and Elahe Arani. Consistency is the key to further mitigating catastrophic forgetting in continual learning. In *Conference on Lifelong Learning Agents*, pp. 1195–1212. PMLR, 2022.

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 135–150. Springer, 2010.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of machine learning research*, 6(Sep):1579–1619, 2005.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 35(6):7499–7519, 2022.

Lucía Bouza, Aurélie Bugeau, and Loïc Lannelongue. How to estimate carbon footprint when training deep learning models? a guide and review. *Environmental Research Communications*, 5 (11):115014, November 2023. ISSN 2515-7620. doi: 10.1088/2515-7620/acf81b. URL http://dx.doi.org/10.1088/2515-7620/acf81b.

Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, 13, 2000.

Zhuoqing Chang, Shubo Liu, Xingxing Xiong, Zhaohui Cai, and Guoqing Tu. A survey of recent advances in edge-computing-powered artificial intelligence of things. *IEEE Internet of Things Journal*, 8(18):13849–13875, 2021.

A. Chaudhry, M. Rohrbach, M. Elhoseiny, S. Dsouza, T. Ajanthan, and P. K. Dokania. Efficient lifelong learning with a-gem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1396–1405, 2019. doi: 10.1109/CVPR.2019.00153.

Vivek Chavan, Paul Koch, Marian Schlüter, and Clemens Briese. Towards realistic evaluation of industrial continual learning scenarios with an emphasis on energy consumption and computational footprint. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11506–11518, 2023.

Jintai Chen, Kuanlun Liao, Yao Wan, Danny Z Chen, and Jian Wu. Danets: Deep abstract networks for tabular data classification and regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 3930–3938, 2022.

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, pp. 785–794. ACM, August 2016. doi: 10.1145/2939672.2939785. URL http://dx.doi.org/10.1145/2939672.2939785.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 20(2):215–232, 1958.

Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) relu networks. *Constructive Approximation*, 55(1):127–172, 2022.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7: 1–30, December 2006. ISSN 1532-4435.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 71–80, 2000.

ElmorLabs. Pmd-usb (power measurement device with usb). https://www.elmorlabs.com/product/elmorlabs-pmd-usb-power-measurement-device-with-usb/, 2023. Accessed: 2025-01-15.

ElmorLabs. Pmd pci-e slot power measurement adapter. https://www.elmorlabs.com/product/pmd-pci-e-slot-power-measurement-adapter/, 2025. Accessed: 2025-01-15.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. ISSN 01621459, 1537274X. URL http://www.jstor.org/stable/2279372.

Ioannis Giagkiozis and Peter J Fleming. Pareto front estimation for decision making. *Evolutionary computation*, 22(4):651–678, 2014.

Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34:18932–18943, 2021.

Yury Gorishniy, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko. Tabr: Tabular deep learning meets nearest neighbors in 2023. *arXiv preprint arXiv:2307.14338*, 2023a.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data, 2023b. URL https://arxiv.org/abs/2106.11959.

Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. Tabm: Advancing tabular deep learning with parameter-efficient ensembling. *arXiv preprint arXiv:2410.24210*, 2024.

Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pp. 986–996. Springer, 2003.

Peter Henderson, Jierui Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 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)*, 2020. arXiv:2002.05651.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025a.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 01 2025b. doi: 10.1038/s41586-024-08328-6. URL https://www.nature.com/articles/s41586-024-08328-6.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. ISSN 03036898, 14679469. URL http://www.jstor.org/stable/4615733.

David Holzmüller, Léo Grinsztajn, and Ingo Steinwart. Better by default: Strong pre-tuned mlps and boosted trees on tabular data. *Advances in Neural Information Processing Systems*, 37: 26577–26658, 2024.

Vikramaditya Jakkula. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37(2.5):3, 2006.

Soham Jana, Henry Li, Yutaro Yamada, and Ofir Lindenbaum. Support recovery with projected stochastic gates: Theory and application for linear models. *Signal Processing*, 213:109193, 2023.

