Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Lightweight Anomaly-Based Intrusion Detection for IoT Networks Using Dispersion-Based Majority Class Reduction and Sparse Autoencoder Feature Extraction

Shrey Sharma
Manipal Academy of Higher Education
Vineetha Pais
A J Institute of Engineering and Technology
Balachandra Muniyal
bala.chandra@manipal.edu

Manipal Academy of Higher Education
Trivikram -
Manipal Academy of Higher Education

Research Article

Keywords: Intrusion detection, Internet of Things, sparse autoencoder, class imbalance, majority class reduction, undersampling, instance selection, lightweight deep learning, network security

Posted Date: November 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11204282/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Lightweight Anomaly-Based Intrusion Detection for IoT Networks Using Dispersion-Based Majority Class Reduction and Sparse Autoencoder Feature Extraction

Shrey sharma[1], Vineetha Pais[2], Balachandra Muniyal[1*], Trivikram[1]

[1]Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India.
[2]A J Institute of Engineering and Technology, Mangalore, India.

*Corresponding author(s). E-mail(s): bala.chandra@manipal.edu;
Contributing authors: shrey.mitmpl2022@learner.manipal.edu;
vineetha.pais@learner.manipal.edu;
trivikram.mitmpl2023@learner.manipal.edu ;

Abstract

The Internet of Things (IoT) has been a common target of network-level attacks due to its rapid growth, and anomaly-based intrusion detection powered by deep learning has emerged as a potential defense. The significant class imbalance found in IoT traffic captures and the restricted processing power on the gateway-level hardware where detection is expected to occur, are the two main challenges in this environment. The majority-reduction balancing techniques are still limited to non-IoT, non-deep learning settings and lack a tractable formulation at IoT-scale candidate-pool sizes, while autoencoder-based feature learning is almost exclusively paired with synthetic minority oversampling. This gap is filled by a collection of classifier heads of decreasing size are fed by the frozen encoder of a sparse autoencoder (SAE), which is trained without labels to develop a compact feature representation. Dispersion-based Majority Class Reduction (DMCR), which scores exactly via an order-statistics identity in $\mathcal{O}(nd \log n)$ time instead of the $\mathcal{O}(n^2)$ required by an all-pairs reading, achieves class balance instead of synthetic oversampling by keeping the most mutually dissimilar real majority samples instead of creating minority ones. Four configurations are compared using the BoT-IoT (5%) dataset. The balancing comparison is further validated

against a classical random-undersampling baseline, against Cosine Similarity-based Majority Class Reduction (CSMCR), and against two synthetic-oversampling variants, ADASYN and Borderline-SMOTE, with every method, DMCR included, evaluated across multiple random seeds. A separate sensitivity analysis further shows that DMCR's stability depends on retaining a sufficiently filtered subset of its candidate pool: collapsing the pool and the retention target together reproduces a complete detection failure on the minority class in one seed out of three, a failure an aggregate F1 score does not reveal. Using DMCR instead of synthetic oversampling reduced training time by roughly one to two orders of magnitude while maintaining detection quality on a set that was about 116× smaller and contained no fake data. A deeper, batch-normalized variant restored full ranking quality at less than a fifth of the baseline's size, while a compact MLP matched the convolutional baseline's zero-false-negative coverage at about a ninth of its parameter count. Results are evaluated with specific caution because of BoT-IoT's stereotypical, lab-generated traffic.

Keywords: Intrusion detection, Internet of Things, sparse autoencoder, class imbalance, majority class reduction, undersampling, instance selection, lightweight deep learning, network security

1 Introduction

The Internet of Things (IoT) already includes sensors, cameras, home appliances, and industrial controls that share data over the Internet with little to no human oversight. The population of these devices is expected to reach tens of billions in few years [1]. IoT devices are vulnerable to widespread exploitation and botnet recruitment due to their connectivity. Many are shipped with default or weak credentials, operate on limited memory and compute budgets, and receive infrequent firmware updates [1, 2].

Lying within the network path, an intrusion detection system (IDS) notifies users of any suspicious activity. There are two general paradigms for detection. Signature-based detection compares observed traffic to a database of known attack patterns. It is quick and accurate for threats that have already been cataloged, but it is unable to identify new ones and creates necessity for ongoing database maintenance [2]. Instead, at the expense of a higher false-positive rate, anomaly-based detection creates a typical behavior model and highlights deviations from it, allowing it to identify threats that have never been seen before [1]. Anomaly-based detection has become the more actively pursued strategy for IoT-specific IDS design due to the rapid change of the IoT threat landscape changes new variants of botnets, scanning techniques, and patterns of denial-of-service develop constantly [1, 2].

Two main issues repeat across this literature and motivate the present work directly. The first is class imbalance. Benign and malicious traffic are rarely balanced in real network captures. Depending on how a dataset is gathered, one class may dominate the other by several orders of magnitude. Regardless of the minority class, a classifier trained naively on such data minimizes total error by favoring the majority class, which results in poor identification of the minority class [3]. Synthetic minority oversampling

[4], interpolates new minority-class points along the line segments joining existing neighbors and its numerous offspring and hybrids, such as feature-weighted SMOTE combined with convolutional recurrent architectures [5], are the predominant remedy in the IDS literature. When the minority pool consists of only a few hundred real samples, as is the case for the dataset used here, oversampling to parity results in millions of synthetic points, all of which the model must then learn from as if they were actual observations [3, 4]. Oversampling is efficient, but it creates data. Instead, a different and relatively unexplored family reduces the majority class by eliminating unnecessary majority instances while attempting to maintain the decision boundary [6]. It has been demonstrated that undersampling is competitive with oversampling at a fraction of the computational cost as long as the discarded information is compensated for by principled instance selection rather than uniform random deletion [7].

The second obstacle is the resource budget available at the point of deployment. Compared to the data-center hardware commonly used to train the underlying models, the routers, gateways, and single-board computers on which an IoT IDS is anticipated to operate have orders of magnitude less computation and memory [1]. In practice, a detector that is accurate in a benchmark setting but too big or slow to operate at the edge is not deployable. Therefore, a workable IoT IDS needs to be light enough to accommodate gateway-class hardware and balanced against imbalance without ballooning the training set needlessly.

Feature learning with autoencoders is a well-known approach to some of this problem. An autoencoder taught to reconstruct its own input through a tiny bottleneck is compelled to encode the input compactly in order to separate representation learning from decision-making. After that, the encoder can be used again as a fixed feature extractor of the downstream classifier [8]. Rao et al. [8] employ this encode-then-classify pattern with a sparsity-regularized autoencoder feeding a deep neural network, reporting strong performance on KDDCup99, NSL-KDD, and UNSW-NB15, while without addressing IoT-specific traffic or edge-resource constraints. Nearer to the present, Alsoufi et al. [9] integrate a sparse autoencoder (SAE) with a convolutional neural network (CNN) classifier for IoT using a moderately strong convolutional head and synthetic oversampling.

Beyond these two threads, the broader IoT-IDS literature illustrates a variety of deep learning designs without converging on a lightweight, edge-ready solution. Bhavsar et al. [10] combine correlation-based feature selection with a CNN classifier across NSL-KDD, CIC-IDS-2017, and IoTID20. Hizal et al. [11] propose a two-stage deep learning detector on the CICIoT2023 dataset and explicitly measure per-packet inference time. Vishwakarma and Kesswani [12] show that a comparatively simple two-phase scheme, pairing a Naïve Bayes ensemble with an unsupervised outlier detector, remains competitive, suggesting that detection quality need not scale with model complexity. All experiments in this paper use the BoT-IoT dataset [13], a realistic IoT botnet capture designed for network forensic analysis and widely used as a benchmark in this space.

This work retains the autoencoder-based feature extraction of the SAE-CNN baseline [9] but reconsiders both of its remaining design choices. First, class imbalance is checked if it can be handled by reducing the majority class through dispersion

criterion. Real majority samples are retained that are maximally distinct from one another rather than fabricating minority samples, instead of relying on synthetic oversampling. Second, once the encoder has learned an informative representation, a substantially smaller classifier head can match the convolutional baseline's detection quality is checked. Answering the second question well requires answering it *twice*, since a smaller model may trade one error type for another. Therefore false positives and false negatives are reported separately throughout, rather than collapsing them into a single accuracy figure, since a missed attack and a false alarm carry different operational costs in a deployed IDS.

