Exploring Anti-Money-Laundering: Classification of Pre-Raised Alarms

A Preliminary Study Using Real-World Alarm Data

9th Semester Master's Project

Aleksandar Nikolaev Stavrev, Joyesh Meshram,
Mihail Grigorov Gerginov, Nicklas Peter Kvist Gislinge

anst24@student.aau.dk, jmeshr24@student.aau.dk,
mgergi24@student.aau.dk, ngisli21@student.aau.dk

Department of Computer Science, Aalborg University
Aalborg, Denmark

***Abstract*—Transaction monitoring systems generate large volumes of Anti Money Laundering (AML) alarms, many of which do not escalate. This paper studies the downstream task of predicting the outcome of alarms after they are raised, using available alarm and customer features. We evaluate monolithic tabular models, ensembles constructed by partitioning data by employee or by alert category, a voting classifier, neural network models, and AutoGluon. Evaluation uses class sensitive metrics and safeguards designed to reduce over optimization. Results show that the partitioned ensembles and the neural models do not yield consistent benefits in this setting, while AutoGluon achieves the strongest overall performance and a robust balance across metrics. The resulting binary decision supports a screening workflow for human investigation.**

***Index Terms*—Anti-Money Laundering (AML); Transaction Monitoring Systems (TMS); Machine Learning; Binary Classification; Ensemble Learning;**

I. Introduction

The amount of reported money laundering in Denmark continues to rise year after year, with the vast majority of reports coming from financial institutions [1]. Institutions produce these reports by manually investigating potentially fraudulent transactions that have been flagged by their internal monitoring systems.

One problem with this system is that, due to the need for regulatory compliance, it is designed to flag too many transactions rather than too few [2]. This results in the majority of work performed by AML employees being spent investigating false positives created by the system. One of the biggest problems affecting AML detection methods is the quality of the data, namely the extreme class imbalance of datasets combined with the need for near perfect precision [3]. In real world scenarios, imbalanced datasets are to be expected, but within AML the majority of alarms are non fraudulent.

As an example of this, the real-world dataset used in this paper consists of $21{,}583$ alarms, while only $2{,}145$ of those were determined to be actual fraud, and $14{,}841$ were non fraudulent. That equates to a dataset imbalance of 87.37% and 12.63%, respectively. This also excludes the currently active alarms that would skew the data more.

Additionally, due to the need for regulatory compliance, it is important for financial institutions to successfully detect truly fraudulent activity, as failure to do so may result in heavy fines placed upon the institution by regulatory bodies [4]. As a by product of this, AML employees spend a lot of time sifting through alarms in order to manually separate fraudulent from non fraudulent activity.

At the transaction level, industry reported false positive rates are typically in the range of 90–95%, meaning that the vast majority of transactions flagged by rule based monitoring systems do not correspond to confirmed fraud [5]. These false positive rates are for the alarm generation stage, where within millions of transactions that they do not directly reflect the class balance of alarm level datasets used for investigation. Once alarms have been raised, the relevant task shifts from detecting suspicious transactions to prioritising already flagged alarms for manual review. It is this post alarm setting that is the focus of the present work.

This costs time, money, and effort that could be better spent investigating real fraudulent transactions, or anticipating and developing measures for emergent fraud patterns.

Much of the AML research studies transaction level detection, where classification models predict whether individual transactions are suspicious or ordinary, often as an alternative to the rules-based systems that many financial institutions use to flag activity [3, 6, 7, 8, 9, 10]. In contrast, fewer studies consider the downstream setting in which an institution has already raised alarms and the task is to classify or triage these pre-raised alerts using the information available at investigation time. This is partly explained by the sensitivity of transaction data and the practical difficulty of obtaining real-world alarm datasets.

In this work, we study post alarm classification as a decision support task. Rather than changing the upstream detection system, we focus on classifying alarms that have already been

raised, with the goal of reducing the time and effort AML employees spend investigating false positives.

One of the few papers that explicitly studies classification of pre-raised alarms [11] proposes an approach that allows approximately 13% of suspicious transactions to go undetected. As in [11], we focus on transactions that have already been flagged by the bank's internal systems.

To support this, we obtained access to internal alarm data through a collaboration with Lån & Spar Bank (L&SB), a Danish bank. The bank primarily handles personal banking, which is reflected in the provided data.

Another important circumstance is that L&SB recently reworked their entire internal alarm system, leading to the obsoletion of data from before January 1st 2025. This results in smaller amounts of data being acquired than would otherwise ordinarily be available.

Given the limited dataset size, we evaluate false positive reduction using traditional classification approaches that are less sensitive to small sample sizes. We then benchmark several modelling strategies, including monolithic models, ensembles, Neural Networks (NNs), Voting Classifiers (VCs), and AutoGluon, to assess whether any approach can achieve performance sufficient to reduce the workload of AML employees.

The remainder of the paper is organized as follows. Section II reviews related work on AML classification. Section III presents the problem formulation and describes the AML workflow considered in this study. Section IV details the data and the preprocessing approach. Section VI explains the modelling approaches taken in this study. Section VI describes the evaluation methodology and experimental setup. Section VII presents the results, and Section VIII finishes with the paper and outlines directions for future work.

II. Related Works

The detection of money laundering through machine learning has evolved over the past decade, with research spanning core detection methodologies, human-AI collaboration, and advanced neural architectures. This section reviews literature within those fields. However, due to the specific niche, role, and organizational placement of the proposed solution within L&SB, certain aspects of this body of work are less directly applicable in our context and are discussed inline below.

A. Traditional Machine Learning Approaches for AML

The application of machine learning to anti-money laundering detection has been widely studied. Early reviews by Chen et al. [6] establish foundational taxonomies of machine learning techniques, including anomaly detection, clustering, classification, and network analysis for suspicious transaction detection. Jensen and Iosifidis [3] provide a more recent framework, distinguishing between client risk profiling and suspicious behaviour flagging, addressing class imbalance challenges, feature engineering issues, and evaluation metrics appropriate for imbalanced datasets.

Building on these foundations, recent work has demonstrated the practical effectiveness of supervised learning approaches on real-world banking data. Jullum et al. [9] use data from DNB, Norway's largest bank, to show that including non-reported alerts in training improves model performance. They also introduce performance measures for comparing ML approaches to existing rule-based AML systems. Namdar et al. [10] present a 16-step supervised learning pipeline achieving 0.961 Area under Receiver Operating Characteristic Curve. (AUC-ROC) on high-risk client identification, with emphasis on explainable AI modules for feature importance, which improves understanding of how models prioritize clients for investigation.

One recent survey by Fan et al. [8] reviews deep learning approaches for mobile transaction AML, covering Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), transformers, and Graph Neural Networks (GNNs) while addressing challenges of false positives, limited data availability, and real-time detection requirements.
Together, the above works provide a general theoretical basis, and serve as a preliminary guide for our model selection.

B. Human-AI Collaboration and Modelling Analyst Behaviour

An underexplored area is how human analysts and AI systems collaborate in AML detection, and how analyst decision-making patterns can be captured and replicated. Recent research [12] demonstrates how Subject Matter Expert (SME) feedback annotations can be propagated through transaction graphs to improve fraud detection models, showing 7.24% average improvement in AUC when human expertise from AML analysts is incorporated into ML systems through feedback loops.

Nagarakanti [13] provides analysis showing that experienced financial professionals improve AI-generated risk assessments by 42.8% when market conditions deviate from historical patterns, and that human interventions improve lending outcomes by 19.7% for edge cases. The documented collaborative fraud detection workflow, in which analysts correctly resolve 81.7% of complex anomaly cases, illustrates the importance and impact of expert analyst decision-making within human–AI systems.

A methodological framework for evaluating Human-AI Collaboration systems [14] includes fraud detection as a domain of study, examining task allocation, interaction patterns, and how humans and AI contribute to collaborative outcomes. The framework discusses detection rates (true/false positives) as evaluation metrics and provides structured approaches for evaluating collaboration modes (AI-Centric, Human-Centric, and Symbiotic), which is directly applicable to understanding how AML analysts should interact with and be modelled within AI systems.
The insights of these papers align closely with the objectives of the proposed system, which only aims to assist AML employees, not replace them. Particularly, we explore the possibility that employee-based ensembles may exhibit particular aptitude for this task. The results will be discussed in section VII.

C. Addressing Class Imbalance in AML Detection

Class imbalance is a practical challenge in AML detection, as fraudulent transactions often constitute less than 1% of all data. Liu et al. [15] address this challenge through PC-GNN, a graph-based approach using label-balanced sampling to construct sub-graphs for mini-batch training, neighbourhood sampling to choose relevant neighbours, and aggregation across different relations. Tests on YelpChi and Amazon benchmark datasets show improvements over state-of-the-art baselines.

Ruchay et al. [16] focus on handling class imbalance in fraudulent bank transactions using the CreditCardFraud[1] dataset. Their work employs Tomek Links' resampling algorithm and emphasizes appropriate evaluation metrics for imbalanced data (Precision, Recall, F1, IBA) rather than accuracy alone, achieving 99.99% accuracy with proper handling of imbalance.

III. Problem Formulation & Prerequisite

This paper aims to help AML employees focus their efforts on the alarms that matter. To do this, we must address the underlying problem that current rules-based AML systems generate a large volume of false-positive alarms, creating a significant operational burden while still requiring near-perfect detection of true fraud due to regulatory constraints. Given this setting, the challenge is not to replace existing alarm systems, but to reduce analyst workload without discarding any alarms outright.