S. Jha et al. Neural processes for continual learning. In *International Conference on Machine Learning*, 2023.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in neural information processing systems*, 34:23928–23941, 2021a.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *NeurIPS*, 2021b. URL https://arxiv.org/abs/2106.11189.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. Achieving forgetting prevention and knowledge transfer in continual learning. *Advances in Neural Information Processing Systems*, 34: 22443–22456, 2021.

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

S Kiebel and A Holmes. *The general linear model*. Academic Press. London, 2007.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

Jyrki Kivinen, Alexander J Smola, and Robert C Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.

Cecilia S Lee and Aaron Y Lee. Clinical applications of continual learning machine learning. *The Lancet Digital Health*, 2(6):e279–e281, 2020.

Ao Li, Chong Zhang, Fu Xiao, Cheng Fan, Yang Deng, and Dan Wang. Large-scale comparison and demonstration of continual learning for adaptive data-driven building energy prediction. *Applied Energy*, 347:121481, 2023.

Xiaodi Li, Dingcheng Li, Rujun Gao, Mahmoud Zamani, and Latifur Khan. Lsebmcl: A latent space energy-based model for continual learning. In *2025 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 0690–0695. IEEE, 2025a.

Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International conference on machine learning*, pp. 3925–3934. PMLR, 2019.

Yichen Li, Haozhao Wang, Wenchao Xu, Tianzhe Xiao, Hong Liu, Minzhu Tu, Yuying Wang, Xin Yang, Rui Zhang, Shui Yu, et al. Unleashing the power of continual learning on non-centralized devices: A survey. *IEEE Communications Surveys & Tutorials*, 2025b.

Yujie Li, Xin Yang, Qiang Gao, Hao Wang, Junbo Zhang, and Tianrui Li. Cross-regional fraud detection via continual learning with knowledge transfer. *IEEE Transactions on Knowledge and Data Engineering*, 2024a.

Yujie Li, Xin Yang, Hao Wang, Xiangkun Wang, and Tianrui Li. Learning to prompt knowledge transfer for open-world continual learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13700–13708, 2024b.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016. URL `http://arxiv.org/abs/1606.09282`.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Wei-Yin Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

David McElfresh and Ameet Talwalkar. Tabzilla benchmark. *NeurIPS*, 2023. Version 1.0, accessed May 2025.

Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Benjamin Feuer, Chinmay Hegde, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? In *Advances in Neural Information Processing Systems (NeurIPS) 2023, Track on Datasets and Benchmarks*, 2023.

Jacob Montiel, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem, and Albert Bifet. Adaptive xgboost for evolving data streams. In *2020 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2020.

Peter Björn Nemenyi. *Distribution-free Multiple Comparisons*. PhD thesis, Princeton University, 1963.

Nikunj C Oza and Stuart J Russell. Online bagging and boosting. In *International workshop on artificial intelligence and statistics*, pp. 229–236. PMLR, 2001.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

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

F. Pellegrini et al. Latent replay for on-device continual learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2020. doi: 10.1109/TNNLS.2020.2971234.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features, 2019. URL https://arxiv.org/abs/1706.09516.

Reshawn Ramjattan, Daniele Atzeni, and Daniele Mazzei. Comparative evaluation of continual learning methods in financial and industrial time-series data. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. IEEE, 2024.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Steven J Rigatti. Random forest. *Journal of insurance medicine*, 47(1):31–39, 2017.

Lior Rokach and Oded Maimon. Decision trees. In *Data mining and knowledge discovery handbook*, pp. 165–192. Springer, 2005.

Ivan Rubachev, Nikolay Kartashev, Yury Gorishniy, and Artem Babenko. Tabred: Analyzing pitfalls and filling the gaps in tabular deep learning benchmarks. *arXiv preprint arXiv:2406.19380*, 2024.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL http://arxiv.org/abs/1606.04671.

Rafał Różycki, Dorota Agnieszka Solarska, and Grzegorz Waligóra. Energy-aware machine learning models—a review of recent techniques and perspectives. *Energies*, 18(11), 2025. ISSN 1996-1073. doi: 10.3390/en18112810. URL https://www.mdpi.com/1996-1073/18/11/2810.