The contributions of this paper are:

- Dispersion-based Majority Class Reduction (DMCR), a balancing scheme that discards redundant majority-class samples while retaining every genuine minority sample, producing a training set roughly 116× smaller than synthetic oversampling and containing no fabricated data.
- A comparison of DMCR against four other balancing baselines, synthetic oversampling (SMOTE and two of its variants, ADASYN and Borderline-SMOTE), Cosine Similarity-based Majority Class Reduction (CSMCR), and naive random undersampling, with every method, DMCR included, evaluated across multiple random seeds to establish statistical robustness.
- A sensitivity analysis of DMCR's retention target that identifies a previously uncharacterized failure mode: collapsing the candidate pool and the retention target together removes the benefit of dispersion-based selection and produces a complete misclassification of the minority class in one of three seeds, a failure that the F1 metric alone fails to expose.
- An order-statistics formulation of the dispersion criterion that computes it exactly in $\mathcal{O}(nd \log n)$ time and $\mathcal{O}(n)$ memory, avoiding the $\mathcal{O}(n^2)$ pairwise-distance computation that would otherwise make scoring a 100,000-sample candidate pool intractable.
- A controlled comparison of three classifier heads (CNN, MLP, MLP2), on top of a common frozen encoder, isolating the individual effects of the balancing strategy and the classifier size on detection performance.
- The identification of two complementary lightweight operating points zero false negatives and zero false positives, suited to different IoT deployment postures, reported alongside an explicit account of the dataset's limitations and the study's scope.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 collects, in one place, the formal notation and every equation used by the pipeline, with the reasoning behind each. Section 4 then describes, in prose, how the dataset is prepared and how each equation from Section 3 is applied within the end-to-end system. Section 5 presents the experimental results across six balancing strategies, a DMCR target-size sensitivity ablation, and four classifier-head configurations, and discusses the effect of the balancing strategy, the effect of retention-target size, the effect of classifier size, the two operating points this yields, and the study's limitations. Section 6 concludes and outlines directions for future work.

2 Related Work

This section situates the proposed pipeline against four bodies of prior work: autoencoder-based feature learning for intrusion detection, strategies for handling class imbalance, deep-learning architectures applied specifically to IoT traffic, and the specific training techniques (sparsity regularization, batch normalization) that the proposed architecture relies on. Table 1 summarizes the studies most directly relevant to this work, the subsections that follow group them thematically and state, for each, what it establishes and what it leaves open.

2.1 Autoencoder-based feature learning

The reference architecture under consideration here is preceded by the practice of training an autoencoder without labels and reusing its encoder as a fixed feature extractor for a different classifier. Rao et al. [8] use a sparsity-regularized autoencoder before a deep neural network and show good results on three well-known intrusion datasets, but they don't take edge hardware resource restrictions or IoT-specific traffic patterns into account. The same encode-then-classify pattern is applied specifically to IoT traffic by Alsoufi et al. [9], who link a sparse autoencoder with a convolutional classifier. This study reproduces and reconsiders this baseline SAE-CNN approach. Both studies rely on the Kullback Leibler sparsity formulation introduced by Ng [14], which this paper also adopts for its autoencoder, while additionally documenting a boundary condition, the clamped activation range required to keep the KL term well-defined that neither prior study reports explicitly.

2.2 Handling class imbalance

He and Garcia [3] have thoroughly cataloged the long-studied issue of imbalanced learning. The reference architecture [9] and more recent hybrid detectors like Qaddos et al. [5] continue to employ synthetic minority oversampling (SMOTE), which was first introduced as SMOTE by Chawla et al. [4]. The majority class is instead reduced by a smaller, complementary body of work: Liu et al. [7] show that undersampling ensembles can match oversampling's detection quality at a fraction of its computational cost, provided the discarded information is compensated for by principled selection rather than uniform random deletion. Wilson and Martinez [6] survey instance-reduction algorithms that discard redundant exemplars while attempting to preserve the decision boundary. DMCR belongs to this second, less-explored family, and unlike [6, 7], is formulated to operate efficiently on top of a learned, high-dimensional representation at IoT-scale candidate-pool sizes (Section 4.4). A more directly comparable majority-reduction technique is Cosine Similarity-based Majority Class Reduction (CSMCR) [16], which scores each majority candidate by its mean cosine similarity to the rest of the candidate pool and discards the most redundant, i.e. most similar samples, retaining those that are directionally distinct from the bulk of the class.

Table 1: Summary of the literature survey

Sl.	Ref, Year	Methodology	Relevance to this work	Gap relative to this study
1	[3], 2009	Survey and taxonomy of learning from imbalanced data; catalogues oversampling, undersampling, and cost-sensitive approaches.	Establishes the theoretical basis for why naively trained classifiers favour the majority class.	Predates deep learning; no network traffic or extreme imbalance.
2	[4], 2002	SMOTE: generates synthetic minority samples by interpolating between real neighbours.	Foundational oversampling technique; used as the balancing method in the baseline this paper reproduces.	Fabricates data; scales poorly with very small minority seeds.
3	[6], 2000	Survey and extension of instance-reduction algorithms that discard redundant exemplars while preserving the decision boundary.	Establishes the instance-selection lineage that DMCR belongs to.	Pre-deep-learning; not evaluated on intrusion data.
4	[7], 2009	Exploratory undersampling ensembles (e.g., EasyEnsemble); shows undersampling can match oversampling at lower cost.	Direct precedent for majority reduction as an alternative to minority inflation.	Tabular ML only; no learned representation or IoT traffic.
5	[8], 2021	Sparse autoencoder with smoothed regularisation feeding a deep neural network; evaluated on KDDCup99, NSL-KDD, UNSW-NB15.	Validates the encode-then-classify pattern used throughout this paper.	No IoT traffic, imbalance handling, or edge-resource focus.
6	[9], 2024	Sparse autoencoder (SAE) feature extraction feeding a CNN classifier for IoT anomaly detection; balances with synthetic oversampling.	Reference architecture this paper reproduces (Experiment 1) and reconsiders.	Uses synthetic oversampling and a heavy CNN head; neither is ablated.
7	[5], 2024	CNN-GRU hybrid with feature-weighted SMOTE; evaluated on IoTID20 and UNSW-NB15.	Represents the synthetic-oversampling route this paper avoids.	Two heavy architectures; no non-synthetic or lightweight comparison.
8	[10], 2023	Correlation-based feature selection with a CNN classifier; evaluated on NSL-KDD, CIC-IDS-2017, IoTID20.	Comparable CNN-based IoT detector validated across datasets.	No imbalance-handling comparison or parameter-count ablation.
9	[11], 2024	Two-stage deep-learning detector on CICIoT2023 with explicit per-packet inference-time measurement.	Motivates this paper's efficiency and inference-time focus.	No classifier miniaturisation or majority reduction explored.
10	[12], 2023	Two-phase Naïve Bayes ensemble with unsupervised outlier detection for suspicious-flow triage.	Shows simple classifiers stay competitive, supporting this paper's shrinking direction.	No learned representation or complementary error-rate reporting.
11	[13], 2019	Introduces the BoT-IoT dataset: a realistic, laboratory-generated IoT botnet capture.	Source of the dataset used in all experiments here.	Dataset only; stereotyped attacks and tiny normal pool (477/3.67M) limit conclusions.
12	[14], 2011	Formalises the sparse autoencoder via a KL penalty between target and observed hidden-unit activation.	Theoretical basis for the SAE component of the pipeline.	Generic formulation; ignores bounded-decoder vs. unbounded-target mismatch.
13	[15], 2015	Introduces batch normalisation to stabilise and accelerate deep network training.	Adopted in MLP2 to recover calibration lost by the single-layer MLP.	Its role in calibrating an undersized head is uncharacterised in prior IDS work.

2.3 Deep learning for IoT intrusion detection

Beyond the balancing question, the broader IoT-IDS literature illustrates a range of deep-learning designs without converging on a lightweight solution. Bhavsar et al. [10] combine correlation-based feature selection with a CNN across three benchmark datasets. Hizal et al. [11] design a two-stage detector and, notably, measure per-packet inference time on CICIoT2023, one of the few studies in this space to treat latency as a first-class metric and Vishwakarma and Kesswani [12] show that a comparatively simple two-phase scheme (a Naïve Bayes ensemble followed by unsupervised outlier detection) remains competitive, suggesting that detection quality need not scale with architectural complexity. BoT-IoT, introduced by Koroniotis et al. [13] and used throughout this paper, remains the standard reference dataset for this line of work despite its laboratory-generated, stereotyped attack traffic.

2.4 Training techniques adopted

Two general-purpose deep learning techniques are adopted directly in the proposed architecture rather than developed anew: the KL-divergence sparsity penalty of Ng [14], used to shape the autoencoder's hidden representation, and batch normalization, introduced by Ioffe and Szegedy [15] to reduce internal covariate shift during training. The latter is repurposed here for a narrower and more specific role than its original motivation: rather than accelerating convergence on a large network, it is used to restore probability calibration in a classifier head that has been deliberately shrunk to a few thousand parameters (Section 4.6), a use case not previously characterized in the IoT-IDS literature.

Examining the literature, four gaps jointly motivate this paper:

1. Majority-reduction studies target general ML, not learned IoT representations, no work pairs a frozen encoder with majority-reduction balancing.
2. Prior instance-selection methods lack a tractable formulation for the multi-million-record, extremely imbalanced pools typical of IoT captures.
3. Most IoT-IDS studies report one architecture-balancing combination, without isolating representation, balancing, and classifier contributions.
4. The FP/FN cost asymmetry is rarely made an explicit design lever, and complementary operating points are seldom offered within one lightweight pipeline.