To address this problem, the paper investigates whether different modelling assumptions and data partitioning strategies affect the ability to classify pre-raised alarms under real-world constraints. We will investigate:

- Whether a monolithic classification model (i.e., a single classifier trained on the complete, non-partitioned dataset, as opposed to ensemble approaches that divide data into subsets) is sufficient for classifying pre-raised AML alarms.
- Whether partitioning data by AML employee captures decision-making patterns that improve classification.
- Whether alarm category–specific models better capture heterogeneity in alarm types. Alarm categories are described in section III-A.
- Whether more expressive models, such a tabular oriented NNs, provide benefits under severe class imbalance and limited data.
- Whether combining heterogeneous classifiers into a voting classifier yields more robust prioritisation.
- Whether an automated ML pipelines, such as AutoGluon, a state-of-the-art automatic Machine Learning (ML) library that automates the ML process, can outperform manually configured models in this constrained setting.

Each model will be tested with multiple different permutations of base-classifier, to ensure thorough analysis of the viability of simplistic classification algorithms for alleviation of AML employee workload. We will also investigate the performance of multiple NN classifiers, even though we suspect their performance will be lacking, due to the nature and amounts of data available.

[1]https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

A. The AML Workflow

To contextualize the problem and its operational constraints, we first describe the AML workflow within L&SB. A sketch of their general workflow, as well as the suggested changes from this project, can be seen in Figure 1.

Within L&SB it is the work of AML employees to distinguish between real and false alarms flagged by their system. They have a rules-based system with approximately 70 different flags that a transaction can trigger. These flags are categorized into nine categories:

1) Tax haven countries
2) Deviation from expected behaviour
3) Risk list entities
4) Watch list entities
5) Debitcard transactions
6) Wire activity
7) ATM/Phone activity
8) Manual alarms
9) Counterparty terror

Of these categories, the only one that is discarded when training our models is 8 - Manual Alarms, as they are not raised by the system, but are instead raised manually by employees. These alarms are created based on reasons that come from outside the system itself (e.g. phone calls, meetings, etc.) Therefore, it would be impossible for our system to classify them.

When a flag is triggered, an alarm is made to the system, detailing everything from what, how, when, and who raised suspicion. Furthermore, the category, account, and person who triggered the alarm are also taken into consideration. It is then up to each AML employee to investigate what type of alarm it is, how it was triggered, if it is consistent with account patterns, and to potentially ask for additional documentation from the account holder. As can be seen in Figure 1, if an employee deems that additional information is required the label of the alarm may change multiple times as information becomes available. If no additional information is required, the alarm will be closed without ever contacting the customer, before the looping node. The end result of this investigation will then be either a *CLOSED* or a *ADDED TO CASE* label, henceforth referred to simply as *ADDED*. The *CLOSED* label means that the alarm was a false positive, and the *ADDED* label means that the bank cannot rule out fraud. When the *ADDED* label is given to an alarm it is then, along with any evidence or documentation gathered by the AML employee, sent to the appropriate authorities. A third label *ACTIVE* can also be given to an alarm but this label is always temporary, as this is the label for an ongoing investigation. In this paper, we discard all instances of this label, as we cannot be certain of its proper label yet. Lastly, if the investigation of an alarm cannot be closed by an AML employee without additional information or actions, the alarm may be assigned a variety of interim-labels, indicating its current status. These labels are unimportant for the models, which is further described in

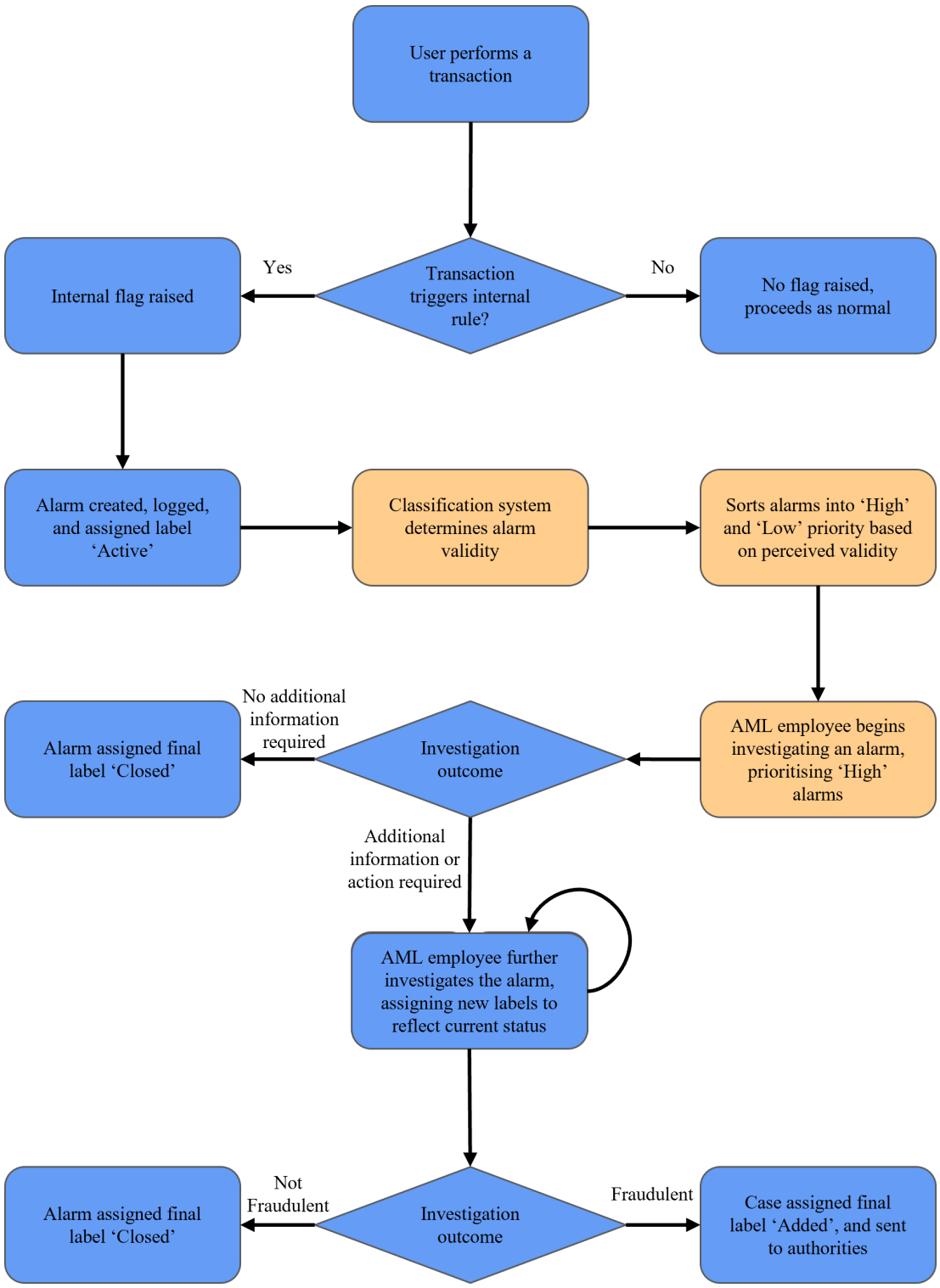

Fig. 1: Proposed workflow of AML employees after implementing our model. The model acts as a screening device, allowing for employees to focus on the alarms that are most likely to be true alarms. Blue nodes represent the existing bank workflow. Tan-coloured nodes are changed or added by implementing the model.

IV-D0b.

In summary, the problem addressed in this paper is whether pre-raised AML alarms can be reliably prioritized using post-alarm classification models, under extreme class imbalance, limited data availability, and strict regulatory constraints that prohibit automatic dismissal of alarms. It should be mentioned

that we will not assign alarms a probability of fraud and will only classify them in a binary manner. This classification is what is meant by 'high' and 'low' priority alarms. The 'high' priority alarms being the ones that the classifier assigns the *ADDED* label, i.e. the ones where L&SB's rule-system and the classification system agree, and the 'low' priority alarms being the ones where the two systems disagree.

IV. Data and Preprocessing

This section describes the datasets, the label definition used for supervised learning, and the preprocessing steps used to construct a single alarms training dataset.

A. Data Sources

All tables provided by Lån & Spar Bank span the period from January 1, 2025 to September 30, 2025, and are pseudonymized. The raw tables used in this study are:

1) **Alarms**: Details all alarms raised by the automatic flagging system. Includes current alarm status, associated account, ID of the AML employee handling the alarm, internal account designations, etc.
2) **Alarm Events**: Shows all actions taken related to a specific alarm. The same alarm ID may appear multiple times in this table if, for example, an AML employee has first requested documentation for the case from the account holder, then added documentation when the account holder has provided it, then closed the alarm.
3) **Customers**: General customer information such as age, place of residence, marital status, etc.
4) **Other tables**: Supplementary tables that are only used for connecting connecting Customers to Alarms.

Table I summarizes the raw table sizes and Table II summarizes alarm label counts.

B. Labels and Data Instances

Each data instance corresponds to a single alarm. Each alarm is associated with a final investigation outcome which is what the models will be trying to predict , either *CLOSED* or *ADDED TO CASE*. We refer to the latter simply as *ADDED*. *CLOSED* indicates that the alarm was resolved as a false positive, while *ADDED* indicates that the bank could not rule out suspicious activity. A third label, *ACTIVE*, is used for ongoing investigations and is always temporary. We discard alarms with this label because they do not yet have a final outcome.

C. Dataset Construction

We construct a single alarm-level dataset by consolidating the relevant source tables into one row per alarm, joining alarm metadata with customer attributes and derived information from alarm events where applicable.

We exclude alarm rows that are not suitable for training, including:

- **Manual alarms**: that are not raised by the rules-based system: These lack supporting data explaining why they were raised.

