Label Unbalance in High-frequency Trading

Zijian Zhao * †, Xuming Chen * ‡, Jiayu Wen * §, Mingwen Liu *, Xiaoteng Ma * ¶, Zhipeng Liang * ‖
* Likelihood Lab † Sun Yat-sen University ‡ Peking University
§ University College London ¶ Tsinghua University ‖ The Hong Kong University of Science and Technology

***Abstract*—In financial trading, return prediction is one of the foundation for a successful trading system. By the fast development of the deep learning in various areas such as graphical processing, natural language, it has also demonstrate significant edge in handling with financial data. While the success of the deep learning relies on huge amount of labeled sample, labeling each time/event as profitable or unprofitable, under the transaction cost, especially in the high-frequency trading world, suffers from serious label imbalance issue. In this paper, we adopts rigurious end-to-end deep learning framework with comprehensive label imbalance adjustment methods and succeed in predicting in high-frequency return in the Chinese future market. The code for our method is publicly available at https://github.com/RS2002/Label-Unbalance-in-High-Frequency-Trading.**

***Index Terms*—High-frequency Trading, Deep Learning, Label Unbalance**

I. Introduction

In high-frequency trading (HFT), sophisticated algorithms make rapid trading decisions based on large volumes of financial data within milliseconds. These decisions often hinge on predictive models that identify favorable trading opportunities.

However, a significant challenge in developing these predictive models is the issue of label imbalance, where certain outcomes or events (labels) are far less frequent than others. For instance, predicting rare events like sudden market crashes or abrupt price changes can be critical for high-frequency traders, but the scarcity of these events in historical data skews the distribution of labels. This imbalance complicates model training, as standard machine learning algorithms may become biased towards the more frequent labels, leading to poor generalization and a higher risk of substantial financial losses. Addressing label imbalance is crucial to enhance the robustness and reliability of predictive models in high-frequency trading, ultimately driving profitability and minimizing risk exposure in fast-paced market environments.

II. Related Work

Rare events are events that happen much less frequently compared to common events. In the field of data mining, identifying rare events is typically a classification problem. Due to their infrequency and casual nature, rare events are challenging to detect, and misclassifying them can lead to significant costs. For instance, mislabeling a return in high-frequency trading (HFT) can result in incorrect buying or selling decisions, leading to profit loss. The scarce occurrences of rare events make the detection task an imbalanced data classification problem. Imbalanced data refers to a dataset in which one or more classes have a significantly greater number of examples than the others. The most prevailing class is called the majority class, whereas the rarest class is called the minority class, typically representing the concept of interest. Although data mining methods are widely used to develop classification models for guiding business and managerial decision-making, the classification of imbalanced data poses significant challenges to traditional classification models. Since most standard classification algorithms, such as logistic regression, support vector machine (SVM), and decision trees, are designed for balanced training sets, they may produce suboptimal classification models, leading to good coverage of the majority examples but frequent misclassification of minority ones. As a result, algorithms that perform well in standard classification frameworks do not necessarily deliver the best performance for imbalanced datasets [1]. There are several reasons for this behaviour [2]:

1) The use of global performance measures, like the standard accuracy rate, to guide the learning process may bias the results towards the majority class.
2) Classification rules that predict the positive class are often highly specific and have very low coverage. This leads to the exclusion of the class in favour of more general rules that predict the negative class.
3) Very small clusters of minority class examples can be identified as noise, which could be falsely discarded by the classifier. On the contrary, noise may be falsely identified as minor examples, as both are rare patterns in the sample space.

The machine learning community has shown significant interest in addressing the imbalanced learning problem in recent years. Over the past decade, various machine learning approaches have been developed to tackle imbalanced data classification. These approaches primarily rely on preprocessing techniques, cost-sensitive learning, and ensemble methods [1]–[3]. Below, we will delve into the details of these three methods:

A. Preprocessing techniques

Preprocessing is commonly conducted before constructing a learning model to improve the quality of input data. Two classical techniques are frequently utilised as preprocessors:

1) Resampling: Resampling techniques are implemented to rebalance the sample space for imbalanced datasets, aiming to mitigate the impact of the skewed class distribution during the learning process. They can be categorised into three groups based on the method used to balance the class distribution [1]:

Corresponding Email: zhaozj28@mail2.sysu.edu.cn

- Over-sampling methods: these mitigate the adverse effects of skewed distribution by generating new minority class samples. Two widely used methods for creating synthetic minority samples are randomly duplicating the minority samples and employing the 'Synthetic Minority Oversampling Technique' (SMOTE).
- Under-sampling methods: these address the issues arising from skewed distribution by discarding samples from the majority class. Random Under Sampling (RUS) is the simplest yet highly effective method, which involves randomly eliminating majority class examples.
- Hybrid methods: these techniques combine over-sampling and under-sampling methods.