3 Mathematical Formulation

This section formalizes the detection pipeline, from problem notation through DMCR balancing, the sparse autoencoder, and classifier heads used in the study.

3.1 Detection problem and notation

Denote the dataset $D = \{(x^{(k)}, y^{(k)})\}_{k=1}^{n}$, where each $x^{(k)} \in \mathbb{R}^d$ is a feature vector describing one network flow and $y^{(k)} \in \{0, 1\}$ its label, with 0 denoting normal traffic and 1 an attack. After feature selection (Section 4.3), the working dimensionality is $d = 27$. The goal is a detector $g_\theta : \mathbb{R}^d \to [0, 1]$ that outputs an estimated attack

probability $\hat{y} = g_\theta(x)$, thresholded into a hard decision at $\tau = 0.5$, applied identically to every classifier head in Section 4.6 so that no head is given a threshold advantage over another. The detector itself factorises into two independently trained parts, $g_\theta = c_\phi \circ \text{enc}_\psi$: a frozen encoder enc_ψ (Section 3.5) that maps raw features to a fixed representation, and an interchangeable classifier head c_ϕ (Section 3.6) trained on top of that representation. This factorisation is what allows the same encoder to be reused, unmodified, across all three classifier heads compared in Section 5.

3.2 Class imbalance notation

Writing n_0 and n_1 for the normal and attack sample counts in the raw dataset respectively, $n_0 = 477$, $n_1 = 3{,}668{,}045$, giving an imbalance ratio $n_1/n_0 \approx 7{,}690$. This ratio is the single number that motivates every balancing decision made in this paper: it is large enough that neither synthetic oversampling nor majority reduction can be applied naively (Section 4.4), and it fixes the direction of the imbalance, attacks are the majority class here, the reverse of what is typical in network captures which is why it is the normal class that must be protected while the majority is trimmed down.

3.3 Feature selection and standardisation

Each candidate feature f is scored against the target by absolute Pearson correlation and retained iff $|r_{f,y}| > r_{\min}$, with $r_{\min} = 0.01$; this threshold yields the $d = 27$ features used throughout (applied as described in Section 4.3). Every retained feature is subsequently standardized,

$$z_i = \frac{x_i - \mu_i}{s_i} \tag{1}$$

with μ_i and s_i the mean and standard deviation of feature i, estimated on the balanced training partition (Section 4.4) rather than the raw one, so that scaling reflects the distribution the model actually trains on; the same μ_i, s_i are then applied, unchanged, to the test partition. Equation (1) produces the standardized vector z that is the direct input to the sparse autoencoder of Section 3.4.

3.4 DMCR balancing formulation

Dispersion-based Majority Class Reduction (DMCR) balances the training set by keeping a diverse subset of the majority class rather than fabricating minority samples. Given a candidate pool P of size $n = |P|$ drawn from the majority class (Section 4.4 describes how this pool is formed), each pool value is first rescaled by its own feature's range within the pool,

$$\tilde{x}_{i,c} = \frac{x_{i,c}}{\max_{j \in P} x_{j,c} - \min_{j \in P} x_{j,c}} \tag{2}$$

with the divisor replaced by 1 wherever a feature has zero range within the pool. This range normalization exists purely to make distances comparable across features of very different scales (e.g., byte counts versus per-flow rates), it is internal to the DMCR criterion and independent of the standardization of (1).

Each range-normalized candidate is then scored by its mean ℓ_1 (Manhattan) distance to every other member of the pool:

$$D(x_i) = \frac{1}{nd} \sum_{x_j \in P} \sum_{c=1}^{d} |\tilde{x}_{i,c} - \tilde{x}_{j,c}| \tag{3}$$

Points in dense, repetitive regions of the pool have many near neighbours and therefore a low mean distance to the rest of the pool, while comparatively isolated points whose information is not duplicated elsewhere have a high one. DMCR keeps the latter and discards the former.

Taken literally, (3) requires all n^2 pairwise distances ten billion evaluations at $n = 100{,}000$. This is not actually necessary, because the ℓ_1 distance decomposes across features: fixing a single feature c and sorting its pool values ascending, $v_{(0)} \le v_{(1)} \le \cdots \le v_{(n-1)}$, with prefix sums $C_p = \sum_{q=0}^{p} v_{(q)}$ and $C_{-1} = 0$, the sum telescopes into

$$\sum_{q=0}^{n-1} \left| v_{(p)} - v_{(q)} \right| = v_{(p)}(2p - n + 1) - C_{p-1} + (C_{n-1} - C_p) \tag{4}$$

Equation (4) is exact, not an approximation. Mapping each feature's per-position results back through that feature's own sorting permutation and summing across all d features recovers (3) exactly, at a cost of

$$\underbrace{O(nd \log n)}_{\text{time}}, \quad \underbrace{O(n)}_{\text{memory}} \tag{5}$$

in place of $O(nd)$ time and $O(n)$ memory for the literal reading of (3)—the only reason a candidate pool of $n = 10^5$ is computationally tractable at all.

Once every candidate is scored and ranked by decreasing dispersion, the N_{maj} highest-scoring candidates are kept and combined with every minority sample in the training partition D_{tr}:

$$S = \{x_{(1)}, \ldots, x_{(N_{\text{maj}})}\} \cup \{x \in D_{\text{tr}} : y = 0\} \tag{6}$$

giving a balanced training set of size $|S| = N_{\text{maj}} + n_0^{\text{tr}}$. The specific values of N_{maj} and n_0^{tr} used in this study, and the resulting size of S, are given in Section 4.4.

3.5 Sparse autoencoder formulation

Given a standardized input $z \in \mathbb{R}^{27}$ from (1), the sparse autoencoder's encoder and decoder are

$$h = \text{ReLU}(W_e z + b_e), \quad h \in \mathbb{R}^{100} \tag{7}$$

$$\hat{z} = \sigma(W_d h + b_d), \quad \hat{z} \in \mathbb{R}^{27} \tag{8}$$

where $\sigma(t) = (1 + e^{-t})^{-1}$ is the logistic sigmoid applied element-wise. The encoded vector h from (7) is the fixed 100-dimensional representation that every classifier head in Section 3.6 consumes, (8) exists only to train the encoder and is discarded afterward.

Training minimizes reconstruction error plus a sparsity penalty. The reconstruction term is mean squared error over a minibatch of size m:

$$L_{\text{rec}} = \frac{1}{m} \sum_{k=1}^{m} \left\| z^{(k)} - \hat{z}^{(k)} \right\|_2^2 \tag{9}$$

A structural mismatch shapes how (9) should be read: the decoder's sigmoid output in (8) is confined to $(0, 1)$, but the reconstruction target z is a standardized score centred at zero, unbounded, roughly half-negative, and prone to large positive outliers from heavy-tailed features. The decoder cannot span the full range of its own target, so (9) is bounded away from zero by construction, regardless of training duration. This is accepted deliberately: the autoencoder's purpose is to shape the encoder through the combined objective below, not to reconstruct faithfully (Section 4.5 discusses this further).

Let $\hat{\rho}_j$ denote hidden unit j's mean activation over a minibatch,

$$\hat{\rho}_j = \frac{1}{m} \sum_{k=1}^{m} h_j^{(k)} \tag{10}$$

and let ρ be a small target activation. Following Ng's sparse-autoencoder formulation [14], the sparsity penalty sums the Kullback–Leibler divergence between target and observed activation across all hidden units:

$$\text{KL}(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j} \tag{11}$$

so that, with penalty weight β, the total training objective is

$$L_{\text{SAE}} = L_{\text{rec}} + \beta \sum_{j=1}^{100} \text{KL}(\rho \| \hat{\rho}_j) \tag{12}$$

Equation (11) is only defined for $\hat{\rho}_j \in (0, 1)$, but the ReLU encoder of (7) has no upper bound, so $\hat{\rho}_j$ can legitimately exceed 1, Section 4.5 explains how this is handled in practice.

3.6 Classifier head formulations

Every classifier head c_ϕ operates on the frozen encoding $h \in \mathbb{R}^{100}$ from (7) and is trained with binary cross-entropy loss over a minibatch of size m.

The convolutional (CNN) head applies a 1-D convolution with kernel k to each of its input channels $u_c \in \mathbb{R}^{32}$:

$$(u_c * k)_t = \sum_i u_{c,t+i} k_i \tag{13}$$

which, following standard deep-learning-framework convention, is cross-correlation rather than a strictly reflected convolution. Section 4.6 gives the full layer-by-layer CNN architecture built around (13).

The single-hidden-layer MLP head computes

$$a = \mathrm{ReLU}(W_1 h + b_1), \quad a \in \mathbb{R}^{32}, \tag{14}$$

$$\hat{y} = \sigma\big(w_2^\top \mathrm{drop}(a) + b_2\big), \tag{15}$$

where drop$(\cdot)$ denotes dropout regularisation.