Description	Count
Customers	420,982
Alarms	21,583
Alarm Events	42,765

TABLE I: Data summary for the raw dataset

Label	Count
CLOSED	14,841
ADDED	2,145
ACTIVE	4,597

TABLE II: Alarm label counts in the raw dataset

- **Alarms closed due to rule revisions** that reflect process changes rather than underlying alert/transaction behavior.
- **Rows with null values**. We focus only on transactions that triggered the internal alarm system, so incomplete records were dropped.

After applying these criteria (and discarding *ACTIVE* alarms), the final alarm-level dataset contains $7{,}634$ alarms.

D. Preprocessing and Feature Engineering

Each raw dataset is cleaned and prepared independently and relevant datasets which were determined to be Alarms, Customers and Alarm events were joined to produce a single dataset.

*a) **Feature Selection**:* Exploratory analysis identified features that contributed no useful information. These features were either IDs or static values or duplicates of other columns. Alarms had 21 such features, while customers had 13 such features. These features were removed before model training. After transforming and pruning, the training dataset contained around 19 feature columns.

*b) **Target Label Preprocessing**:* Initially only the final label of an alarm was used for classification. However, after a discussion with L&SB, they requested that if an alarm had ever been investigated at all by a human, i.e., if an alarm ever had any of the before-mentioned interim-label, then they would like the system to detect such alarms. These modifications to the dataset were insignificant and only two rows were impacted which made no difference with the current data but might be of use in the future.

*c) **Feature Engineering and Encoding**:* Feature construction followed established practices for tabular modeling. Continuous variables such as Risk Score were scaled where appropriate to mitigate difference in magnitude, categorical variables were one hot encoded, and datetime fields such as the date since the customer has been active to number of days since they have been active for have been converted to days and birthday has been converted to age.

*d) **Implementation Note**:* Data processing is implemented in Python, using Polars for most transformation and loading steps and Pandas for quick inspection and validation.

E. Transaction Linkage and Scope

As with other reasearch in AML fraud detectiong we were also given accesss to transactions There was no clear way to connect them. The best approach was to use some information (such as using the alarm timestamp, total alarm amount, and the bank days mentioned) associated with some alarms to guess the associated transactions. Only around 70% of the 7,634 alarms were theoretically linkable. We were informed that additional payment-processing data from Netcompany Banking Services might enable more reliable alarm to transaction linkage, but this data is not available to us. Given this limited scope of connection we do not train a transaction-level dataset in this study.

V. Modelling Approaches

We treat alarm handling as a supervised binary classification task. Given the features available at decision time, the model predicts whether an alarm is more likely to be escalated (*ADDED TO CASE*) or not (*CLOSED*). We compare tabular baselines, sequence models over customer alarm histories, and simple ensembles.

A. Tabular baselines and tree ensembles

We benchmark multiple classifier families under a single, consistent training and evaluation protocol. The scikit learn estimator interface makes this practical, because it standardizes fitting, prediction, and model selection across a wide range of methods [17]. We treat model choice as empirical, since there is no universally best learner across problem settings [18].

We include Logistic Regression as a transparent baseline. In high stakes decisions, there are strong arguments for using interpretable models when they are adequate for the task. [19]

We also include simple classical baselines. k Nearest Neighbours is a standard nonparametric reference method in pattern recognition [20], and Gaussian Naive Bayes provides a lightweight probabilistic baseline.

For tree models, we include both single trees and ensembles. Random Forests are a widely used ensemble method based on bagging over trees. [21] We also include gradient boosting, which builds an additive model by iteratively fitting weak learners to improve the objective. [22] To cover widely used scalable implementations of boosting, we include XGBoost and LightGBM. [23, 24]

Because alarm outcomes are typically imbalanced, we report class sensitive metrics and summarize trade offs using precision and recall rather than accuracy alone. [25]

The evaluated classifiers are:

- Logistic Regression
- K Nearest Neighbours
- Gaussian Naive Bayes
- Decision Tree
- Random Forest
- Histogram based Gradient Boosting
- XGBoost
- LightGBM
- Recurrent Neural Network
- GRU
- Transformer

B. Sequence Models

In addition, we evaluate sequence oriented neural architectures as a separate experimental track, including recurrent neural networks, GRU, and Transformers [26, 27, 28].

Neural Networks: Neural networks represent a complementary approach to the traditional machine learning classifiers discussed above. We include them to investigate whether sequence-aware architectures can exploit temporal dependencies in alarm histories. However, we hypothesize that neural networks perform suboptimally compared to classical machine learning methods in this setting, as suggested by recent evaluations of deep models on tabular datasets [29, 30]. The primary reason is the limited size of the available dataset: with approximately 7,600 alarms and 19 features, the data volume falls well below the scale at which deep learning methods typically demonstrate their advantage over gradient-boosted trees and other tabular classifiers. Neural networks generally require larger datasets to learn robust representations and avoid overfitting, whereas tree-based ensembles and linear models are better suited to small-to-medium tabular datasets with hand-crafted features [29, 30].

1) Model Choice: To model temporal dependencies in alarm histories, we employ neural architectures that learn a latent representation of a customer's alarm sequence. In general, these models process the sequence of past alarms and produce a hidden representation that summarizes the relevant information, which then serves as input to a linear classifier for the current decision.

RNNs provide a natural baseline for this setting by iteratively updating a hidden state as new alarms arrive. However, standard RNNs are known to suffer from vanishing gradients, which limits their ability to capture dependencies over longer sequences [31].

To address this, we also consider Gated Recurrent Units (GRUs), which introduce update and reset gates to regulate the flow of information across time steps. These gating mechanisms allow the model to retain or discard historical information selectively, making GRUs well suited for scenarios where older alarms may become irrelevant as customer behaviour evolves [32, 33].

In addition, we consider Transformer-based encoders, which replace recurrence with self-attention mechanisms. Self-attention allows each alarm in a sequence to attend directly to all others, enabling the model to learn which past alarms are most informative for the current decision, regardless of their temporal distance [28].

2) Variable-length Histories and Padding: Customer alarm histories vary substantially in length (1-62 alarms). To accommodate this variability, sequences are padded to a fixed maximum length of 10 alarms. For recurrent models, padded elements are excluded from computation via sequence packing, while the Transformer model employs padding masks and masked pooling operations. As a result, the learned representations depend only on observed alarms and not on

artificial padding.

C. Ensembling

1) Voting Classifier: Using the Scikit-learn VotingClassifier we can create an ensemble that introduces diversity between the models. This method takes outputs from the ensemble models and performs a majority vote to create a final result. This works well on models that perform similarly well and can eliminate their individual weak-points. However, the implementation in Scikit-learn is designed in a way where it always fits the models itself. This made it difficult to get a fair comparison between other methods such as what we have already done. To bypass these limitation we inject the pre-trained models and skip the fitting step. This saved on duplicate training time and ensured accurate comparisons. This also meant going back to the original implementation and making sure to save the models, as well as any other information that is needed to test the results, such as the train-test split or the label encoder.

VI. Experiments

This section will outline our evaluation criteria, our experimental setup, as well as the results of our training and testing.

A. Evaluation Criteria

Before discussing the experiments and their results, we must first discuss on what basis the models will be evaluated. The purpose of this paper is to assist AML employees in focusing on the alarms that matter. For a model to do this in the optimal way, it must distinguish between true alarms and false alarms perfectly. However, the creation of a perfect classifier is an unattainable goal, so we must look elsewhere. In operational settings, the classifier should ideally satisfy one of two extreme requirements. The first option is to correctly classify all *ADDED* alarms, which is equivalent to achieving 100% *recall* on the *ADDED* label. The second option is to not classify any *ADDED* alarms incorrectly, which would be equivalent to achieving 100% *precision* on the *CLOSED* label[2]. Definitions for these two metrics can be seen in equations 1 and 2.

EQUATIONS (1) to (5):
TP = True Positive, or correctly classified instances of the positive class.
TN = True Negative, or correctly classified instances of the negative class.
FP = False Positive, instances of the negative class that have been falsely classified as positive.
FN = False Negative, instances of the positive class that have been falsely classified as negative.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1}$$

[2]While these two metrics are mathematically identical in the perfect case, outside of such scenarios their practical qualities differ. Proof of this can be found in Appendix A.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

By striving for these metrics, we would minimize the two most important error types. Should our solution reach 100% *recall* on the *ADDED* label, and this performance also hold on unseen data, then no *ADDED* alarms would be classified as *CLOSED* (i.e., $FN = 0$). Conversely, should our solution reach 100% *precision* on the *ADDED* label, and this performance also hold on unseen data, then no *CLOSED* alarms would be classified as *ADDED* (i.e., $FP = 0$).
Since 100% performance cannot be guaranteed, we report precision and recall to quantify these trade-offs on held-out test data.

Judging which model might be best for this sorting process is not easy, since achieving a high score on *recall* or *precision* is not enough. There must be balance between the metrics, which is represented by the addition of summary metrics such as *accuracy*, *F1-score*, and the more general F_β*-score*. The definitions of these metrics can be seen in equations 3, 4, and 5.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

In addition to $F1$, we report the F_β*-score*, which allows explicit control over the trade-off between precision and recall:

$$\text{F}_\beta\text{-score} = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}} \tag{5}$$

When $\beta > 1$, recall is weighted more heavily than precision when $\beta < 1$, precision is weighted more heavily. In this study we report F_3 to reflect a strong preference for recall on the *ADDED* class, while still penalising excessive false positives. For $\beta = 3$ we have $\beta^2 = 9$, so

$$F_3 = \frac{(1+9)\,PR}{9P + R} = \frac{10PR}{9P + R},$$

which can be interpreted as weighting recall nine times more than precision.
The reason for a high *recall* or *precision* score not being enough, is that a classifier labelling all alarms as *ADDED*, would achieve 100% *recall* on the *ADDED* label. However, such a classifier would not be useful for the purpose of lowering false positives, and determining high-priority alarms. Due to the fact that nothing could be designated as low-priority, since nothing would have been labelled *CLOSED*, thereby necessitating the *accuracy*, *F1-score* metrics and F_β*-score*.