In studies comparing the effectiveness of various re-sampling methods, important insights were gained regarding the selection of re-sampling methods [1], [4], [5]. When dealing with datasets containing hundreds of minority observations, it was observed that an under-sampling method outperformed an over-sampling method in terms of computational time. However, in scenarios where only a few dozen minority instances were present, the over-sampling method SMOTE was found to be the preferable option. When the training sample size is excessively large, a hybrid approach involving SMOTE and under-sampling is recommended as an alternative. Finally, SMOTE exhibits a slightly higher efficacy in detecting outliers.

2) Feature selection and extraction: In general, the aim of feature selection is to choose a subset of k features from the entire feature space. This subset should enable a classifier to achieve optimal performance, where k is a user-specified or adaptively selected parameter. Feature selection can be categorised into filters, wrappers, and embedded methods [1].

Another approach to address dimensionality is feature extraction, which corresponds to dimensionality reduction and involves transforming data into a low-dimensional space. However, it's important to note that feature selection techniques are distinct from feature extraction. While feature selection returns a subset of the original features, feature extraction generates new features from the original ones using functional mapping. There are various techniques for feature extraction, such as Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and Non-negative Matrix Factorization (NMF). Feature extraction methods are more commonly used for unstructured data like images, text, and speech [1].

It was observed in the related papers that filter and wrapper feature selection methods were the most frequently utilised. Regarding filter methods, a variety of metrics were employed to rank the features, while heuristic search was a common choice for wrapper methods. It is also found the frequent use of feature selection and extraction for addressing real-world problems such as disease diagnosis, textual sentiment analysis, fraud detection, and other rare event detection problems [1].

B. Cost-sensitive learning

Cost-sensitive learning considers the varying costs of misclassifying different classes. Cost matrices are commonly used

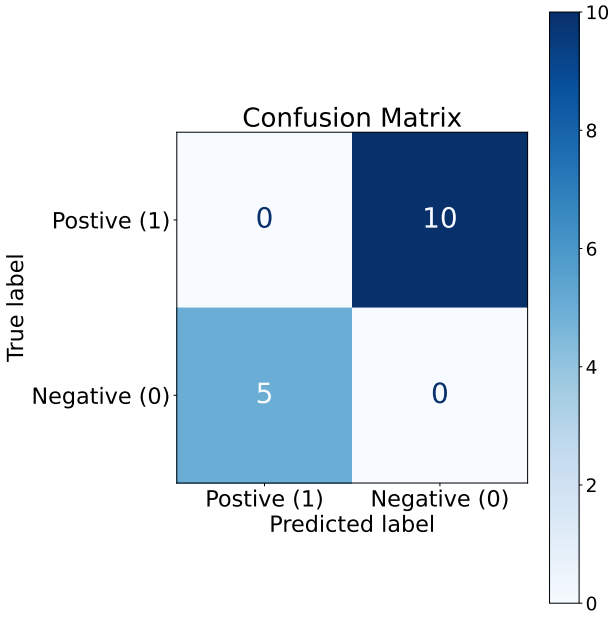

Fig. 1. Sample Cost Matrix

to define these costs, with C_{ij} indicating the cost of misclassifying examples from class i as class j. In specific fields, domain experts can determine cost matrices, resulting in a fixed cost matrix. Alternatively, in data stream scenarios, these matrices may vary at each optimisation loop step based on the algorithm's efficiency, which is referred to as an adaptive cost matrix [1], [2], [6]. For example, a fixed cost matrix with binary classes $\{0, 1\}$, corresponding to the first condition, is depicted in Fig. 1. The cost of misclassifying a sample to class 0 is 10, while the cost of misclassifying to class 1 is 5. The reason for the higher cost of misclassifying to class 0 is that our primary interest lies in the minority class, which we have defined as the positive class (1).