The deeper, batch-normalised MLP2 head extends (14)–(15) with a second hidden layer and, for a pre-activation a_i within a minibatch of mean μ_B and variance σ_B^2, inserts batch normalisation before each nonlinearity:

$$\mathrm{BN}(a_i) = \gamma \frac{a_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta_{\mathrm{bn}} \tag{16}$$

with learnable scale γ and shift β_{bn}, and a small constant ϵ for numerical stability. Section 4.6 explains why (16) was introduced specifically for this head and what problem it fixes.

4 Proposed Methodology

This section describes how the detection pipeline of Section 3 was actually built, from raw traffic captures to a trained, evaluable model. Fig. 1 sketches the data flow from the raw CSVs through to a balanced, standardized training set while Fig. 2 sketches the model-side flow, showing how that training set feeds an autoencoder whose encoder is subsequently frozen and reused across classifiers. Throughout, equation numbers refer back to Section 3.

4.1 Dataset

The BoT-IoT dataset [13], in particular the 5% subset made available as four CSV files, is the basis for all investigations. When combined, these records total $n = 3{,}668{,}522$ across 46 raw columns. They were gathered from a controlled testbed and encompass botnet behaviors like information-theft traffic, distributed denial-of-service, denial-of-service, and reconnaissance. The dataset is unique and challenging because of the direction of its imbalance: the attack class predominates by about 7,690 to 1, which is the opposite of what is typically observed in network captures where benign traffic makes up the vast majority. This distribution is displayed in Fig. 3. The main balancing

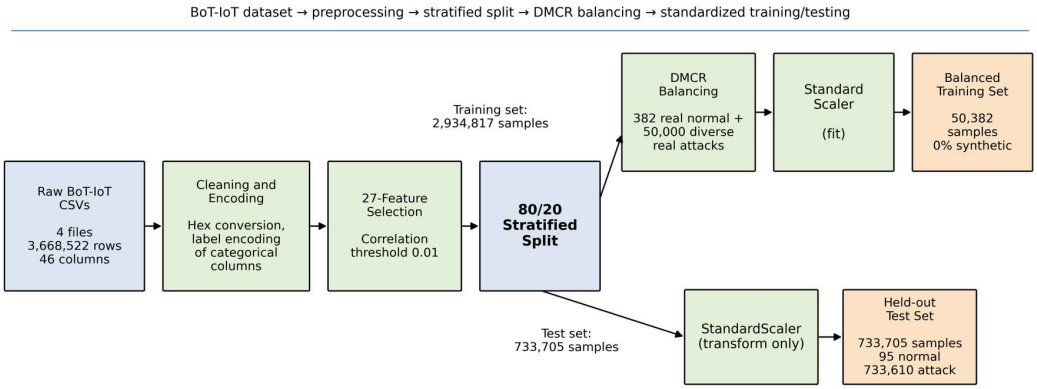


Fig. 1: Data flow of the proposed pipeline: the four raw BoT-IoT CSV files are merged and cleaned, categorical fields are encoded, 27 features are retained by correlation-based selection, an 80:20 stratified split is applied, and DMCR balances the training partition using real samples only.

issue this paper addresses is protecting normal traffic rather than attacks while reducing the majority, since normal traffic is the scarce class in this situation.

4.2 Assumptions

Everything that follows is based on four assumptions. As ground truth is taken that BoT-IoT designations at face value. Instead of rederiving results on the entire capture, the 5% selection was taken as indicative of the entire dataset. Multi-class attack classification was set aside and limited the problem to binary detection (normal versus attack). Additionally, any time provided in this study should be interpreted as a relative comparison across the experiments conducted on the same machine rather than as an absolute benchmark that can be applied to other systems, since wall-clock timings are dependent on the hardware used to measure them.

4.3 Preprocessing

The four raw files are combined into a single table and cleaned prior to other procedures. One column stores the destination port as hexadecimal text sprinkled with sentinel placeholder values, this is converted to plain integers. Two other columns, address and connection state are categorical and are label-encoded into numeric form so the network can consume them. Finally, columns are dropped carrying no discriminative signal for detection, the per-record sequence identifier and the raw start/end timestamps.

Feature selection. The $d = 27$ features used throughout are obtained by scoring each candidate feature using the absolute Pearson correlation and keeping it only above the threshold $r_{\min} = 0.01$. The held-out fifth of the data technically had a say in which 27 features were retained, even though it had no bearing on fitting any statistic used for

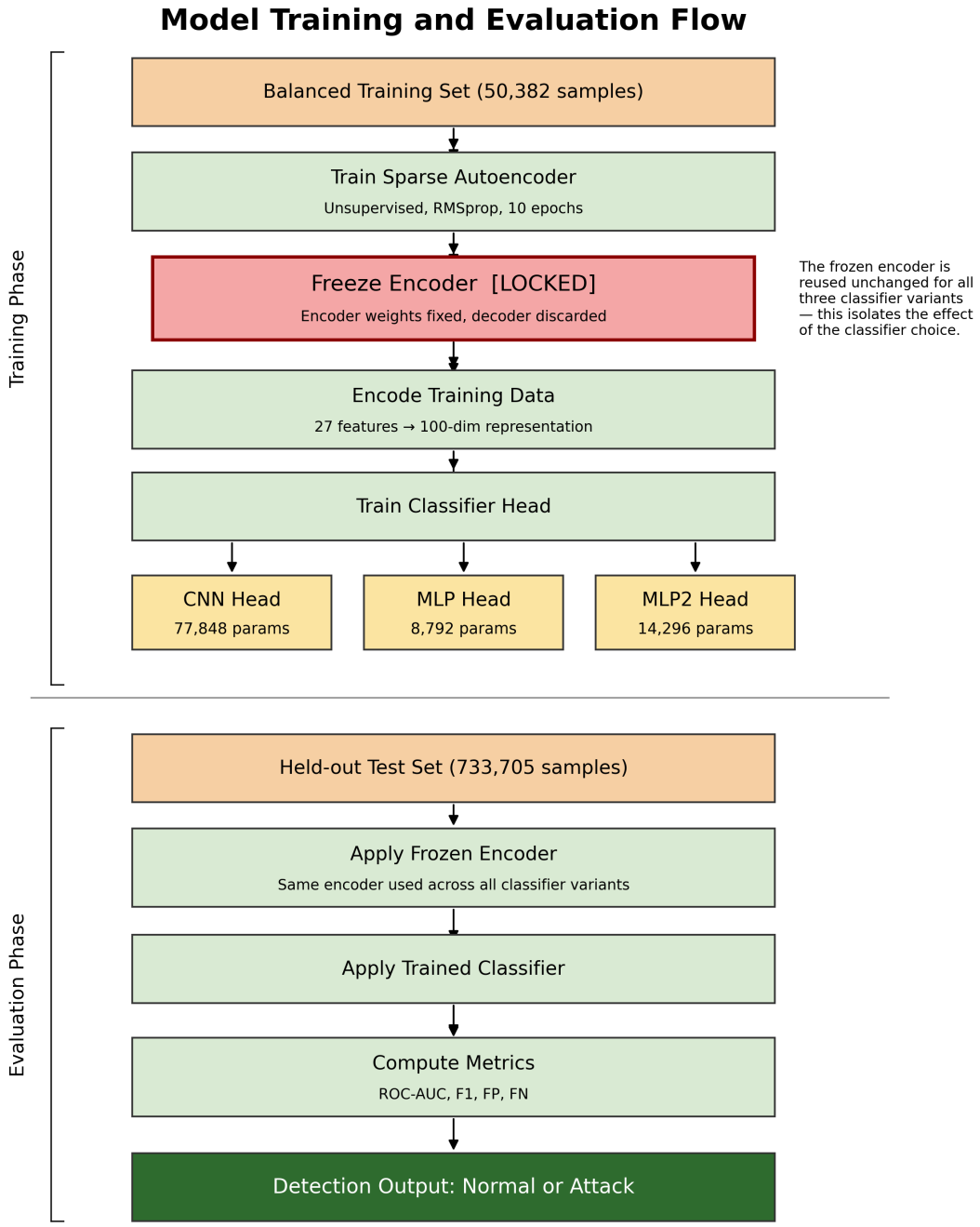


Fig. 2: Model flow of the proposed pipeline: a sparse autoencoder is trained unsupervised on the balanced data and its encoder is then frozen; an interchangeable classifier head (CNN, MLP or MLP2) consumes the frozen 100-dimensional encoding and produces the normal/attack decision, evaluated on the held-out test partition.