Mohammad Javad Shafiee, Bartłomiej Chywl, Francis Li, and Alexander Wong. Netscore: Towards universal metrics for large-scale performance evaluation of deep neural networks. In *NeurIPS Workshop*, 2018.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 2024a.

Xinyu Shi, Jianhao Ding, Zecheng Hao, and Zhaofei Yu. Towards energy efficient spiking neural networks: An unstructured pruning framework. In *The Twelfth International Conference on Learning Representations*, 2024b.

H. Shin, J. K. Lee, J. Kim, J. Kim, and S. Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

Hind Taud and Jean-Franccois Mas. Multilayer perceptron (mlp). In *Geomatic approaches for modeling land change scenarios*, pp. 451–455. Springer, 2017.

Tomaso Trinci, Simone Magistri, Roberto Verdecchia, and Andrew D. Bagdanov. How green is continual learning, really? analyzing the energy consumption in continual training of vision foundation models. *arXiv preprint arXiv:2409.18664*, 2024. Accepted to GreenFOMO Workshop at ECCV 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017a. URL http://arxiv.org/abs/1706.03762.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017b.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383, 2024a.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024b. URL https://arxiv.org/abs/2302.00487.

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

Alexander Wong. Netscore: towards universal metrics for large-scale performance analysis of deep neural networks for practical on-device edge usage. In *International Conference on Image Analysis and Recognition*, pp. 15–26. Springer, 2019.

Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Hebbian learning based orthogonal projection for continual learning of spiking neural networks. *arXiv preprint arXiv:2402.11984*, 2024.

Zeyu Yang, Karel Adamek, and Wesley Armour. 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*, pp. 1–17. IEEE, November 2024. doi: 10.1109/sc41406.2024.00028. URL http://dx.doi.org/10.1109/SC41406.2024.00028.

Han-Jia Ye, Huai-Hong Yin, and De-Chuan Zhan. Modern neighborhood components analysis: A deep tabular baseline two decades later. *arXiv e-prints*, pp. arXiv–2407, 2024.

Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela Van der Schaar. Vime: Extending the success of self-and semi-supervised learning to tabular domain. *Advances in neural information processing systems*, 33:11033–11043, 2020.

Guri Zabërgja, Arlind Kadra, Christian MM Frey, and Josif Grabocka. Is deep learning finally better than decision trees on tabular data? *arXiv preprint arXiv:2402.03970*, 2024.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pp. 3987–3995. PMLR, 2017.