By assuming higher costs for incorrectly classifying samples from the minority class compared to the majority class, cost-sensitive learning can be integrated at both the data level (such as re-sampling and feature selection) and the algorithmic level. At the algorithmic level, the primary concept is to establish direct cost-sensitive learning, which involves incorporating and utilising misclassification costs into the learning algorithms. Regarding the data level, cost-sensitive learning concerns the incorporation of a 'preprocessing' mechanism for the training data or a 'postprocessing' of the output in a manner that does not alter the original learning algorithm [2]. This category can be further divided into two primary categories: thresholding and sampling:

- **Thresholding** operates on the principle of basic decision theory, which involves assigning instances to the class with the lowest expected cost. For example, in a binary classification problem, a typical decision tree assigns the class label of a leaf node based on the majority class

of the training samples that reach the node. A cost-sensitive algorithm designates the class label to the node that minimizes the classification cost.

- **Sampling** is associated with modifying the training dataset. The most common technique is to adjust the original class distribution of the training dataset based on the cost decision matrix through under-sampling/over-sampling or assigning instance weights.

Guo Haixiang et al. [1] provided a comprehensive overview of the cost-sensitive learning methods published in the last decade in Table I. They also concluded that in comparison to resampling methods, cost-sensitive learning is more computationally efficient and may be better suited for handling big data streams. However, they noted that this approach is significantly less popular than resampling methods. One potential explanation is that resampling is a more common choice for researchers not well-versed in machine learning. Unlike cost-sensitive learning, which often requires modification of the learning algorithm, resampling methods are much simpler to implement directly in both single and ensemble models.

C. Ensemble methods

Ensemble-based classifiers, also known as multiple classifier systems, are recognised for enhancing the performance of a single classifier by combining multiple base classifiers that individually outperform it. Ensembles of classifiers have emerged as a potential solution to address the imbalanced data classification problem. Ensemble-based methods involve a blend of ensemble learning algorithms and one of the previously discussed techniques, namely data preprocessing ensembles or cost-sensitive learning solutions. When a data-level approach is added to the ensemble learning algorithm, the new hybrid method typically preprocesses the data before training each classifier. On the contrary, cost-sensitive ensembles guide the cost minimisation procedure through the ensemble learning algorithm instead of modifying the base classifier to accommodate costs during the learning process.

A comprehensive taxonomy for ensemble methods for learning with imbalanced classes can be referenced in [7], as summarised in Fig. 2. The authors categorise four distinct families among ensemble approaches for imbalanced learning. They identify cost-sensitive boosting approaches, which are similar to cost-sensitive methods, but with the cost minimisation procedure guided by a boosting algorithm. Additionally, they distinguish three more families sharing a common feature: all embed a data preprocessing technique in an ensemble learning algorithm. These three families are categorised based on the ensemble learning algorithm used, i.e. boosting, bagging, and hybrid ensembles. According to the study, the authors concluded that ensemble-based algorithms are worthwhile because they surpass the performance of solely employing preprocessing techniques prior to training the classifier.

III. Preliminary

Given the input data $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where each $\mathbf{x}_i \in \mathbb{R}^d$ is a feature vector representing the state of the market at time i. The feature vector may include various indicators, such as Price changes (ΔP), Volume traded (V), Bid-ask spread (S), Order book imbalance (OBI), Volatility (σ).

Each instance $\mathbf{x}_i$ has a corresponding label $y_i \in \{1, 2, \ldots, K\}$, where K is the number of possible outcomes or classes. For example, $K = 2$ might represent a binary classification task, such as predicting whether the price will go up or down in the next time interval. In particular, we consider the three-class classification problem in the horizon 1 min, where

$$y_i = \begin{cases} 1 & \text{if } R_i > \text{fee} \\ -1 & \text{if } R_i < -\text{fee} \\ 0 & \text{if } |R_i| \leq \text{fee} \end{cases}$$

where:

- R_i is the forward 1 minute return for instance i.
- fee is the trading fee.

Objective. The objective is to learn a classification function $f : \mathbb{R}^d \to \{1, 2, \ldots, K\}$ that maps each input vector $\mathbf{x}_i$ to its corresponding label y_i. This function f is often parameterized by a model θ, such as a logistic regression, neural network, or other machine learning model.

Note that in the short horizon such as 1 min, most of the return can not cover the trading fee, which results in the case where most of the label y_i takes value of 0, i.e., highly unbalanced label.

IV. Methodology

A. Overview

As shown in Fig. 3, our method consists of three main phases during training: data processing, training, and validation.