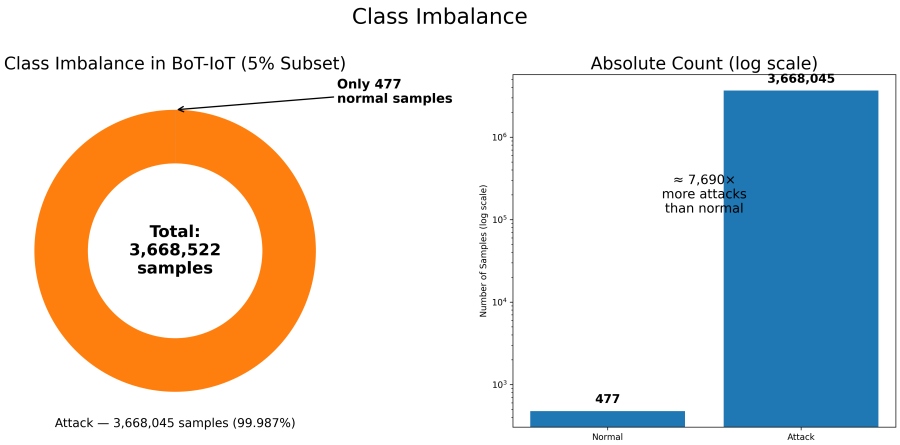


Fig. 3: Class distribution of the BoT-IoT (5%) dataset. Normal traffic accounts for only 477 of 3,668,522 records, an imbalance ratio of approximately 7,690:1 in favour of attack traffic.

scaling or training any model. This is because this correlation is calculated across all n records *before* the train/test split is drawn. This is a minor case of selection leaking.

Splitting and scaling. 2,934,817 training records (382 of which are normal) and 733,705 test records (95 normal) remain after an 80:20 stratified split maintains the imbalance ratio on both sides. After that, each retained feature is standardized according to (1), with μ_i and s_i fitted on the *balanced* training partition (Section 4.4) purposefully after balancing rather than before. This ensures that scaling reflects the data distribution the model will actually train on and is applied unaltered to the test partition.

4.4 Balancing with Dispersion-based Majority Class Reduction (DMCR)

Instead of manufacturing new normal samples to match the attack majority, the majority is shrunk. The reasoning is straightforward: a majority class nearly ten thousand times larger than the minority is bound to carry enormous redundancy, so if a well-chosen subset that stands in for the whole without losing its diversity is identified, much smaller training set is got for free. DMCR does exactly this, using the dispersion criterion of (3) to keep the majority samples that sit furthest, on average, from everything else in a candidate pool, on the logic that points buried in dense, repetitive regions are the ones most safely discarded, while comparatively isolated points carry information the rest of the pool doesn't already encode.

Candidate pool. Since none of the 2,934,435 training attacks are exact duplicates, the majority partition is de-duplicated on exact feature-value equality, which proves to be a no-op for this dataset. Nevertheless, one is kept because correctness on other datasets, where duplicates do occur, depends on it. Draw $n = |\mathcal{P}| = 100{,}000$ candidates

uniformly at random, without replacement, under three seeds, from the de-duplicated pool.

Range normalization and scoring; Every pool value is rescaled per (2) so that features on very different scales, for instance, byte counts versus per-flow rates.

Selection With the pool ranked by decreasing dispersion, the $N_{\text{maj}} = 50{,}000$ highest-scoring candidates are kept and combined with all $n_0^{\text{tr}} = 382$ minority samples, giving a balanced training set of size $|\mathcal{S}| = 50{,}382$, roughly 116 times smaller than the synthetically oversampled alternative (≈5,868,870 samples) and, crucially, built entirely from real observations. The set is shuffled under a fixed seed, and a 90:10 stratified split within it supports early stopping. Fig. 4 contrasts the two balancing regimes directly.

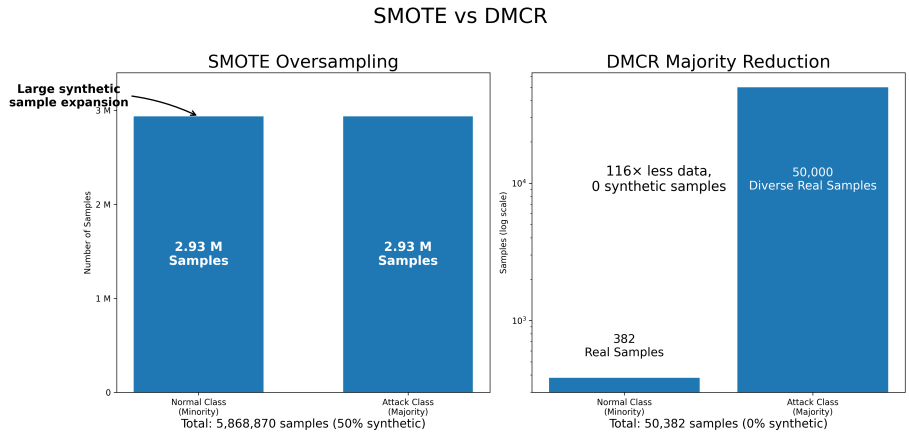


Fig. 4: Training-set composition under synthetic oversampling (SMOTE) versus DMCR majority reduction. SMOTE inflates the minority class with synthetic points to ∼5.87M total samples; DMCR retains real samples only, for a total of 50,382 (116× smaller, 0 synthetic).

One property of this procedure deserves to be stated outright rather than left implicit. The reduction from 2,934,435 attacks down to 50,000 happens in two stages: a uniform random draw first narrows the field to a 100,000-candidate pool, and only within that pool does dispersion-based selection make the final 2:1 cut. The dispersion criterion of (3) therefore governs only the last step, the bulk of the reduction is achieved by random sampling, and each candidate is ranked against the pool rather than against the full 2.9-million-sample majority. Consequently, the comparison reported in Section 5 isolates DMCR against synthetic oversampling, it does not isolate the dispersion criterion against a same-sized uniform random subsample, since that second ablation was not run. What our results do support is that a 50,382-sample subset built this way preserves the baseline's detection quality at a small fraction of its training cost; attributing that specifically to dispersion-based ranking, rather than to sample-size reduction alone, is left as future work (Section 6).

4.5 Sparse autoencoder

Feature extraction relies on the unsupervised sparse autoencoder of (7) (12). The model is trained with $\rho = 0.3$, $\beta = 1.0$, RMSprop at a small learning rate and a large batch size, for a handful of epochs with early stopping, and totals 5,527 parameters.

Two implementation details are worth stating explicitly, since neither is visible from the loss formulation alone. First, the decoder's sigmoid output in (8) cannot span the full range of the standardized reconstruction target, so the reconstruction loss is bounded away from zero by construction regardless of training duration. This is accepted deliberately, since the autoencoder's role here is to shape the encoder, not to reconstruct faithfully, and no downstream result depends on reconstruction fidelity. Second, because the ReLU encoder of (7) is unbounded above, the mean activation $\hat{\rho}_j$ in (10) can legitimately exceed 1, which would leave the KL term of (11) undefined. This is handled by clamping $\hat{\rho}_j$ to $[10^{-8}, 1 - 10^{-8}]$ before evaluating (11). One consequence follows directly: for any hidden unit whose true mean activation climbs past 1, the sparsity penalty saturates at the value it takes at the clamp boundary, so such a unit is no longer penalized any more heavily than one sitting just under the boundary, the sparsity pressure reported here is therefore a lower bound on what an unclamped formulation would impose, and $\rho = 0.3$ should be read as a target for the *clamped* quantity rather than the raw activation.

Once trained, the encoder $\mathrm{enc}_\psi(\boldsymbol{z}) = \boldsymbol{h}$ from (7) is frozen and reused as a fixed feature extractor by every classifier head below; only $\boldsymbol{h}$ ever passes downstream, and the decoder of (8) is discarded. Fig. 5 shows the architecture.

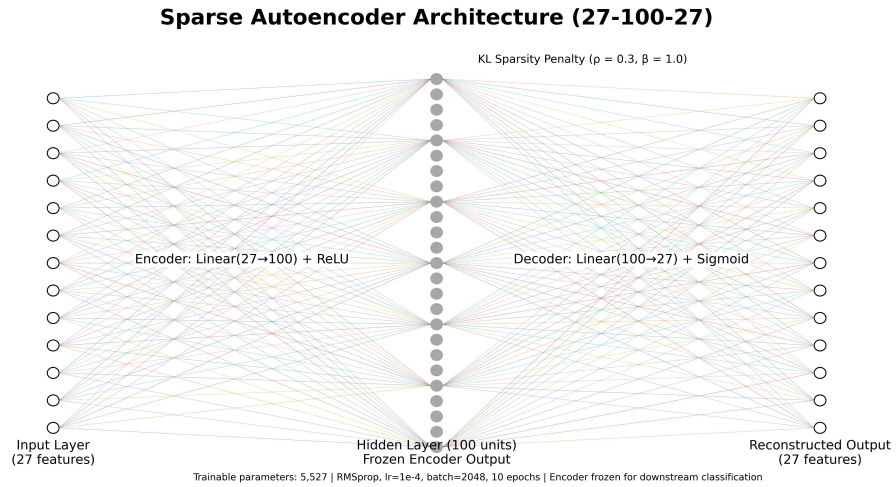


Fig. 5: Sparse autoencoder architecture (27–100–27). The encoder is frozen after unsupervised training and reused for all downstream classifiers.

4.6 Classifier heads

Three heads c_ϕ sit on top of the same frozen 100-dimensional encoding $\boldsymbol{h}$, each producing a single sigmoid output $\hat{y}$ read as $P(\text{attack} \mid \boldsymbol{x})$, thresholded and trained with the binary cross-entropy loss.