A Extended Experiments

A.1 Datasets and Stream Segmentation

We evaluate IMLP on 36 classification tasks from the TabZilla benchmark (McElfresh & Talwalkar, 2023), 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 2 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

```
best_remainder ← N
k* ← k_min
for k = k_min to min(k_max, N) do
    num_segments ← ⌊N/k⌋
    remainder ← N mod k
    if remainder = 0 then
        return k                                   ▷ Perfect division found
    if remainder < best_remainder then
        best_remainder ← remainder
        k* ← k
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:

Table 2: Statistics of datasets. OpenML classification tasks and stream-segmentation parameters used in this study. # Inst, stands for the number of instances, # Feat. stands for the number of features. Seg. size stands for the segment size bound. # Segs stands for the number of segments. 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

Numerical Features:

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

A.2.4 Data Splitting Strategy

Our splitting protocol ensures a 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 criteria.

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 $\bigcup_{t=0}^{T} \mathcal{T}_t$ where $\mathcal{T}_t$ 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_t 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 2 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 2. The script requires the preprocessed datasets, but no pipeline re-execution.

A.4 BASELINES

We implement most of the baseline methods according to the publicly available codebases and integrate them into the same backbone for benchmarking.

- XGBoost (Chen & Guestrin, 2016). https://github.com/dmlc/xgboost
- LightGBM (Ke et al., 2017). https://github.com/microsoft/LightGBM
- CatBoost (Prokhorenkova et al., 2019). https://github.com/catboost/catboost
- TabPFN v2 (Hollmann et al., 2025a). https://github.com/automl/TabPFN
- TabM (Gorishniy et al., 2024). https://github.com/yandex-research/tabm
- Real-MLP (Holzmüller et al., 2024). https://github.com/dholzmueller/realmlp-td-s_standalone
- TabR (Gorishniy et al., 2023a). https://github.com/yandex-research/tabular-dl-tabr
- ModernNCA (Ye et al., 2024). https://github.com/YyzHarry/ModernNCA
- MLP (Taud & Mas, 2017). https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- TabNet (Arik & Pfister, 2021). https://github.com/dreamquark-ai/tabnet
- DANet (Chen et al., 2022). https://github.com/QwQ2000/DANets
- ResNet (Gorishniy et al., 2021). https://github.com/yandex-research/tabular-dl-revisiting-models
- STG (Jana et al., 2023). https://github.com/runopti/stg
- VIME (Yoon et al., 2020). https://github.com/jsyoon0823/VIME
- k-NN (Guo et al., 2003). https://scikit-learn.org/stable/modules/neighbors.html
- SVM (Jakkula, 2006). https://scikit-learn.org/stable/modules/svm.html
- Linear Model (Kiebel & Holmes, 2007). https://scikit-learn.org/stable/modules/linear_model.html
- Random Forest (Rigatti, 2017). https://scikit-learn.org/stable/modules/ensemble.html#random-forests
- Decision Tree (Rokach & Maimon, 2005). https://scikit-learn.org/stable/modules/tree.html

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 4 contrasts IMLP with a standard MLP of equivalent capacity:

Table 4: 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

Table 5: 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

B.2 Layer-wise Architecture Specification

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:

$$Q = W_q \cdot x \in \mathbb{R}^{B\times 1\times 256} \quad \text{(query from current input)} \tag{9}$$

$$K = W_t \cdot H_{\text{stacked}} \in \mathbb{R}^{B\times W\times 256} \quad \text{(keys from previous features)} \tag{10}$$

$$\text{Scores} = \text{bmm}(K, Q^T) \in \mathbb{R}^{B\times W\times 1} \tag{11}$$

$$\alpha = \text{softmax}(\text{Scores.squeeze}()) \in \mathbb{R}^{B\times W} \tag{12}$$

$$\text{Context} = \text{bmm}(\alpha.\text{unsqueeze}(1), K) \in \mathbb{R}^{B\times 1\times 256} \tag{13}$$

where:

- $H_{\text{stacked}} = \text{stack}(\{h_{t-W}, \ldots, h_{t-1}\}) \in \mathbb{R}^{B\times W\times 256}$
- 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_{prev}, window size W
Ensure: Updated window H_{new}