(1) Data Processing: Details regarding the data structure of our dataset and the data processing methods are outlined in Section V-A. In summary, we represent the data features as a 13-dimensional vector for each second and utilize the preceding 60 seconds of data to predict the current return, which is categorized into three classes: -1, 0, and +1. However, we identified a significant label imbalance issue in the dataset, with approximately 80% of samples belonging to class 0, while classes +1 and -1 each account for only about 10%. To address this, we implemented strategies for tackling long-tail distribution challenges during the training phase. The dataset is then divided into training, validation, and test sets in a 8:1:1 ratio. To prevent information leakage, we split the dataset chronologically rather than through random sampling. Lastly, we offer an optional normalization operation for each sample:

$$\begin{aligned} \mu^{(j)} &= \frac{\sum_{i=1}^{60} x_i^{(j)}}{60} , \\ \sigma^{(j)} &= \sqrt{\frac{\sum_{i=1}^{60} (x_i^{(j)} - \mu^{(j)})^2}{60}} , \\ \text{Norm}(x_i^{(j)}) &= \frac{x_i^{(j)} - \mu^{(j)}}{\sigma^{(j)}} , \end{aligned} \tag{1}$$

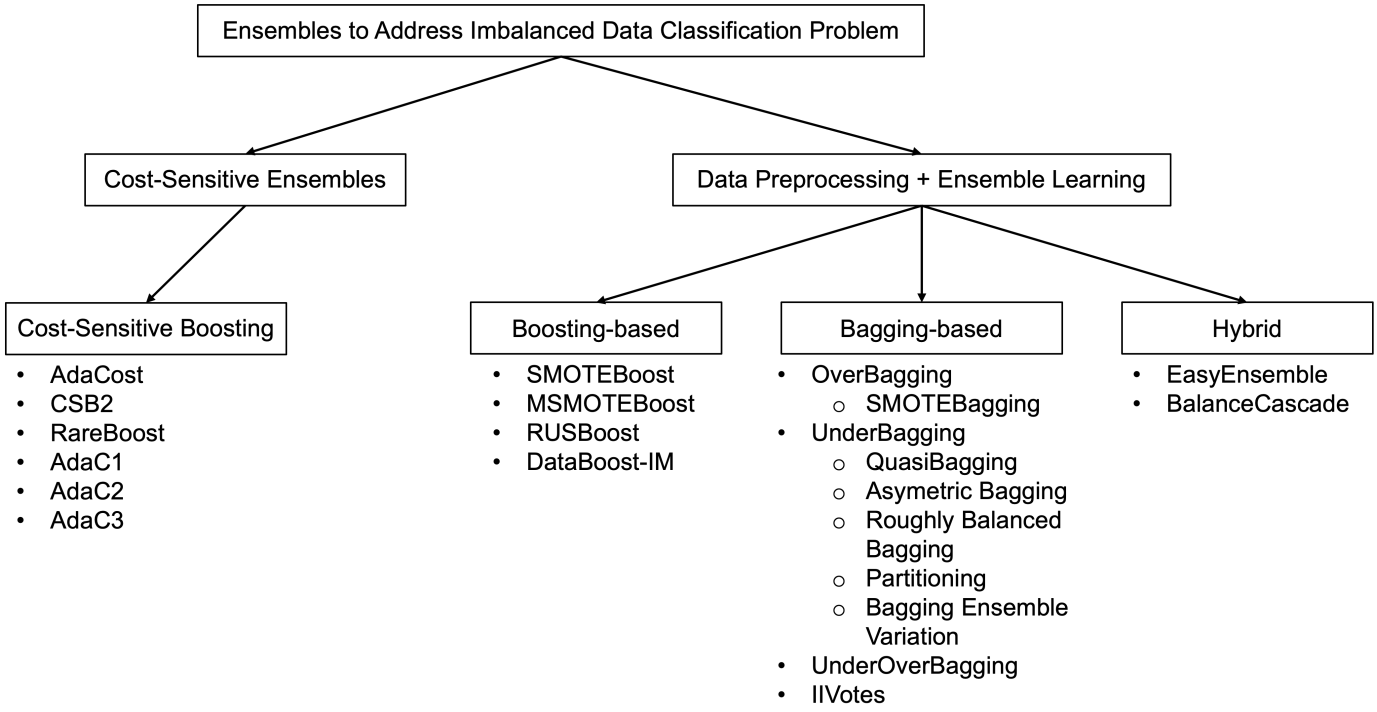

Fig. 2. Galar et al.'s proposed taxonomy for ensembles to address the imbalanced data classification problem