1) CNN. The first head mirrors the baseline's convolutional design and is built around the 1-D convolution of (13). The 100-dimensional encoding is truncated to 96 and reshaped into a 3×32 tensor, three channels, each 32 wide. 96 was chosen because it factors evenly into $3{\times}32$ for the convolutional reshape; the remaining 4 dimensions are dropped. The full head runs Conv1d($3 \to 32$, $k = 3$, pad$= 1$) $\to$ ReLU $\to$ MaxPool1d(2) $\to$ Conv1d($32 \to 64$, $k = 3$, pad$= 1$) $\to$ ReLU $\to$ MaxPool1d(2) $\to$ Flatten $\to$ Linear($512 \to 128$) $\to$ ReLU $\to$ Dropout(0.5) $\to$ Linear($128 \to 1$) $\to \sigma$. The two pooling stages successively halve the width from 32 down to 16 and then to 8, leaving a flattened vector of $64 \times 8 = 512$ entries. Counting layer by layer gives $320 + 6{,}208 + 65{,}664 + 129 = 72{,}321$ head parameters, which combined with the 5,527-parameter autoencoder totals 77,848. This is the architecture used in Experiments 1 and 2.

2) MLP. The second head strips the convolutional layers out entirely, replacing them with the single hidden layer of (14)–(15). This is the smallest classifier considered anywhere in this study 8,792 parameters, encoder included.

3) MLP2. The third head builds on the MLP by adding a second hidden layer ($100 \to 64 \to 32$) and inserting the batch normalization after each layer, before the nonlinearity. Batch normalization was introduced here for a reason that the plain single-layer MLP showed a calibration problem, low decision-boundary confidence that the extra depth alone did not fix on its own. MLP2 totals 14,296 parameters including the encoder, still far below the convolutional head. Fig. 6 contrasts the two MLP variants directly.

4.7 Tools Used

The pipeline is implemented in Python 3.11 with PyTorch [17] on CUDA 12.8. scikit-learn [18] handles preprocessing, the stratified split, and metric computation, while imbalanced-learn [19] supplies the SMOTE [4] implementation used for the baseline comparison. All training and evaluation ran on a single NVIDIA GeForce RTX 5050 Laptop GPU (Blackwell architecture). Trained checkpoints are saved together with the fitted scaler and the selected-feature list, so every reported result can be reproduced from the saved artifacts alone.

5 Implementation and Experimental Evaluation

This section describes how the pipeline of Section 4 was actually run and evaluated, and what the results say. The evaluation is separated into three experiment families sharing the same frozen encoder and 733,705-record test partition: a **balancing-strategy comparison**, which fixes the CNN head and varies only how the training set is balanced, now run across multiple random seeds for every method *including* DMCR; a **target-size sensitivity ablation**, which fixes the balancing method to DMCR and

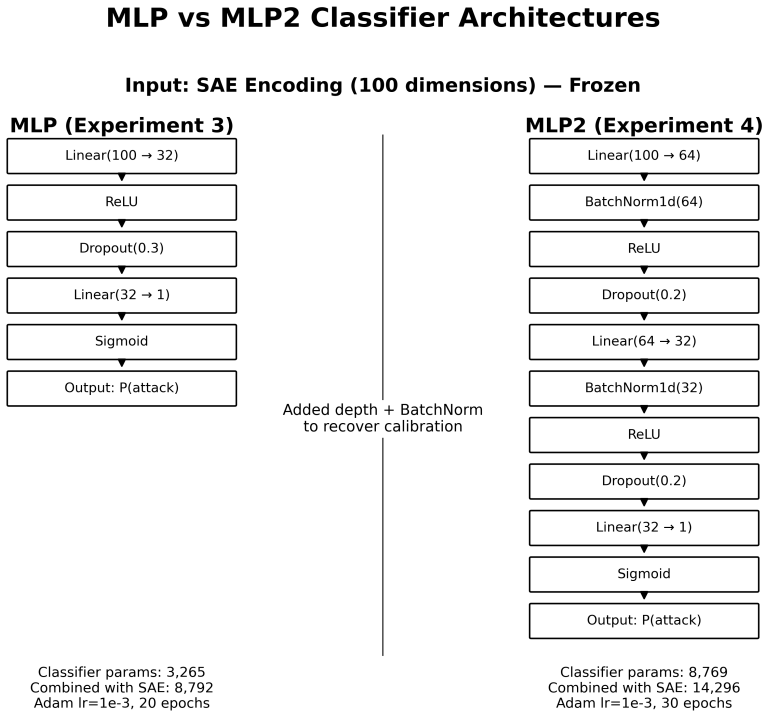


Fig. 6: The two multilayer perceptron classifier heads: MLP (left) and the deeper, batch-normalised MLP2 (right), both operating on the frozen 100-dimensional SAE encoding.

the CNN head and varies only how many majority samples N_{maj} are retained; and a classifier-size comparison, which fixes DMCR as the balancing method and varies only the classifier head, following the original single-run protocol.

5.1 Experimental setup

The balancing-strategy comparison places six balancing methods behind an identical CNN classifier and an identical frozen SAE encoder, each repeated across three independent random seeds (42-44) so that every reported metric, DMCR included, reflects a distribution rather than a single draw:

- **DMCR**: the dispersion-based majority reduction of Section 4.4, retaining 50,000 selected attacks plus all 382 normal training samples (50,382 total), evaluated across three seeds (42-44).
- **CSMCR**: the cosine-similarity-based majority reduction, retaining 50,000 selected attacks plus all 382 normal training samples (50,382 total), evaluated across three seeds (42-44), scored over the 100,000-candidate pool using a chunked cosine-similarity computation (chunk size 128).

- **SMOTE**: synthetic oversampling to full balance (≈5,868,870 samples), evaluated across three seeds (42-44).
- **ADASYN**: adaptive synthetic oversampling, which places synthetic points preferentially near the decision boundary rather than uniformly between neighbours; oversamples to approximate balance (≈5,868,870 samples, with the exact minority count varying slightly by seed since ADASYN's synthetic-sample count is itself data-dependent), evaluated across three seeds (42-44).
- **Borderline-SMOTE**: a SMOTE variant that synthesizes only from minority samples near the class boundary; oversamples to full balance (≈5,868,870 samples), evaluated across three seeds (42-44).
- **Random undersampling**: the majority class subsampled uniformly at random down to the minority count, giving a balanced set of only 764 samples (382 normal, 382 attack), evaluated across three seeds (42-44).

It is important to be clear about what the random-undersampling baseline is and is not. How well does the model perform if the majority class is just discarded down to the size of the minority, with no selection criteria at all? This is a traditional undersampling comparison. In order to isolate the contribution of the dispersion criteria specifically, a random subsample of the same size as DMCR's 50,000 would be drawn. This is not the ablation reported below; it is marked as future work in Section 4.4 and remains open (Section 6).

A second, newly added comparison probes how sensitive DMCR is to its own retention target rather than to the choice of balancing method. Holding the 100,000-sample candidate pool fixed, N_{maj} is swept over {10,000, 30,000, 50,000, 100,000}, each evaluated across the same three seeds. The $N_{\text{maj}} = 100{,}000$ setting is a limiting case in which the candidate pool and the retention target coincide: no candidate is discarded, so the dispersion criterion has nothing left to rank away, and DMCR degenerates to using the entire randomly drawn pool.

A four-experiment structure is reused, unchanged, in the classifier-size comparison: Experiment 1 replicates the SMOTE+CNN baseline; Experiment 2 isolates the balance shift by switching in DMCR while maintaining the CNN head. Experiment 3 keeps DMCR and swaps the CNN for the single-hidden-layer MLP, isolating the classifier-size change; and Experiment 4 keeps DMCR and swaps in the deeper, batch-normalized MLP2.

5.2 Hyperparameters

Table 2 lists the hyperparameters used for each pipeline component. Where a hyperparameter is shared across all classifier heads (e.g., the decision threshold, the Adam optimizer).

5.3 Results

This section presents the results across multiple experiments. Table 3 reports mean ± standard deviation across seeds for each balancing method, holding the CNN head and encoder fixed. DMCR is now evaluated under the same three-seed protocol as every other method, closing the single-run gap identified in earlier versions of this study.