```
h_current ← FeatureExtractor(x, Context(x))
H_new ← H_prev ∪ {h_current}
if |H_new| > W then
    H_new ← H_new[1 :]                ▷ Remove oldest feature
return H_new
```

B.3.3 Feature Normalization

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

$$\tilde{h}_i = \frac{h_i}{\|h_i\|_2 + \epsilon} \tag{14}$$

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:

$$\begin{aligned}
\text{Query projection:} &\quad \mathcal{O}(B \cdot d_{\text{in}} \cdot d_h) && (15)\\
\text{Key projection:} &\quad \mathcal{O}(B \cdot W \cdot d_h^2) && (16)\\
\text{Attention scores:} &\quad \mathcal{O}(B \cdot W \cdot d_h) && (17)\\
\text{Context aggregation:} &\quad \mathcal{O}(B \cdot W \cdot d_h) && (18)\\
\text{Feature extraction:} &\quad \mathcal{O}(B \cdot (d_{\text{in}} + d_h) \cdot 512) && (19)\\
\hline
\text{Total:} &\quad \mathcal{O}(B \cdot (d_{\text{in}} \cdot d_h + W \cdot d_h^2)) && (20)
\end{aligned}$$

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 6 compares IMLP with alternative continual learning approaches:

Table 6: 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 7: 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 Extended Results

C.1 Statistical Tests

We conducted comprehensive statistical analysis following the methodology of Demšar Demšar (2006) 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.

C.1.1 Friedman Omnibus Test Results

All statistical tests were conducted with $N = 36$ datasets and $k = 20$ classifiers at significance level $\alpha = 0.05$. The critical difference for post-hoc comparisons is

$$\text{CD} = q_{0.05}\sqrt{\frac{k(k+1)}{6N}} = 3.532\sqrt{\frac{20 \cdot 21}{6 \cdot 36}} \approx 4.92.$$

Metric	Friedman χ^2	p-value	Null Hypothesis
Balanced Accuracy	298.90	3.36×10^{-52}	Rejected
Log-Loss	430.31	2.13×10^{-79}	Rejected
NetScore-T (Balanced)	545.36	1.65×10^{-103}	Rejected
NetScore-T (Log-Loss)	395.52	3.75×10^{-72}	Rejected
Total Energy (Joules)	562.96	3.26×10^{-107}	Rejected
Total Time (Seconds)	548.94	2.91×10^{-104}	Rejected

Table 8: Friedman omnibus test results across all metrics with $k = 20$. All tests decisively reject the equal-performance null, which warrants post-hoc pairwise analysis.

C.1.2 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.

C.1.3 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 9: 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.

C.1.4 Efficiency-Adjusted Performance

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.

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 10: Dataset count where IMLP outperforms baselines on efficiency and composite metrics. IMLP dominates other neural methods but trails tree-based approaches.

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.

C.1.5 Landscape Analysis

The pairwise analysis reveals three distinct performance tiers:

1. **Accuracy Leaders**: TabPFNv2, LightGBM, MLP, and CatBoost dominate predictive metrics, with TabPFNv2 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.

C.1.6 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.

C.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: $\quad \mathrm{NS}_t^{(m)} \leftarrow P_t^{(m)} / \log_{10}(E_t^{(m)} + 1)$
4: $\quad$ scores.append($\mathrm{NS}_t^{(m)}$)
5: **return** $\frac{1}{T} \sum_{i=1}^{T} \mathrm{scores}[i]$

C.3 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.

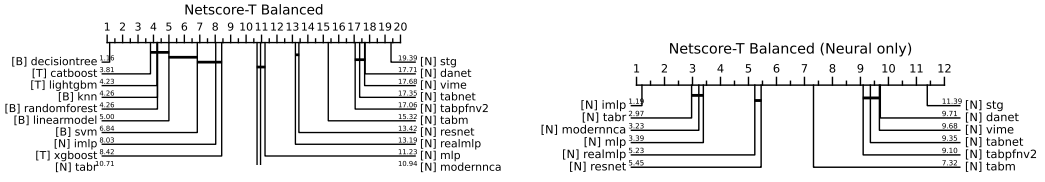
Table 11: Performance statistics across 36 TabZilla datasets. Results are reported as mean ± standard deviation across streams. Models are grouped into GBDT, Neural network-based, and Basic ones. Energy is measured in Joules, and Time in seconds.

Type	Model	Bal. Acc.	Log-Loss	NS-T (Bal.)	NS-T (Log.)	Energy (J)	Time (s)
GBDT	LightGBM	0.849 ± 0.149	0.269 ± 0.269	0.614 ± 0.226	7.251 ± 10.636	2408 ± 8293	32.1 ± 107.2
	CatBoost	0.805 ± 0.176	0.389 ± 0.350	0.611 ± 0.222	5.283 ± 8.901	2270 ± 6612	28.3 ± 78.9
	XGBoost	0.764 ± 0.177	1.207 ± 0.671	0.417 ± 0.128	0.627 ± 0.332	2091 ± 4322	13.8 ± 22.5
Basic	SVM	0.807 ± 0.170	0.345 ± 0.327	0.495 ± 0.211	4.905 ± 8.768	3720 ± 7389	84.4 ± 168.3
	k-NN	0.781 ± 0.155	1.447 ± 1.487	0.610 ± 0.219	1.232 ± 1.330	1191 ± 2958	16.3 ± 37.8
	LinearModel	0.758 ± 0.200	0.479 ± 0.497	0.568 ± 0.169	4.362 ± 6.108	297 ± 182	7.0 ± 4.4
	RandomForest	0.738 ± 0.179	0.560 ± 0.394	0.595 ± 0.159	3.404 ± 6.802	283 ± 310	5.4 ± 4.3
	DecisionTree	0.717 ± 0.189	0.693 ± 0.448	0.955 ± 0.385	2.537 ± 2.916	178 ± 447	4.6 ± 11.1
Network-based	DANet	0.812 ± 0.172	0.349 ± 0.334	0.248 ± 0.055	2.159 ± 2.596	32382 ± 26865	406.3 ± 336.4
	TabNet	0.807 ± 0.177	0.357 ± 0.337	0.250 ± 0.059	1.965 ± 2.413	23312 ± 18268	285.3 ± 226.3