To rank classifiers on their ability to correctly classify the alarms, we will evaluate them based on eight different criteria, with eight versions of each base classifier being trained, one optimized for each criteria. The eight different criteria are:

- Recall *ADDED*
- Precision *ADDED*
- Accuracy
- F-Beta *ADDED*
- Recall *CLOSED*
- Precision *CLOSED*
- F1-Score
- F-Beta *CLOSED*

The definitions of each metric can be seen in Equations (1) to (5). Should no model exhibit clear dominance over others, the best will be decided by amount of high scoring metrics with ties broken by summation of model metric values.

B. Experimental Setup

This section provides clarification on the details of the experimental setup and training of all models.

1) Train-Validation-Test Protocol: All experiments follow the same three-way split into train, validation, and test data to ensure comparable results across traditional tabular models, ensemble/voting variants, and neural networks. This means that all Monolith Models (MMs), all Employee Ensembles (EEs), and all Category Ensembles (CEs) were trained on the same split. The reason for this is that later on in the analysis, we will be combining different base classifiers from the originally homogeneous ensembles into a voting classifier, and if they were not trained on the same split, some models may have encountered the test data in their own training. Model parameters are fit on the training split. Across all experiments, we tune this decision rule to maximize recall on the *ADD* class on the validation split, reflecting the priority of surfacing potentially fraudulent alarms for review. This validation-time threshold selection is applied consistently even when the model itself was trained or optimized using a different scoring metric. Final performance is reported once on the held-out test split using this fixed configuration.

Where model families require stratification or partitioning (e.g., employee- or category-specific splits for ensembles), the partitioning is applied *within* the train/validation/test framework so that no model or component is tuned on, or exposed to, the test split.

2) Cross Validation: To obtain robust performance estimates during model selection under severe class imbalance, we use *stratified* k-fold cross validation on the training split. Stratification ensures that each fold preserves the overall *CLOSED*/*ADDED* class distribution. We set $k = 5$ by default and reduce it only if necessary to ensure that every fold contains at least one instance of the minority class, with $k \geq 2$, avoiding folds with no minority examples. For each hyperparameter setting, the model is trained on $k - 1$ folds and evaluated on the remaining fold. Scores are aggregated across folds using the mean to rank hyperparameters settings.

3) Data Imbalance: For dealing with the imbalance present within our dataset, we utilise both Synthetic Minority Over-sampling Technique (SMOTE) and weighting of the classes. Given the constraints of the available dataset and the system's role as a decision-support tool rather than a stand-alone detector, more complex resampling or graph-based approaches are considered out of scope.

4) Amount of Models: Since we have multiple different base classifiers, as well as multiple evaluation criteria, we decided to train multiple models with different purposes to see if any of them display particular aptitude for each task. This means that $8 \cdot 8 = 64$ different MMs will be trained, due to there being eight different base classifiers, when excluding the NNs, and eight different criteria. For the EE $8 \cdot 8 \cdot 10 = 640$ will be trained, due to there being ten different employee train-test splits, and $8 \cdot 8 \cdot 9 = 576$ for the CE, as there are nine categories for train-test splits.

5) Ensembles: In this experiment, we aim to find out if splitting the data based on some attribute and training an ensemble of multiple smaller models can help increase the scores of various performance metrics, compared to a monolithic approach. Ensemble models were originally inspired by how a team of people, each an expert in a different field, would achieve better results by making individual decisions and voting to decide on a final answer. As mentioned earlier we found two appropriate divisions of the data, one based on the type of alarm, and one based on the employee who handled it.

In terms of voting, both a regular majority vote, where the answer with most votes wins, and a vote where at least one positive label occurs were tested. The first option is standard and would usually lead to higher accuracy, the second one provided bias towards the underrepresented class. However, both voting types achieved similar results and ultimately regular majority vote was used in the shown results.

6) Neural Network Training: Neural architectures process each customer's alarm history and produce a scalar logit via a final linear layer. Applying the sigmoid function to this logit yields a value $p \in [0, 1]$, which we interpret as the escalation probability, i.e., the likelihood that the alarm should be escalated to the *ADDED* class.

We classify an alarm by comparing p against a decision threshold τ: if $p \geq \tau$, we label the alarm as *ADDED*, otherwise as *CLOSED*. During training, we fix $\tau = 0.5$, but during evaluation we treat τ as a hyperparameter and tune it within $[0.5, 0.95]$ to trade recall for precision.

We initially optimize using standard Binary Cross-Entropy (BCE):

$$\mathcal{L}_{\text{BCE}} = -[y \log(p) + (1 - y) \log(1 - p)].$$

Given the class imbalance in our dataset (87.37% *CLOSED* vs 12.63% *ADDED*), standard BCE risks being dominated by the majority class, causing the model to optimize primarily for correctly classifying non-fraudulent cases while neglecting the minority class. As noted by Ruchay et al. [16], this is problematic in AML contexts, where missing true fraudulent activity carries regulatory consequences.

To address this, we also train with focal loss as introduced by Lin et al. [34], which uses a modulating factor $(1 - p_t)^\gamma$ to

down-weight well-classified examples:

$$\mathcal{L}_{\texttt{focal}} = -\alpha(1-p_t)^{\gamma}\log(p_t).$$

Here, p_t denotes the predicted probability for the ground-truth class, α is a class-balancing weight, and $\gamma \geq 0$ is a focusing parameter. When an example is easily classified ($p_t \to 1$), the modulating factor approaches zero and the loss contribution becomes negligible. Conversely, for hard or misclassified examples ($p_t \to 0$), the loss remains significant. This mechanism allows the model to focus training on hard examples, predominantly the minority *ADDED* class, without requiring explicit resampling techniques such as SMOTE.
Focal loss suits our recall-oriented objective: by reducing the gradient contribution from the abundant *CLOSED* examples, the model can achieve high recall on *ADDED* alarms at lower decision thresholds, ensuring that potentially fraudulent cases are surfaced for human review.

7) Voting Classifier: In this experiment we look into utilizing a voting classifier to improve performance. While a VC is similar to an ensemble in principle, it is utilized quite differently in this implementation. While the ensemble components were trained on partitioned data, the ensemble as a whole is trained on the complete data. The VC is treated as an ensemble of completed models, each trained on their own complete data. In our experiments, the VC acts as an ensemble of ensembles. However, the VC has not used the ensembles exclusively. It has also utilized the monolithic models, so while such a description is intuitive, it is not entirely true.
To achieve fair testing of the model, we save the original monolith base models, and the *X_train* and *y_train* values. Since all models are trained on the same data split it ensures none of them have seen the data in the training split and the final test would remain fair. To bypass the limitation of the library we inject the pre-trained models and skip the fitting step. This saved on duplicate training time and ensured accurate comparisons.
Another benefit of reusing saved models was deciding on the models we wanted to combine within the *VotingClassifier*. A good approach would have been to simply take the best performing models and use them, but since we had the framework in place already we were able to test all possible unique combinations. Furthermore, to preserve the hyperparameter optimization, the combinations included only models optimized for the same metric, henceforth called the 'scorer'. Utilizing identically optimized component models would also suggest that the VC as a whole would be optimized for that specific metric. Additionally, to keep testing time reasonable, only 3 models were used per voting classifier. Since the initial results were already promising, and diminishing returns become more likely with additional models, we did not further explore this amount of classifiers.

8) AutoGluon: AutoGluon was trained and evaluated as a separate experimental track from our scikit-learn models. AutoGluon is a framework for supervised learning on tabular data that trains multiple candidate models and can optionally improve performance through internal ensembling. Notably, AutoGluon's reference pipeline (v1.4, `extreme` preset, 4-hour budget) is currently ranked #1 by Elo on the TabArena public leaderboard across the full 51-dataset IID suite.[3] We use it as a strong automated benchmark for our alarm classification task.

VII. Results

More results can be seen in Appendix B, however for the sake of comparison and readability, summary tables have been used in the following sections.

A. Monolith vs. Ensembles

The monolithic approach demonstrates the most balanced performance, securing the highest scores in *Precision Added*, *Accuracy*, and *FBeta*, as seen in Table VIII. When it comes for individual metrics, the results are balanced between the different approaches. Among the base classifiers, Random Forest (RFC) was the most volatile, appearing as both a top-tier performer (e.g., in *Precision Closed*) and a bottom-tier performer (e.g., in *Recall Closed* and *Accuracy*). This mostly depends on the eight optimization criteria.
The results within the ensembles follow a similar pattern. However, the values reach much more extreme polarities, as shown in Table VIII. This means that ensembles more often reach near-perfect scores in one metric while also performing with near-zero scores in others. For instance, the RFC with *rec_add* scorer reaching 1.000 in *Recall Added* but only 0.002 in *Recall Closed*.
This is further reinforced by comparing the group mean values per approach. As we can see in Table III mono achieves higher mean. If we isolate the ranking to only viable models then the difference become much lower as seen in Table IV, but still if we take the top 15 models, 11 are using the mono approach. Categorizing a model as viable is further explained in VII-C.

Approach	Mean Value	n
Mono	0.5205	64
Category Ensemble	0.3626	64
Employee Ensemble	0.2880	64

TABLE III: Ranking of approach by mean fbeta

Approach	Mean Value	n
Mono	0.6968	18
Employee Ensemble	0.6778	7
Category Ensemble	0.6637	4

TABLE IV: Ranking of approach by mean fbeta on viable models

This suggests model *over-optimization* which is further discussed in the following chapter. As shown in Table V, both the category ensemble (25.0%) and employee ensemble (23.4%)

[3]https://huggingface.co/spaces/TabArena/leaderboard

are prone to rates of over-optimization. This suggests the monolith approach is more reliable for maintaining balanced classification performance.