Table 2: Hyperparameters by pipeline component

Component	Hyperparameter	Value
DMCR balancing	Candidate pool size n	100,000
	Retained majority samples N_{maj}	50,000
	Random seed	42-44 (three seeds)
	Validation split of balanced set	90:10, stratified
DMCR target-size ablation	Candidate pool size n	100,000 (fixed)
	Retained majority samples N_{maj}	10,000 / 30,000 / 50,000 / 100,000
	Random seed	42-44 (three seeds per setting)
	Validation split of balanced set	90:10, stratified
CSMCR balancing	Candidate pool size n	100,000
	Retained majority samples N_{maj}	50,000
	Random seed (pool draw)	42-44 (three seeds)
	Cosine-similarity chunk size	128
	Validation split of balanced set	90:10, stratified
Feature selection	Correlation threshold $r_{\min}$	0.01
	Retained feature count d	27
Sparse autoencoder	Hidden layer size	100
	Sparsity target ρ	0.3
	Sparsity weight β	1.0
	Optimiser	RMSprop
	Parameters	5,527
CNN head	Optimiser	Adam
	Dropout	0.5
	Head / total params	72,321 / 77,848
MLP head	Optimiser	Adam, lr $= 10^{-3}$
	Epochs	20
	Dropout	0.3
	Head / total params	3,265 / 8,792
MLP2 head	Optimiser	Adam, lr $= 10^{-3}$
	Epochs	30
	Dropout	0.2
	Head / total params	8,769 / 14,296
All heads	Decision threshold τ	0.5

SMOTE, CSMCR, and DMCR are showed very similar performance across the three seeds, on ROC–AUC (all ≥ 0.999997) and on F1 (all ≥ 0.99998) across three independent seeds each, and both majority-reduction methods clearly outperform random undersampling (0.996137 ± 0.002931, over 11,000 missed attacks on average), suggesting that a 50,000-sample selected subset can preserve detection quality, although the specific contribution of dispersion-based selection versus random sampling remains to be established. Reducing the majority to the bare minority count of 382, as random undersampling does, is not a free lunch. ADASYN and Borderline-SMOTE, despite also being synthetic-oversampling methods, show visibly larger ROC–AUC variance than

Table 3: Balancing strategy comparison: CNN head, mean ± std across three seeds (42-44)

Method	Seeds	Training samples	Synth.	ROC–AUC	F1	FP	FN
SMOTE	3	5,868,870	Yes	0.999999 ±0.000001	0.999995 ±0.000001	1.33 ±1.53	6.33 ±2.89
ADASYN	3	≈5,868,870	Yes	0.977182 ±0.039520	0.999990 ±0.000002	4.67 ±8.08	10.67 ±5.13
Borderline-SMOTE	3	5,868,870	Yes	0.994728 ±0.009127	0.999994 ±0.000001	1.67 ±2.08	7.00 ±2.65
Random U/S	3	764	No	0.996137 ±0.002931	0.992143 ±0.003709	0.00 ±0.00	11,425.33 ±5,356.31
CSMCR†	3	50,382	No	0.999997 ±0.000002	0.999984 ±0.000003	1.00 ±0.00	22.33 ±4.93
DMCR (proposed)	3	50,382	No	0.999997 ±0.000001	0.999986 ±0.000008	1.33 ±0.58	19.33 ±11.06

† Cosine Similarity-based Majority Class Reduction.

Table 4: Training time breakdown (seconds), mean over three seeds

Method	Balancing	SAE	Classifier	Total
SMOTE	3.09	550.60	3,251.23	3,804.92
ADASYN	3.22	540.34	3,231.87	3,775.42
Borderline-SMOTE	3.32	451.68	2,807.29	3,262.29
Random undersampling	0.34	0.14	0.52	1.00
CSMCR†	25.76	3.57	32.77	62.10
DMCR (proposed)	10.85	4.27	27.85	42.97

SMOTE itself: ADASYN's mean of 0.9772 is driven almost entirely by a single seed (seed 44: ROC–AUC = 0.9315, FP = 14), and Borderline-SMOTE's seed 44 similarly dips to ROC–AUC = 0.9842 (FP = 4), indicating that occasional seed-level instability in this extreme, inverted-imbalance setting is not unique to majority-reduction methods.

Table 4 reports the stage-wise training-time breakdown, averaged over the same seeds.

Recomputed against these multi-seed means, DMCR trains roughly 88.6× faster than SMOTE (3,804.92/42.97), and CSMCR trains roughly 61.3× faster than SMOTE (3,804.92/62.10). ADASYN and Borderline-SMOTE offer no comparable speed advantage over SMOTE, since both still oversample to full parity and remain dominated by the ∼5.87M-sample classifier-training stage; their totals (3,775.42s and 3,262.29s) sit in the same order of magnitude as SMOTE's. Random undersampling remains by far the fastest method (1.00s total), at the detection-quality cost shown in Table 3.

5.3.1 DMCR target-size sensitivity

Table 5 and Fig. 7 report DMCR's detection quality as the retention target N_{maj} is swept from 10,000 to 100,000 against a fixed 100,000-candidate pool, each point averaged over three seeds.

Table 5: DMCR target-size ablation: CNN head, mean ± std across three seeds, 100,000-candidate pool fixed throughout

N_{maj}	Training samples	ROC–AUC	F1	FP	FN
10,000	10,382	0.99987 ± 0.00021	0.999959 ± 0.000031	8.67 ± 8.62	51.00 ± 37.47
30,000	30,382	0.999995 ± 0.000002	0.999981 ± 0.000001	2.00 ± 1.00	26.33 ± 1.53
50,000	50,382	0.999997 ± 0.000001	0.999986 ± 0.000008	1.33 ± 0.58	19.33 ± 11.06
100,000[†]	100,382	$\mathbf{0.7541 \pm 0.4259}$	0.999968 ± 0.000029	32.33 ± 54.28	14.00 ± 13.11

[†]At $N_{\text{maj}} = 100{,}000$ the candidate pool and the retention target coincide, so no dispersion-based discarding occurs; this row reflects training on the full randomly drawn pool rather than a dispersion-selected subset.

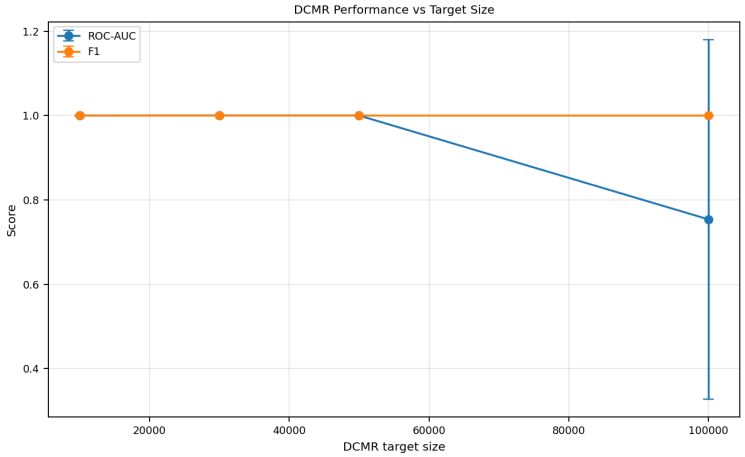


Fig. 7: DMCR ROC–AUC and F1 as a function of the retained majority-class target size N_{maj}, mean over three seeds with error bars showing ±1 standard deviation. F1 stays visibly flat at approximately 1.0 across every target size, while ROC–AUC collapses at $N_{\text{maj}} = 100{,}000$.

Detection quality is stable and essentially monotonic through $N_{\text{maj}} = 50{,}000$: mean ROC–AUC rises from 0.99987 at 10,000 to 0.999997 at 50,000, and FN falls from a mean of 51.0 to 19.3, consistent with a larger, more diverse retained pool giving the classifier more attack variation to learn from. This trend reverses sharply at $N_{\text{maj}} = 100{,}000$. Mean ROC–AUC falls to 0.754 with a standard deviation (0.426) larger than any other value in the table, and inspecting the individual seeds explains why: seed 42 collapses to ROC–AUC = 0.262 with every one of the 95 held-out normal samples misclassified as an attack (TN = 0, FP = 95), while seeds 43 and 44 remain near-perfect (ROC–AUC 0.999997 and 0.999998 respectively). One seed in three, at this setting, produces a detector that fails completely on the class it exists to protect.

This failure is invisible in the F1 column. F1 remains at approximately 0.99997 across the entire sweep, including the seed-42 collapse at $N_{\text{maj}} = 100{,}000$ (F1 = 0.999935 for that run specifically), because F1 as computed here is dominated by the attack class,

which outnumbers normal samples in the test set by roughly 7,700 to 1. A classifier that predicts "attack" for essentially every input still scores near-perfect F1 while offering no protection to the class this paper set out to protect; ROC–AUC exposes the failure immediately. This is a direct, practical illustration of why FP and FN are reported separately throughout this paper rather than collapsed into a single aggregate score, and it extends that argument to the aggregate metrics themselves: F1 alone would have hidden this instability entirely. Practically, this indicates that DMCR's dispersion-based selection step is not merely a computational convenience retaining a meaningfully filtered subset ($N_{\text{maj}} \ll$ pool size) appears necessary for stability.

5.3.2 Classifier head comparison

Table 6 repeats, unchanged, the single-run classifier-size comparison: DMCR balancing is fixed and only the classifier head varies.