	ResNet	0.805 ± 0.160	1.489 ± 1.469	0.312 ± 0.065	1.036 ± 1.334	5422 ± 3672	66.1 ± 45.1
	VIME	0.738 ± 0.197	1.098 ± 1.544	0.238 ± 0.060	0.763 ± 0.633	21705 ± 34212	261.8 ± 416.9
	STG	0.416 ± 0.166	1.162 ± 0.689	0.151 ± 0.073	0.436 ± 0.191	6747 ± 5671	85.6 ± 71.8
	TabPFN v2	0.862 ± 0.151	0.240 ± 0.265	0.252 ± 0.052	4.693 ± 5.598	72319 ± 100877	291.1 ± 405.3
	ModernNCA	0.843 ± 0.145	1.298 ± 1.344	0.346 ± 0.077	1.571 ± 2.049	4829 ± 4630	54.9 ± 53.1
	TabM	0.839 ± 0.161	0.288 ± 0.307	0.280 ± 0.056	3.370 ± 3.921	15558 ± 12321	96.6 ± 76.8
	TabR	0.836 ± 0.155	0.554 ± 0.656	0.345 ± 0.072	2.535 ± 3.027	4125 ± 3625	46.1 ± 41.2
	Real-MLP	0.823 ± 0.159	0.342 ± 0.314	0.313 ± 0.065	2.438 ± 2.656	6243 ± 4735	62.8 ± 47.4
	MLP	0.829 ± 0.162	0.329 ± 0.326	0.341 ± 0.073	2.846 ± 2.956	3241 ± 2581	41.4 ± 32.9
	IMLP (Ours)	**0.807 ± 0.164**	**0.399 ± 0.365**	**0.430 ± 0.095**	**3.169 ± 4.294**	**845 ± 386**	**9.9 ± 4.5**

C.4 Critical Difference Analysis

Figure 4 compares all $k = 20$ classifiers using NetScore-T (Balanced). Tree-based ensembles (CatBoost, LightGBM, XGBoost, RandomForest) achieve the best ranks overall. Neural networks cluster lower, though IMLP clearly outperforms the other neural baselines.

Restricting to neural methods ($k = 11$) in Figure 4 (right) confirms this: IMLP achieves the best rank, followed by TabR, ModernNCA, and MLP, while TabPFNv2, TabNet, VIME, DANet, and STG perform worst. This demonstrates IMLP's consistent advantage over alternative neural continual learners.

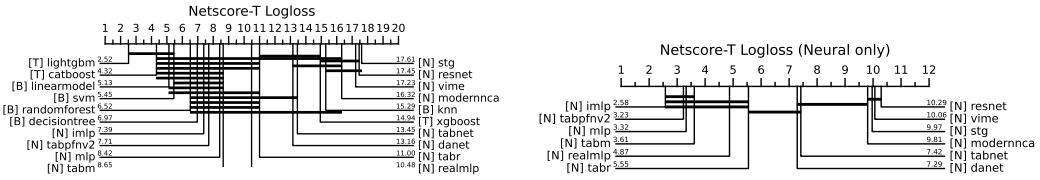


(a) All models ($N = 36$, $k = 20$). (b) Neural models only ($N = 36$, $k = 11$).

Figure 4: Critical difference diagrams for NetScore-T (Balanced). IMLP achieves the best neural rank and remains competitive when compared with all models.

Figure 5 compares all $k = 20$ classifiers using NetScore-T (Log-Loss). LightGBM and CatBoost achieve the strongest ranks overall, followed by linear baselines. Neural methods trail behind but IMLP remains the top-performing network, ranking above other deep baselines and some tree methods.

Restricting to neural methods ($k = 11$) in Figure 5 (right) highlights IMLP's clear advantage. It leads the group, with TabPFNv2, MLP, and TabM following, while RealMLP, TabR, and especially attention-based models (DANet, TabNet, ModernNCA, VIME, ResNet, STG) cluster lower. This confirms that IMLP achieves consistently lower log-loss than competing neural continual learners.



(a) All models ($N = 36$, $k = 20$). (b) Neural models only ($N = 36$, $k = 11$).

Figure 5: Critical difference diagrams for NetScore-T (Log-Loss). IMLP is the strongest neural method and remains competitive overall.

C.5 Pareto-optimal Trade-offs

To complement the critical-difference and efficiency analyses, we quantify each model's *Pareto efficiency* in the accuracy–energy plane. For model m,

$$\text{ParetoEff}(m) = \frac{\#\{\text{Pareto-optimal configs of } m\}}{\#\{\text{all configs of } m\}}.$$

Model	Total	Pareto	Efficiency (%)
imlp	67	4	5.97
tabnet	42	1	2.38
mlp	74	1	1.35
tabr	75	1	1.33
modernnca	75	1	1.33
danet	32	0	0.00
realmlp	72	0	0.00
tabm	63	0	0.00
resnet	68	0	0.00
tabpfnv2	34	0	0.00
vime	31	0	0.00

Table 12: Per-model Pareto efficiency on the accuracy–energy frontier across all datasets and seeds.

IMLP attains the highest Pareto efficiency among neural models (5.97%), followed by TabNet (2.38%), while most other networks rarely appear on the frontier. This confirms that IMLP provides the most favorable accuracy–energy trade-offs within the neural family.

C.6 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.

C.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.

C.6.2 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 13: Per-segment energy consumption. IMLP maintains constant computational cost (~ 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.

C.6.3 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 14: 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.

C.6.4 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 15: 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.

C.6.5 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.

D 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.

D.1 Shared Hyperparameter Optimization

Both MLP and IMLP models utilize identical optimized hyperparameters obtained through the comprehensive search described in Section D.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.

D.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.

D.2 External Dependencies and Platform Compatibility

D.2.1 Core Dependencies

The framework integrates with TabZilla McElfresh et al. (2023) for baseline model implementations:

```
# Install core dependencies
pip install -r requirements.txt
```

D.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

D.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.

D.3 Hyperparameter Optimization Framework

Following the methodology of Kadra et al. Kadra et al. (2021b), 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.

D.3.1 SEARCH SPACE DEFINITION

The optimization space encompasses multiple regularization families:

Implicit Regularization:

- Batch Normalization: `use_batch_norm` $\in$ {True, False}
- Stochastic Weight Averaging: `use_swa` $\in$ {True, False}

Explicit Regularization:

- Weight Decay: `use_weight_decay` $\in$ {True, False}
- Weight Decay Coefficient: `weight_decay` $\in [10^{-5}, 10^{-1}]$ (log-uniform)
- Dropout: `use_dropout` $\in$ {True, False}
- Dropout Patterns: `dropout_shape` $\in$ {funnel, long_funnel, diamond, triangle}
- Dropout Rate: `dropout_rate` $\in [0.0, 0.8]$ (uniform)

Architectural Variations:

- Skip Connections: `use_skip` $\in$ {True, False}
- Skip Types: `skip_type` $\in$ {Standard, ShakeShake, ShakeDrop}
- ShakeDrop Probability: `shakedrop_prob` $\in [0.0, 1.0]$ (uniform)

Training Techniques:

- Data Augmentation: `augmentation` $\in$ {None, MixUp}
- Augmentation Magnitude: `aug_magnitude` $\in [0.0, 1.0]$ (uniform)
- Mixed Precision: `use_amp` $\in$ {True, False}
- Gradient Clipping: `max_grad_norm` $\in [0.1, 10.0]$ (log-uniform)

D.3.2 OPTIMIZATION ALGORITHM

We employ Optuna Akiba et al. (2019) 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

D.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

D.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`

D.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.

D.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

D.4 Hardware Requirements and Energy Measurement

D.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.

D.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. (2024), 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.