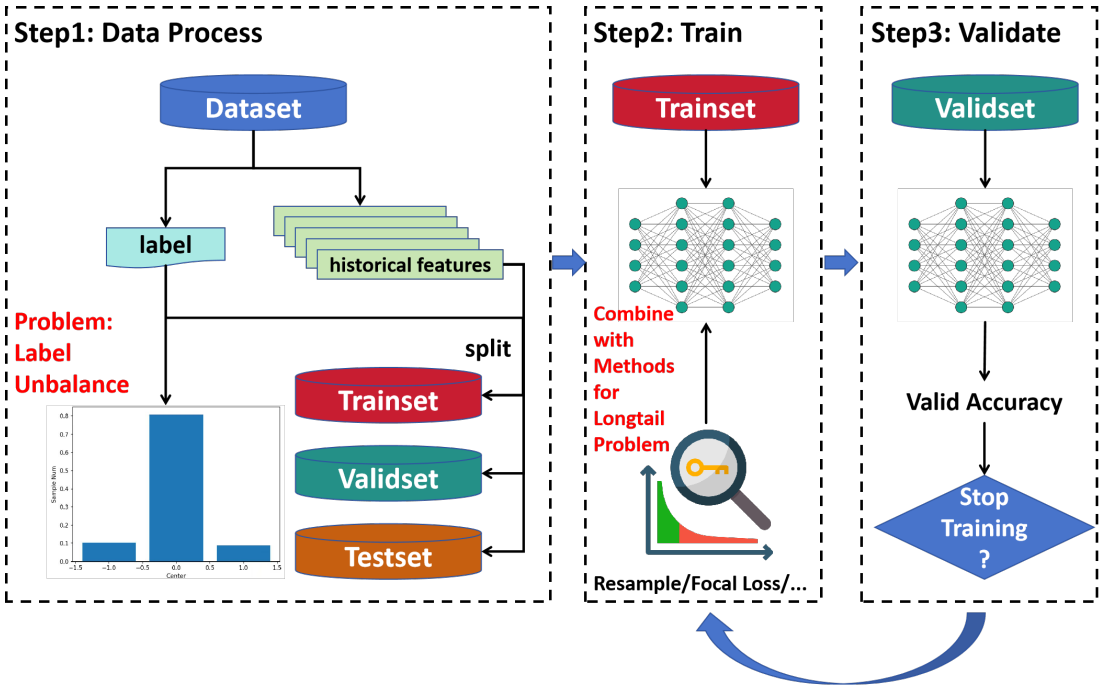

Fig. 3. Workflow of Proposed Method

TABLE I
SUMMARY OF COST-SENSITIVE LEARNING METHODS

Method	Detail Strategy
Methods based on training data modification	Modifying the decision thresholds or assigning weights to instance when resampling the training dataset according to the cost decision matrix
Changing the learning process or learning objective to build a cost-sensitive classifier	Modifying the objective function of SVM/ELM using a weighting strategy
	Tree-building strategies that could minimise misclassification costs
	Integrating a cost factor into the fuzzy rule-based classification system
	Cost sensitive error function on neural network
	Cost-sensitive boosting methods
Methods based on Bayes decision theory	Incorporating cost matrix into Bayes based decision boundary

where $x_i^{(j)}$ represents the j^{th} dimension of the i^{th} position in sample x, μ and σ represent the mean and standard deviation, respectively. Although this normalization has proven effective in mitigating covariate shift and enhancing model generalization across various fields [8], [9], we observe significant variance in our experiments. Its impact appears to differ across neural networks and methods for addressing label imbalance. At this stage, we cannot definitively categorize its effect as beneficial or detrimental in our task, warranting further investigation.

(2) Training: During this phase, we utilize the training set to train a neural network, incorporating strategies to mitigate label imbalance. We will elaborate on our network architecture and approaches to address label imbalance later in this section.

(3) Validation: To determine model convergence, we employ a standard early stopping mechanism. We monitor the model's accuracy at the end of each epoch and cease training when the accuracy does not improve over several consecutive epochs.