B. Over-optimized Models

During model selection and hyperparameter optimization, we can observe that a subset of runs converged to extreme highs and extreme lows for certain metrics. Such models have learned a prediction pattern that is dominated by a single class (or produces scores that yield near-zero discriminative results), which is very problematic under class imbalance as a model can appear competitive on aggregate metrics such as accuracy. Over-optimized models predict the majority class, while failing to identify the minority class of interest.

We can identify an over-optimized models if any of the following conditions hold:

- **Predicts almost all negatives:** $\text{recall}_{\text{pos}} \leq 0.01$ and $\text{recall}_{\text{neg}} \geq 0.95$
- **Predicts almost all positives:** $\text{recall}_{\text{neg}} \leq 0.01$ and $\text{recall}_{\text{pos}} \geq 0.95$
- **High accuracy but finds virtually no positives:** $\text{accuracy} \geq 0.83$ and $\text{recall}_{\text{pos}} \leq 0.01$
- **Near-zero positive precision:** $\text{precision}_{\text{pos}} \leq 0.05$
- **Near-zero F1 score:** $F_1 \leq 0.05$

These criteria are are not intended as a statistical test of performance, but as practical guardrails to identify models whose behaviour is dominated by a single class or whose positive-class performance is so poor that the model is effectively unusable. We therefore report *over-optimized rates* (Table V) as the fraction of trained models within each approach that triggered at least one of the above conditions.

C. Viable vs. Non-viable Non-overoptimized Models

While the over-optimized criteria in Appendix VII-B remove runs that are effectively dominated by a single class, many remaining models still fail to meet minimum operational requirements. To make this distinction explicit, we further partition *non-overoptimized* models into two groups based on recall constraints that reflect the intended prioritization use-case.

Viable models are defined as the *non-overoptimized* runs that satisfy both recall thresholds:

$$\texttt{recall}_{\texttt{added}} \geq 0.90 \quad \wedge \quad \texttt{recall}_{\texttt{closed}} \geq 0.10.$$

Non-viable models are the remaining *non-overoptimized* runs that miss at least one of these thresholds. Over-optimized models are excluded from both categories.

The asymmetry of these thresholds is deliberate. The primary requirement is high recall on *ADDED*, since false negatives are operationally costly. The weaker constraint on *CLOSED* ensures the model retains at least minimal discriminative ability on non-escalations, preventing a trivial "flag-everything" regime from being treated as operationally acceptable.

The resulting Viable rates are shown in Table VI. In absolute terms, the monolithic approach yields the largest pool of viable configurations (18/64), whereas the category and employee ensembles yield substantially fewer viable configurations (4/64 and 7/64, respectively). Put differently, even after excluding over-optimized runs, most trained models are still non-viable under these recall constraints (Category Ensemble: 44/48 non-overoptimized, Employee Ensemble: 42/49; Mono: 49/64). This indicates that simultaneously achieving very high escalation recall while retaining non-trivial recall on closed alarms is a relatively stringent requirement, and it is satisfied most consistently by the monolithic training regime.

Approach	Over-Optimized Rate (Fraction)	Over-optimized Count	Total Models
Category Ensemble	0.250 (16/64)	16	64
Employee Ensemble	0.234 (15/64)	15	64
Mono	0.000 (0/64)	0	64

TABLE V: Over-optimized Rates Across Approaches

Approach	Viable Rate (Fraction)	Viable Count	Non-overoptimized
Category Ensemble	0.083 (4/48)	4	48
Employee Ensemble	0.143 (7/49)	7	49
Mono	0.281 (18/64)	18	64

TABLE VI: Comparison of Viable Rates Across Approaches

		Recall		Precision			
Clf	Scorer	Added	Closed	Added	Closed	Accuracy	FBeta
LR	rec_add	**0.984**	0.205	0.192	0.985	0.330	0.696
KNC	prec_add	0.272	0.998	**0.971**	0.877	0.881	0.293
KNC	f1_add	0.480	**0.982**	0.837	0.908	**0.901**	0.501
DTC	rec_add	**0.183**	0.926	0.321	**0.855**	0.806	**0.191**
RFC	rec_add	1.000	**0.059**	**0.169**	1.000	**0.210**	0.671
RFC	fbeta_add	0.972	0.397	0.236	**0.986**	0.489	**0.741**

TABLE VII: Summary of minimum and maximum values per metric for the monolith method.

		Recall		Precision			
Clf	Scorer	Added	Closed	Added	Closed	Accuracy	FBeta
LR	rec_add	0.992	0.140	0.181	**0.989**	0.277	0.685
LR	rec_add	0.996	**0.016**	0.163	0.952	0.174	0.659
LR	prec_add	0.004	0.998	0.333	**0.839**	0.838	0.005
LR	fbeta_add	**0.988**	0.089	0.172	0.974	0.234	0.671
KNC	prec_add	0.175	0.982	0.652	0.861	**0.852**	0.189
KNC	prec_add	0.130	**0.983**	0.593	0.855	0.845	0.141
KNC	rec_closed	0.907	0.382	0.220	0.955	0.466	**0.691**
RFC	rec_add	1.000	0.002	**0.161**	1.000	**0.162**	0.658
RFC	prec_add	0.024	0.999	**0.857**	0.842	0.842	0.027
RFC	prec_closed	**0.012**	1.000	1.000	0.841	0.841	**0.014**

TABLE VIII: Summary of minimum and maximum values per metric for the two ensemble methods. Rows in blue correspond to the employee ensemble, and rows in green correspond to the category ensemble. Best and worst values in each column are highlighted in green and red, respectively.

After analyzing the results, we can go trough the models and isolated the better performing ones based on certain criteria. This helps us separate models that do not perform well or being

over-optimized during training. Over-optimized models try to satisfy the given loss function and sacrifices other metrics. For example if a model is optimized for recall on the positive label it might achieve perfect score there and a really bad score on the other metrics. Additionally, due to the imbalanced data, such models can still achieve good accuracy simply due to always predicting the majority class.

		Recall		Precision			
Clf	Scorer	Added	Closed	Added	Closed	Accuracy	FBeta
LR	rec_add	0.992	0.140	0.181	**0.989**	0.277	0.685
LR	rec_add	0.996	**0.016**	0.163	0.952	0.174	0.659
LR	fbeta_add	**0.988**	0.089	0.172	0.974	0.234	0.671
KNC	prec_add	0.272	0.998	**0.971**	0.877	0.881	0.293
KNC	f1_add	0.480	0.982	0.837	0.908	**0.901**	0.501
DTC	rec_add	**0.988**	0.051	0.167	0.956	0.202	0.662
RFC	rec_add	1.000	0.002	**0.161**	1.000	**0.162**	0.658
RFC	prec_closed	**0.012**	1.000	1.000	0.841	0.841	**0.014**
RFC	accuracy	0.439	**0.990**	0.893	0.902	**0.901**	0.463
RFC	fbeta_add	0.972	0.397	0.236	0.986	0.489	**0.741**
HGBC	prec_add	0.000	0.998	0.000	**0.839**	0.837	0.000

TABLE IX: Summary of minimum and maximum values per metric

D. Neural Network Results

Figures 2 and 3 showcase a summary of the results of the training of the neural networks.

For the RNN trained with BCE (Table X), the best trade-off between precision and recall for escalated cases is obtained at a threshold of 0.5. At this operating point, the model achieves a balanced F1 score while maintaining non-trivial recall for both escalated ($\texttt{recall}_{\texttt{added}} \approx 0.52$) and closed ($\texttt{recall}_{\texttt{closed}} \approx 0.54$) alarms. This indicates that BCE yields a relatively smooth score distribution, enabling balanced operating points that simultaneously capture escalations and avoid excessive false positives.

	Recall		Precision			
threshold	added	closed	added	closed	f1score	accuracy
0.050	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.100	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.150	0.935	0.064	0.162	0.836	0.277	0.205
0.200	0.925	0.080	0.163	0.846	0.278	0.218
0.250	0.919	0.088	0.164	0.848	0.278	0.223
0.300	0.914	0.094	0.164	0.849	0.278	0.227
0.350	0.909	0.101	0.164	0.851	0.278	0.233
0.400	0.898	0.114	0.164	**0.852**	0.278	0.241
0.450	0.527	0.523	0.177	0.851	0.265	0.524
0.500	0.516	0.539	**0.178**	0.851	0.265	0.535
0.550	0.022	0.962	0.100	0.835	0.035	0.809
0.600	0.016	0.970	0.094	0.835	0.028	0.815
0.650	0.011	0.977	0.083	0.836	0.019	0.820
0.700	0.011	0.980	0.095	0.836	0.019	0.823
0.750	**0.000**	0.996	**0.000**	0.837	**0.000**	0.834
0.800	**0.000**	0.997	**0.000**	0.837	**0.000**	0.835
0.850	**0.000**	0.997	**0.000**	0.837	**0.000**	0.835
0.900	**0.000**	0.999	**0.000**	0.837	**0.000**	**0.837**
0.950	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**

TABLE X: Threshold sweep for BCE loss on the test set (RNN)

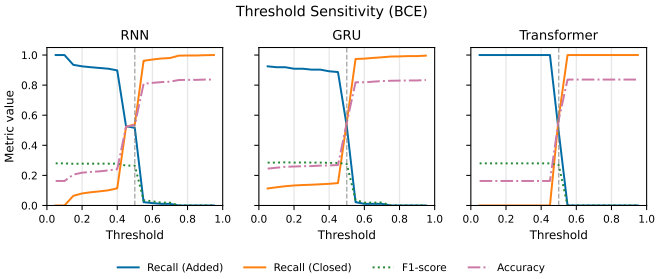

Fig. 2: Threshold sensitivity for RNN, GRU, and Transformer under binary cross-entropy loss. Each subplot shows recall (ADDED/CLOSED), F1-score, and accuracy across thresholds; the dashed line marks the standard 0.5 operating point.