Table 6: Classifier head comparison – DMCR balancing, single run per configuration

Metric	Exp. 1 SMOTE+CNN	Exp. 2 DMCR+CNN	Exp. 3 DMCR+MLP	Exp. 4 DMCR+MLP2
Training samples	5,868,870	50,382	50,382	50,382
Parameters	77,848	77,848	8,792	14,296
Total training time (s)	1,310.2	19.83	22.27	33.87
Inference (μs/sample)	1.504	1.027	0.568	1.936
ROC–AUC	0.999999	0.999998	0.9559	0.999999
False positives (FP)	0	1	10	0
False negatives (FN)	10	19	0	33

The timings in Table 6 correspond to the original single-run experimental protocol and are therefore not directly comparable with the three-seed mean timings reported in Table 4. As Fig. 8 shows, replacing the CNN with the single-layer MLP cuts parameters by nearly 90% (77,848 → 8,792) and drives false negatives to zero, at the cost of ten false positives and a drop in ROC-AUC to 0.9559, good coverage but weak probability calibration. MLP2's extra hidden layer and batch normalization restore ROC–AUC to 0.999999 and eliminate false positives entirely, at the cost of 33 missed attacks, using only 14,296 parameters (about 82% smaller than the CNN).

Fig. 9 contrasts these two error profiles directly, and together they give a deployer two distinct operating points from one pipeline: the MLP (FN = 0, FP = 10) suits passive monitoring, such as feeding a SIEM, where a missed attack is the costlier failure; the MLP2 (FP = 0, FN = 33) suits active-response settings, such as automatically disconnecting a flagged device, where a false alarm itself carries a cost. Neither dominates the other, and the pipeline's contribution is offering both from a single frozen encoder rather than one fixed compromise.

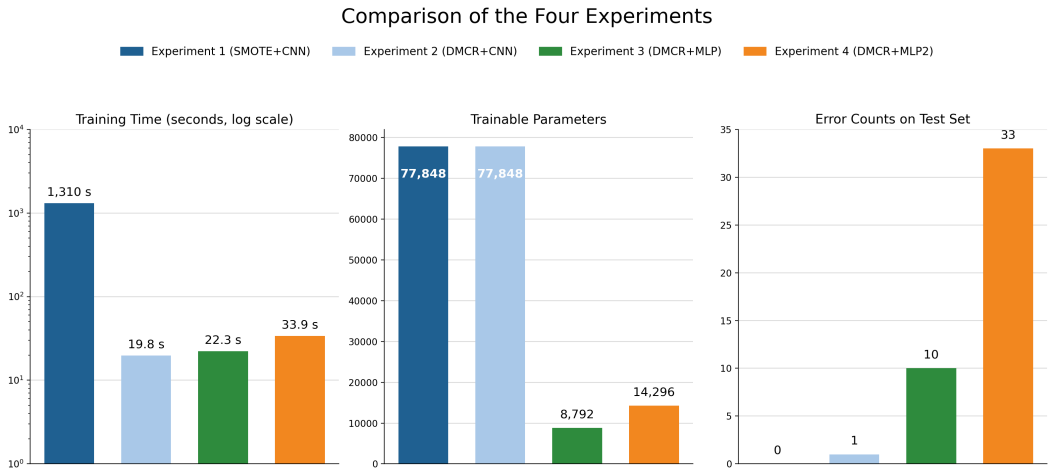


Fig. 8: Comparison of the four configurations across training time (log scale), trainable parameters, and error counts on the test set.

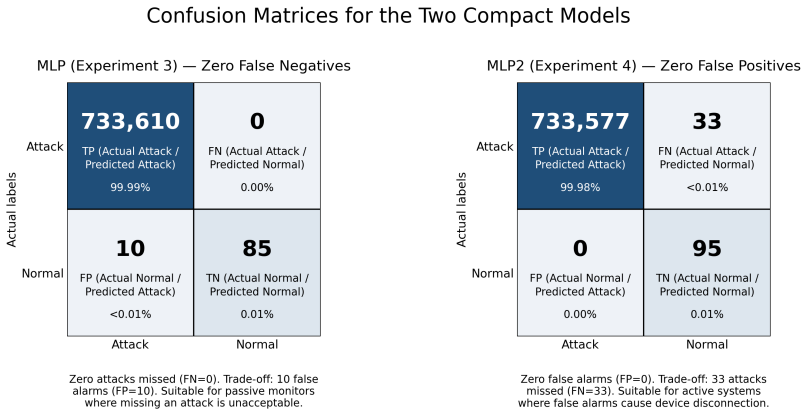


Fig. 9: Confusion matrices for the two compact models: MLP (zero false negatives) and MLP2 (zero false positives).

6 Conclusion and Future Scope

This study reviewed two design decisions made in a previous IoT intrusion-detection pipeline: the size of the classifier head and the way the majority class is balanced. Instead of creating fake minority samples, DMCR maintains the most mutually dissimilar actual majority samples, which are precisely scored using an order-statistics identity, in order to balance the training set. Five more balancing techniques were evaluated over three seeds: random undersampling, SMOTE, ADASYN, Borderline-SMOTE, and

CSMCR. DMCR, CSMCR, and SMOTE were statistically indistinguishable on ROC-AUC and F1, however random undersampling missed thousands of attacks on average. There was just one unstable seed in each of the oversampling versions (ADASYN, Borderline-SMOTE), indicating that the high, inverted imbalance in this dataset causes seed-level instability generally rather not only in majority-reduction techniques.

A follow-up sensitivity analysis found a limit to DMCR's stability: when the candidate pool and the retention target are collapsed to the same size, the dispersion criterion has nothing left to discard, and one of three seeds failed to detect any normal traffic at all — a failure invisible in the F1 score, reinforcing the paper's choice to report FP/FN separately. On the classifier side, a single-hidden-layer MLP matched the CNN baseline's zero-false-negative coverage at a ninth of its parameters, and a deeper, batch-normalized MLP2 matched its zero-false-positive coverage at under a fifth of its parameters, giving a deployer two lightweight operating points instead of one fixed compromise.

There are scope for future directions. The required ablation is a same-size random subsample from the same pool. It is unknown if DMCR's dispersion ranking, in particular, influences its detection quality rather than sample-size reduction alone. A more detailed sweep and a mechanical explanation are required for the target-size instability itself. It is recommended to repeat the MLP/MLP2 comparison across seeds as it is still single-run. Beyond this, before the pipeline's lightweight, DMCR-balanced design can be deemed deployment-ready, it should be verified on actual low-power hardware (such as a Raspberry Pi) and on other, more diverse IDS datasets.

Declaration of conflict of interest

The authors declare no conflicts of interest.

Data availability

Datasets used in this study are publicly available.

Funding Information

For this study, external funding has not been received.

Ethics, Consent to Participate, and Consent to Publish declarations

Not Applicable.

Author contributions

The first and fourth author contributed to the methodology and helped in conducting experiments. The second author contributed to the drafting and writing of the manuscript. The third author is responsible for the supervision and validation of the study. All the authors have reviewed the manuscript.

References

[1] M. M. Rahman, S. A. Shakil, and M. R. Mustakim. A survey on intrusion detection system in IoT networks. *Cyber Security and Applications*, 3:100082, 2025.

[2] S. Kumar, S. Gupta, and S. Arora. Research trends in network-based intrusion detection systems: A review. *IEEE Access*, 9:157761–157779, 2021.

[3] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[4] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[5] A. Qaddos, M. U. Yaseen, A. S. Al-Shamayleh, M. Imran, A. Akhunzada, and S. Z. Alharthi. A novel intrusion detection framework for optimizing IoT security. *Scientific Reports*, 14(21789), 2024.

[6] D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[7] X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2009.

[8] K. Narayana Rao, K. Venkata Rao, and P. V. G. D. Prasad Reddy. A hybrid intrusion detection system based on sparse autoencoder and deep neural network. *Computer Communications*, 180:77–88, 2021.

[9] M. A. Alsoufi, M. M. Siraj, F. A. Ghaleb, M. Al-Razgan, M. S. Al-Asaly, T. Alfakih, and F. Saeed. Anomaly-based intrusion detection model using deep learning for IoT networks. *Computer Modeling in Engineering & Sciences*, 141(1):823–845, 2024.

[10] M. Bhavsar, K. Roy, J. Kelly, and O. Olusola. Anomaly-based intrusion detection system for IoT application. *Discover Internet of Things*, 3(1):5, 2023.

[11] S. Hizal, U. Cavusoglu, and D. Akgun. A novel deep learning-based intrusion detection system for IoT DDoS security. *Internet of Things*, 28:101336, 2024.

[12] M. Vishwakarma and N. Kesswani. A new two-phase intrusion detection system with naïve bayes machine learning for data classification and elliptic envelope method for anomaly detection. *Decision Analytics Journal*, 7:100233, 2023.

[13] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-IoT dataset. *Future Generation Computer Systems*, 100:779–796, 2019.

[14] A. Ng. Sparse autoencoder. Cs294a lecture notes, Stanford University, 2011.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, 2015.

[16] Arvind Prasad, Wael Mohammad Alenazy, Naved Ahmad, Gauhar Ali, Hanaa A Abdallah, and Sadique Ahmad. Optimizing iot intrusion detection with cosine similarity based dataset balancing and hybrid deep learning. *Scientific Reports*, 15(1):30939, 2025.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pages 8024–8035, 2019.

[18] 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.

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