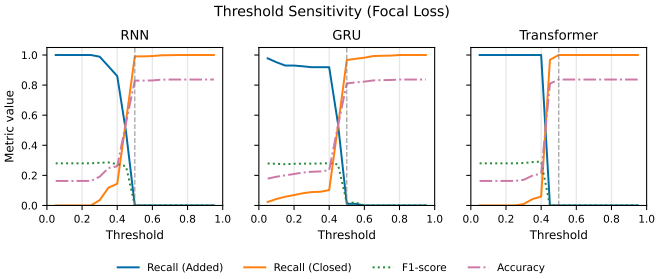

Fig. 3: Threshold sensitivity for RNN, GRU, and Transformer under focal loss. All architectures exhibit recall-dominant regimes at low thresholds with limited balanced operating regions. The dashed line marks the 0.5 threshold.

In contrast, the RNN trained with focal loss (Table XI) exhibits a strongly recall-dominated regime. At thresholds up to 0.35, the model achieves near-perfect recall for escalated cases ($\texttt{recall}_{\texttt{added}} \geq 0.925$), indicating that nearly all escalations are ranked above the decision threshold. However, this behaviour is accompanied by near-zero recall for closed cases and low overall accuracy, reflecting a deliberate prioritization of false-negative avoidance over balanced discrimination.

	Recall		Precision			
threshold	added	closed	added	closed	f1score	accuracy
0.050	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.100	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.150	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.200	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.250	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.300	0.989	0.035	0.166	**0.944**	0.284	0.191
0.350	0.925	0.118	0.169	0.890	**0.286**	0.249
0.400	0.860	0.144	0.163	0.841	0.274	0.260
0.450	0.489	0.562	**0.178**	0.850	0.261	0.550
0.500	**0.000**	0.991	**0.000**	0.836	**0.000**	0.830
0.550	**0.000**	0.991	**0.000**	0.836	**0.000**	0.830
0.600	**0.000**	0.993	**0.000**	0.836	**0.000**	0.831
0.650	**0.000**	0.998	**0.000**	0.837	**0.000**	0.836
0.700	**0.000**	0.999	**0.000**	0.837	**0.000**	**0.837**
0.750	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.800	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.850	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.900	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.950	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**

TABLE XI: Threshold sweep for focal loss on the test set (RNN)

Compared to the RNN, the GRU exhibits improved discrimination between escalated and closed alarms, yielding higher precision and recall for escalated cases at an operating point of 0.5 (Table XII). This suggests that the GRU's gating mechanisms improve the aggregation of temporal risk signals, resulting in more informative sequence representations.

	Recall		Precision			
threshold	added	closed	added	closed	f1score	accuracy
0.050	**0.925**	0.113	0.168	0.885	0.285	0.245
0.100	0.919	0.121	0.169	0.885	0.285	0.251
0.150	0.919	0.128	0.170	**0.891**	**0.287**	0.257
0.200	0.909	0.133	0.169	0.882	0.285	0.259
0.250	0.909	0.136	0.170	0.884	0.286	0.261
0.300	0.903	0.138	0.169	0.880	0.285	0.262
0.350	0.903	0.141	0.170	0.882	0.285	0.265
0.400	0.892	0.144	0.168	0.873	0.283	0.266
0.450	0.887	0.149	0.168	0.872	0.283	0.269
0.500	0.527	0.554	**0.187**	0.858	0.276	0.550
0.550	0.022	0.974	0.138	0.837	0.037	0.819
0.600	0.011	0.976	0.080	0.836	0.019	0.819
0.650	0.011	0.981	0.100	0.836	0.019	0.823
0.700	0.011	0.985	0.125	0.837	0.020	0.827
0.750	**0.000**	0.990	**0.000**	**0.836**	**0.000**	0.829
0.800	**0.000**	0.991	**0.000**	**0.836**	**0.000**	0.830
0.850	**0.000**	0.993	**0.000**	**0.836**	**0.000**	0.831
0.900	**0.000**	0.993	**0.000**	**0.836**	**0.000**	0.831
0.950	**0.000**	**0.996**	**0.000**	0.837	**0.000**	**0.834**

TABLE XII: Threshold sweep for BCE loss on the test set (GRU)

When trained with focal loss (Table XIII), the GRU exhibits a pronounced recall-oriented operation, once again achieving high recall for escalated cases at low thresholds ($\texttt{recall}_{\texttt{added}} \approx 0.925$ at threshold 0.25). Similarly, this behaviour is accompanied by poor recall for closed cases and limited flexibility at intermediate thresholds. While the GRU moderates the extreme score polarization, focal loss remains ill-suited for balanced operating points in this setting.

	Recall		Precision			
threshold	added	closed	added	closed	f1score	accuracy
0.050	**0.978**	**0.023**	0.163	0.846	0.279	0.178
0.100	0.952	0.043	0.162	0.820	0.277	0.191
0.150	0.930	0.058	0.161	**0.812**	0.274	0.200
0.200	0.930	0.069	0.162	0.835	0.277	0.209
0.250	0.925	0.081	0.163	0.848	0.278	0.219
0.300	0.919	0.089	0.164	0.850	0.278	0.224
0.350	0.919	0.091	0.164	0.853	0.279	0.226
0.400	0.919	0.102	0.166	**0.867**	**0.281**	0.235
0.450	0.543	0.528	**0.183**	0.856	0.273	0.531
0.500	0.011	0.967	0.059	0.834	0.018	0.811
0.550	0.011	0.975	0.077	0.835	0.019	0.818
0.600	**0.000**	0.982	**0.000**	0.835	**0.000**	0.823
0.650	**0.000**	0.993	**0.000**	0.836	**0.000**	0.831
0.700	**0.000**	0.995	**0.000**	0.837	**0.000**	0.833
0.750	**0.000**	0.996	**0.000**	0.837	**0.000**	0.834
0.800	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.850	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.900	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.950	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**

TABLE XIII: Threshold sweep for focal loss on the test set (GRU)

For the Transformer trained with binary cross-entropy (Table XIV), the results reveal a pronounced failure mode: at thresholds below 0.5, the model achieves perfect recall for escalated cases ($\texttt{recall}_{\texttt{added}} = 1.0$) but with zero recall for closed cases, while at thresholds ≥ 0.55 the pattern reverses completely with zero recall for escalated cases and perfect recall for closed cases. This extreme polarization suggests that the Transformer's score distribution has collapsed, with predictions clustered around a narrow threshold band. At 0.5, the model achieves moderate recall for both classes ($\texttt{recall}_{\texttt{added}} \approx 0.49$, $\texttt{recall}_{\texttt{closed}} \approx 0.59$), but this balanced regime is fragile and disappears with small threshold shifts. Unlike the recurrent architectures, the Transformer fails to maintain stable operating regimes under BCE, highlighting architecture-specific weaknesses in probability calibration.

	Recall		Precision			
threshold	added	closed	added	closed	f1score	accuracy
0.050	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.100	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.150	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.200	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.250	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.300	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.350	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.400	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.450	**1.000**	**0.000**	0.163	**0.000**	**0.280**	0.163
0.500	0.489	0.586	**0.186**	**0.855**	0.270	0.570
0.550	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.600	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.650	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.700	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.750	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.800	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.850	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.900	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.950	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**

TABLE XIV: Threshold sweep for BCE loss on the test set (Transformer)

Under focal loss (Table XV), the Transformer exhibits a recall-dominated regime similar to the recurrent models, achieving perfect recall for escalated cases at threshold 0.4 ($\texttt{recall}_{\texttt{added}} = 1.0$). While self-attention slightly moderates the collapse observed in recurrent architectures, focal loss still yields limited flexibility at intermediate operating points, reinforcing its suitability for recall-critical rather than balanced settings.

	Recall		Precision			
threshold	added	closed	added	closed	f1score	accuracy
0.050	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.100	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.150	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.200	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.250	**1.000**	**0.000**	0.163	**0.000**	0.280	0.163
0.300	**1.000**	0.009	0.164	**1.000**	0.282	0.170
0.350	**1.000**	0.042	0.168	**1.000**	0.288	0.198
0.400	**1.000**	0.059	**0.171**	**1.000**	**0.292**	0.212
0.450	**0.000**	0.968	**0.000**	0.833	**0.000**	0.810
0.500	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.550	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.600	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.650	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.700	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.750	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.800	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.850	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.900	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**
0.950	**0.000**	**1.000**	**0.000**	0.837	**0.000**	**0.837**

TABLE XV: Threshold sweep for focal loss on the test set (Transformer)

Across the recurrent architectures (RNN and GRU), BCE consistently supports balanced operating points near a threshold of 0.5, where both recall for escalated and closed cases remain non-trivial. However, the Transformer exhibits a failure mode under BCE, suggesting that self-attention mechanisms require careful calibration to maintain stable probability distributions. In contrast, focal loss enforces recall-dominant regimes regardless of model capacity, indicating that loss choice has a stronger influence on decision geometry than architectural complexity in recurrent models, while the Transformer demonstrates architecture-specific weaknesses that transcend loss function selection.

E. Voting Classifier Results

The performance of the VC is presented in Table XIX. It differs from the individual models as it is more balanced in terms of results with lower maximums but higher minimums across the metrics.

The prior over-optimization seen in MM, CE, and EE approaches is not present here due to the added diversity when combining models. By combining predictions, the VC filters out the extreme results, leading to a more stable performance in general.

However, despite this increased consistency, the VC showed lower results in Accuracy and F-beta, compared to the viable models found in the other methods.

F. AutoGluon Results

AutoGluon achieves the strongest performance among all evaluated approaches, as shown in Table XVI.

G. Summary of Findings

The experimental evaluation conducted in this study examines traditional machine learning classifiers, ensemble methodologies, voting classifiers, and neural network architectures. Among the traditional classifiers, monolithic models outperform both employee-based and category-based ensembles, with the LogisticRegression optimized for fbeta on the *ADD* label achieving an accuracy of 52.3% with overall balanced metrics and an f-beta score of 72.6% (see Table XVII).

The advantage of monolithic models relative to ensemble approaches can be attributed to the limited size of the available dataset, which renders further partitioning inefficient and prevents ensemble components from learning sufficiently robust patterns. The ensemble hypothesis, which is motivated by the conjecture that human decision making patterns may exhibit themselves within data partitioned per employee and thereby yield performance gains, does not materialize in practice. This outcome suggests that the data volume is insufficient to support such fine-grained specialization. This conclusion is reinforced by the over-optimization analysis, where employee-based and category-based ensembles contain more over-optimized models models than the monolithic approach, indicating weaker generalization under the same protocol.

The neural network experiments show that deep learning approaches achieve comparable performance to traditional machine learning methods on this limited dataset. Under binary cross-entropy loss at threshold 0.5, the best-performing neural architecture (GRU) achieves an accuracy of approximately 55% and an F1 score of 0.27, which is similar to the best traditional classifier (LogisticRegression at 52.3% accuracy). Focal loss does not improve overall performance: while it achieves high recall for escalated cases (e.g., 92.5% for the GRU at threshold 0.25), this comes at the expense of low recall for closed cases, yielding accuracy below 25% and F1 scores of approximately 0.28. The modest accuracy achieved by all neural architectures suggests that the limited dataset size constrains the ability of deep learning methods to learn robust representations.

Among the neural architectures, binary cross-entropy loss facilitates balanced operating points for recurrent models (RNN and GRU), where both recall for escalated and closed cases remain non-trivial near a threshold of 0.5. The GRU architecture exhibits marginally better discrimination between escalated and closed alarms relative to the standard RNN (F1 of 0.276 vs 0.265), potentially attributable to its gating mechanisms. The Transformer architecture exhibits unstable behavior under BCE, with predictions clustered around a narrow threshold band: at 0.5 it achieves moderate recall for both classes (≈ 0.49 for escalated, ≈ 0.59 for closed), but small threshold shifts cause complete collapse to single-class prediction. Focal loss consistently induces recall-dominant operational regimes across all architectures, achieving near-perfect or high recall for *ADDED* cases at low classification thresholds, albeit at the

expense of diminished precision and reduced flexibility at intermediate operating points.

The voting classifier experiments demonstrate that combining different base classifiers trained on identical data partitions can reduce extreme behaviors and improve stability. By systematically evaluating combinations of three models optimized for identical performance metrics, it is possible to leverage the complementary strengths of distinct algorithms while preserving the benefits derived from hyperparameter optimization. However, the voting classifier does not consistently outperform the strongest monolithic models on balanced metrics, and the best configurations can correspond to unfavorable trade-offs between *ADDED* and *CLOSED* performance.

Additionally, AutoGluon achieves the strongest overall performance on this dataset, including configurations with high accuracy and strong F-beta scores, while also maintaining comparatively better balance across class-wise metrics than most alternatives (see Table XV). While its results can vary across configurations, its best models are competitive across both performance and balance criteria.

In conclusion, the experimental results show that monolithic traditional machine learning models outperform data-partitioned ensemble methodologies and the evaluated neural network architectures on this dataset, and that AutoGluon achieves the best overall results.

VIII. Conclusion & Future Works

A. Conclusion

This paper set out to examine whether ensemble-based approaches built on employee and category data partitioning provide any advantages over monolithic models, voting-based combinations, or neural networks for use in post-alarm AML classification. It has been found that the partitioned ensemble approaches do not provide benefits on this dataset. In practice, they perform worse than monolithic training and also produce a higher share of over-optimized models. Voting classifiers try to mimic human behaviours by combining various base models, although the viable models show good performance they have a higher chance of being over optimized. Monoliths on average perform better than ensembles and have the best model between the employe ensemble, category ensemble and monoliths. Furthermore, the neural network experiments show that the evaluated architectures are performing great with the given data. Changes in loss functions change the results to more recall-heavy predictions, however, often at the expense of poorer performance on the other metrics. In contrast, AutoGluon shows the strongest overall performance across the reported results. As such, the findings support monolithic tabular modelling as a reliable baseline for this task, and AutoGluon as generally the most promising candidate for producing a binary screening decision that supports human review.

B. Future Work

To further investigate this topic, one could start with the following subjects:

a) Larger Voting Classifiers: The VCs constructed in this paper were all consisting of three models. One could explore how the performance of the classifiers change when utilizing a larger number of models within the VC.

b) Larger dataset: With more information from NetCompnany linking the transactions to the alarms might provide useful features to train on. Additionally with more time L&SB will accumulate more data.

c) Hyperparameter optimization: Explore further hyperparameter optimization such as Optuna, RandomSearch, multi-objective optimization and compare their results for possible gains.

IX. Acknowledgements

We would like to thank our partner, L&SB, for providing us with training data, enlightening us regarding the intricacies of AML in praxis, as well as giving us various comparison points that have acted as guidelines and goalposts for the performance and capabilities of our system.

Lastly, we would like to thank our supervisor, Christian Schilling, of the Department of Computer Science at Aalborg University, for his skilled supervision of this project.

References

[1] Finans Danmark. *Hvidvask*. URL: https://finansdanmark.dk/tal-og-data/institutter-filialer-ansatte/kriminalitet/hvidvask/ (visited on 12/18/2025).

[2] Flagright. *Understanding False Positives in Transaction Monitoring*. 2024. URL: https://www.flagright.com/post/understanding-false-positives-in-transaction-monitoring (visited on 01/06/2026).

[3] Rasmus Jensen and Alexandros Iosifidis. *Fighting Money Laundering with Statistics and Machine Learning*. 2023. arXiv: 2201.04207 [stat.ML]. URL: https://arxiv.org/abs/2201.04207.

[4] *Et indblik i udmålingen af bøder ved overtrædelse af hvidvaskloven*. URL: https://denmark.dlapiper.com/da/nyhed/et-indblik-i-udmaalingen-af-boeder-ved-overtraedelse-af-hvidvaskloven (visited on 12/13/2025).

[5] Berkan Öztaş et al. "Transaction Monitoring in Anti-Money Laundering: A Qualitative Analysis and Points of View from Industry". In: *Future Generation Computer Systems* 159 (2024), pp. 161–171.

[6] Zhiyuan Chen et al. "Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review". In: *Knowledge and Information Systems* 57 (2018). DOI: 10.1007/s10115-017-1144-z.

[7] Dattatray Vishnu Kute et al. "Deep Learning and Explainable Artificial Intelligence Techniques Applied for Detecting Money Laundering—A Critical Review". In: *IEEE Access* 9 (2021), pp. 82300–82317. DOI: 10.1109/ACCESS.2021.3086230.

[8] Jiani Fan et al. *Deep Learning Approaches for Anti-Money Laundering on Mobile Transactions: Review, Framework, and Directions*. 2025. arXiv: 2503.10058 [cs.LG]. URL: https://arxiv.org/abs/2503.10058.

[9] Martin Jullum et al. "Detecting money laundering transactions with machine learning". In: *Journal of Money Laundering Control* (2020). DOI: 10.1108/JMLC-07-2019-0055.

[10] Khashayar Namdar et al. *Anti-Money Laundering Machine Learning Pipelines: A Technical Analysis on Identifying High-risk Bank Clients with Supervised Learning*. 2025. arXiv: 2509.09127 [cs.AI]. URL: https://arxiv.org/abs/2509.09127.

[11] A. N. Bakry, A. S. Alsharkawy, M. S. Farag, et al. "Automatic Suppression of False Positive Alerts in Anti–Money Laundering Systems Using Machine Learning". In: *The Journal of Supercomputing* 80 (2024). Published 20 October 2023; Issue date March 2024, pp. 6264–6284. DOI: 10.1007/s11227-023-05708-z. URL: https://doi.org/10.1007/s11227-023-05708-z.

[12] Prashank Kadam. *Enhancing Financial Fraud Detection with Human-in-the-Loop Feedback and Feedback Propagation*. 2024. arXiv: 2411.05859 [cs.LG]. URL: https://arxiv.org/abs/2411.05859.

[13] R. C. Nagarakanti. "Human-AI Collaboration in Financial Services: Augmenting Decision-Making with Cloud-Native Intelligence". In: *European Journal of Computer Science and Information Technology* 13.22 (2025), pp. 23–41.

[14] George Fragiadakis et al. *Evaluating Human-AI Collaboration: A Review and Methodological Framework*. 2025. arXiv: 2407.19098 [cs.HC]. URL: https://arxiv.org/abs/2407.19098.

[15] Yang Liu et al. "Pick and Choose: A GNN-based Imbalanced Learning Approach for Fraud Detection". In: *Proceedings of the Web Conference 2021*. Association for Computing Machinery, 2021, pp. 3168–3177. DOI: 10.1145/3442381.3449989. URL: https://doi.org/10.1145/3442381.3449989.

[16] Alexey Ruchay et al. "The Imbalanced Classification of Fraudulent Bank Transactions Using Machine Learning". In: *Mathematics* 11.13 (2023), p. 2862. DOI: 10.3390/math11132862. URL: https://www.mdpi.com/2227-7390/11/13/2862.

[17] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: https://arxiv.org/pdf/1201.0490.

[18] David H. Wolpert and William G. Macready. "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893.

[19] Cynthia Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead". In: *Nature Machine Intelligence* 1 (2019), pp. 206–215. DOI: 10.1038/s42256-019-0048-x.

[20] Thomas M. Cover and Peter E. Hart. "Nearest Neighbor Pattern Classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[21] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[22] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451.

[23] 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*. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[24] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. 2017.

[25] Takaya Saito and Marc Rehmsmeier. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". In: *PLOS ONE* 10.3 (2015), e0118432. DOI: 10.1371/journal.pone.0118432.

[26] Jeffrey L. Elman. “Finding Structure in Time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211.

[27] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. 2014, pp. 1724–1734.

[28] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6000–6010.

[29] Ravid Shwartz-Ziv and Amitai Armon. “Tabular Data: Deep Learning Is Not All You Need”. In: *Advances in Neural Information Processing Systems*. 2022.

[30] Lionel Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. “Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?” In: *Proceedings of the National Academy of Sciences* 119.29 (2022). DOI: 10.1073/pnas.2122150119.

[31] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the Difficulty of Training Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on Machine Learning*. PMLR, 2013, pp. 1310–1318.

[32] Kyunghyun Cho et al. “Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[33] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *arXiv preprint arXiv:1412.3555* (2014). URL: https://arxiv.org/abs/1412.3555.

[34] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV]. URL: https://arxiv.org/abs/1708.02002.

APPENDIX

A. Proofs

1) Proof: Recall Added 100% = Precision Closed 100%: Proof of Recall added = Precision Closed when both are at 100%:

Claim: $\text{Recall}_{\text{Added}} = 1 \iff \text{Precision}_{\text{Closed}} = 1.$

Setup:

Consider a binary classification problem with classes $\{\text{Added}, \text{Closed}\}$. Let "Added" be the positive class and "Closed" be the negative class, and define the confusion matrix counts:

	$\widehat{\text{Added}}$	$\widehat{\text{Closed}}$
Added	TP	FN
Closed	FP	TN

The recall for the class Added is

$$\text{Recall}_{\text{Added}} = \frac{TP}{TP + FN}.$$

The precision for the class Closed (treating Closed as the class of interest) is

$$\text{Precision}_{\text{Closed}} = \frac{TN}{TN + FN}.$$

(Here, the denominator $TN + FN$ counts all predictions of $\widehat{\text{Closed}}$, of which TN are correct and FN are incorrect.)

Proof:

$$(\Rightarrow) \quad \text{Recall}_{\text{Added}} = 1 \implies \frac{TP}{TP + FN} = 1 \implies TP = TP + FN \implies FN = 0.$$

If $FN = 0$, then

$$\text{Precision}_{\text{Closed}} = \frac{TN}{TN + FN} = \frac{TN}{TN + 0} = 1.$$

Hence $\text{Recall}_{\text{Added}} = 1 \Rightarrow \text{Precision}_{\text{Closed}} = 1$.

$$(\Leftarrow) \quad \text{Precision}_{\text{Closed}} = 1 \implies \frac{TN}{TN + FN} = 1 \implies TN = TN + FN \implies FN = 0.$$

If $FN = 0$, then

$$\text{Recall}_{\text{Added}} = \frac{TP}{TP + FN} = \frac{TP}{TP + 0} = 1.$$

Hence $\text{Precision}_{\text{Closed}} = 1 \Rightarrow \text{Recall}_{\text{Added}} = 1$.

Conclusion: $\text{Recall}_{\text{Added}} = 1 \iff FN = 0 \iff \text{Precision}_{\text{Closed}} = 1.$ $\square$

B. Summary Tables

This appendix contains summary tables from the full data. Each table is a summary of a larger table, and contains only the rows exhibiting either a minimum or maximum value for each column, for better read- and comparability.
Rows shaded in light red correspond to `mono`, light blue to `employee_ensemble`, and light green to `category_ensemble`. Cells highlighted with dark green are the highest values in the column. Cells highlighted with dark red are the lowest values in the column.
The appearance sequence of the rows within the table is arbitrary, as it was dependant on the execution order during training.

1) Gluon:

	Recall		Precision			
Model	Added	Closed	Added	Closed	Accuracy	Fbeta
XGBoost_BAG_L2	0.943	**0.628**	**0.327**	0.983	**0.785**	**0.794**
WeightedEnsemble_L3	0.951	0.577	0.302	0.984	0.764	0.783
WeightedEnsemble_L2	**0.963**	0.546	0.290	**0.987**	0.755	0.782
LightGBM_BAG_L2	**0.939**	0.596	0.308	0.981	0.767	0.780
RandomForest_BAG_L1	0.955	0.550	0.289	0.985	0.752	0.777
RandomForest_BAG_L2	**0.963**	0.527	0.281	**0.987**	0.745	0.775
XGBoost_BAG_L1	0.951	0.440	0.246	**0.979**	0.696	0.739
LightGBM_BAG_L1	0.959	**0.377**	**0.228**	0.980	**0.668**	**0.727**

TABLE XVI: AutoGluon model performance using macro-averaged metrics.

2) Viable Models Tables:

		Recall		Precision			
Base Classifier	Scorer	Added	Closed	Added	Closed	Accuracy	FBeta
RF	rec_closed	**0.996**	**0.102**	0.175	**0.992**	0.246	0.679
LR	rec_add	0.992	0.140	0.181	0.989	0.277	0.685
DT	fbeta_add	0.992	0.130	0.179	0.988	0.269	0.683
HGBC	fbeta_closed	0.984	0.245	0.200	0.987	0.364	0.707
LR	rec_add	0.984	0.204	0.192	0.985	0.330	0.696
LR	rec_closed	0.976	0.123	0.176	0.963	0.260	0.671
RF	fbeta_add	0.972	0.397	0.236	0.986	0.489	**0.741**
RF	fbeta_closed	0.968	0.112	0.173	0.947	0.249	0.663
DT	fbeta_add	0.959	0.151	0.178	0.951	0.282	0.667
LR	rec_closed	0.955	0.103	**0.170**	0.923	**0.240**	0.653
KNC	rec_add	0.943	0.187	0.182	0.945	0.309	0.665
GNB	rec_add	0.939	0.262	0.196	0.957	0.371	0.681
GNB	prec_add	0.939	0.262	0.196	0.957	0.371	0.681
GNB	rec_closed	0.939	0.262	0.196	0.957	0.371	0.681
GNB	prec_closed	0.939	0.262	0.196	0.957	0.371	0.681
GNB	acc	0.939	0.262	0.196	0.957	0.371	0.681
GNB	f1_add	0.939	0.262	0.196	0.957	0.371	0.681
GNB	fbeta_add	0.939	0.262	0.196	0.957	0.371	0.681
GNB	fbeta_closed	0.939	0.262	0.196	0.957	0.371	0.681
RF	rec_closed	0.939	0.213	0.186	0.948	0.330	0.669
LR	fbeta_closed	0.939	0.119	0.170	**0.911**	0.252	**0.646**
LR	fb_add	0.931	**0.445**	**0.244**	0.971	**0.523**	0.726
DT	fbeta_closed	0.923	0.182	0.178	0.925	0.301	0.651
RF	fbeta_add	0.919	0.304	0.202	0.951	0.403	0.678
LR	fbeta_closed	0.911	0.293	0.198	0.945	0.393	0.670
KNC	rec_closed	**0.906**	0.382	0.220	0.955	0.466	0.691

TABLE XVII: Viable models with simple GridSearch optimization

		Recall		Precision			
Base Classifier	Scorer	Added	Closed	Added	Closed	Accuracy	FBeta
RF	rec_closed	**0.996**	**0.102**	**0.176**	**0.992**	**0.246**	0.679
LR	rec_add	0.992	0.140	0.181	0.989	0.277	0.685
DT	fbeta_add	0.992	0.130	0.180	0.988	0.269	0.683
HGBC	fbeta_closed	0.984	0.245	0.200	0.987	0.364	0.707
LR	rec_add	0.984	0.205	0.192	0.985	0.330	0.696
LR	rec_closed	0.976	0.123	0.176	0.963	0.260	0.671
RF	fbeta_add	0.972	**0.397**	**0.236**	0.986	**0.489**	**0.741**
RF	fbeta_closed	0.968	0.112	0.173	0.947	0.250	0.663
DT	fbeta_add	0.959	0.151	0.178	0.951	0.282	0.667
LR	rec_closed	**0.955**	0.103	**0.170**	**0.923**	**0.240**	**0.653**

TABLE XVIII: Top viable models with simple GridSearch optimization

3) Voting Classifier:

		Recall		Precision			
Base Classifier	Scorer	Added	Closed	Added	Closed	Accuracy	FBeta
DTC, GNB, LGBMC	rec_add	**0.547**	0.024	0.010	0.745	0.463	0.562
HGBC, LR, RFC	fbeta_closed	0.000	1.000	**0.161**	0.000	0.161	0.000
DTC, GNB, LR	rec_add	0.717	**0.016**	**0.011**	0.791	0.604	0.723
DTC, GNB, RFC	rec_add	**0.738**	0.016	0.012	**0.796**	**0.621**	**0.743**
LGBMC, RFC, XGBC	rec_add	0.010	0.638	0.110	0.127	0.111	**0.011**
KNC, RFC, XGBC	rec_closed	0.001	0.642	0.110	**0.011**	0.104	0.001
HGBC, LR, XGBC	fbeta_closed	0.000	**0.972**	0.157	0.000	0.157	0.000
DTC, KNC, LGBMC	f1_add	0.009	0.150	0.028	0.054	**0.032**	0.010

TABLE XIX: Summary of minimum and maximum values per metric for the vote classifier