B. Backbone Models

1) Multilayer Perceptron (MLP): A Multilayer Perceptron (MLP) is a type of neural network that consists of multiple layers, including an input layer, hidden layers, and an output layer, where each layer is made up of perception nodes (neurons). The nodes in the hidden layers are referred to as activations. Fig. 4 shows an MLP featuring two hidden layers, along with an input and an output layer. An activation j in layer $d+1$ can be formulated as:

$$\begin{aligned} A_j^{(d+1)} &= h_j^{(d+1)}(\boldsymbol{X}) \\ &= g(w_{j0}^{(d+1)} + \sum_i w_{ji}^{(d+1)} A_i^{(d)}) \end{aligned} \tag{2}$$

where $g(z)$ is a nonlinear activation function specified beforehand, $\boldsymbol{X}$ is the input vector of p variables, and the superscript notation indicates to which layer the neuron and weights (coefficients) belong. Since each activation is directly a function of the activations $A_i^{(d)}$ from the previous layer d down to the input layer, they are inherently a function of the input vector $\boldsymbol{X}$. Through successive transformations, the network can create fairly intricate transformations of $\boldsymbol{X}$ that ultimately serve as features fed into the output layer.

After the transformations, we proceed to the output layer. Let Z_m denote the mth possible output, which is computed similarly to the activations. If the output Y is continuous, we simply set $\hat{Y} = Z_m$. For categorical responses, however, we apply activation functions such as the sigmoid function to produce the output $\hat{Y}_m$ [10].

In our implementation, both the activation functions in the hidden layers and the output layer utilise the LeakyReLU function, defined as follows:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases} \tag{3}$$

where negative_slope is a hyperparameter.

2) Long Short-Term Memory (LSTM): Recurrent Neural Networks (RNNs) are specifically created to handle and leverage sequential input data, such as financial time series data. Let $X = \{X_1, X_2, \ldots, X_T\}$, where each vector X_t in the input sequence contains p components and the hidden layer has K units. The activation can be expressed as:

$$A_{tk} = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{tj} + \sum_{s=1}^{K} u_{ks} A_{t-1,s}) \tag{4}$$

where w_{kj} are weights for input layer, and u_{ks} are the weights for the hidden-to-hidden layers. The resulting output is given by:

$$O_t = \beta_0 + \sum_{k=1}^{K} \beta_k A_{tk} \tag{5}$$

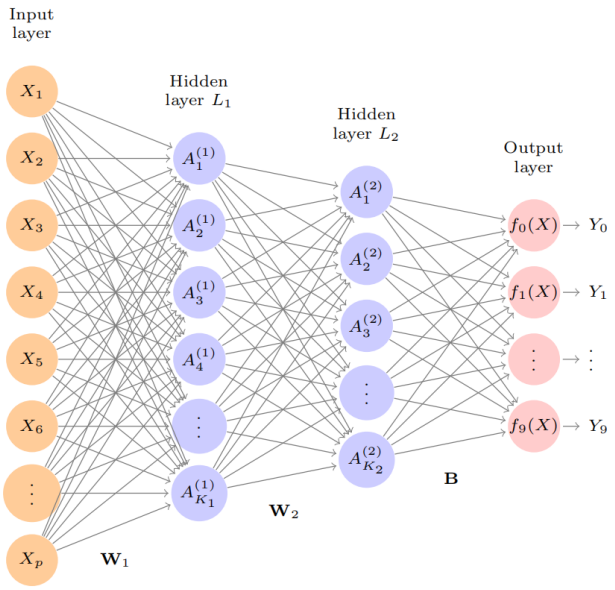

Fig. 4. Enter Caption

where weights β_k are the weights corresponding to the output layer, used for a quantitative prediction or with an additional sigmoid function for a binary outcome. The weights w_{kj}, u_{ks} and β_k are consistently applied as each element of the sequence is processed, exemplifying the weight sharing characteristic inherent to RNNs [10].

Conventional RNNs have difficulty learning tasks that require long-term dependencies because of issues with vanishing gradients and the decay of error flow. LSTMs provide a novel architecture that preserves a consistent error flow and tackles the vanishing gradient issue by using specialized units (memory cells) that retain their internal state over time, enabling them to "remember" information for extended periods. From Figure 5, we can see the layout of a memory cell. Three main components oversee the functioning of the memory cells [11]:

1) Input Gate: Determines when new input is introduced to the memory cell, denoted by i.
2) Forget Gate: Decides when to erase or reset the cell's state, denoted by f.
3) Output Gate: Controls when the state of the memory cell is sent out to the rest of the network, denoted by o.

If we denote the collection of weights w_{kj} and u_{ks} as $\mathbf{W}$ and $\mathbf{U}$ respectively, we can express the three components as follows:

$$\begin{aligned} i_t &= \sigma(\mathbf{W}x_t + \mathbf{U}h_{t-1}) \\ f_t &= \sigma(\mathbf{W}x_t + \mathbf{U}h_{t-1}) \\ o_t &= \sigma(\mathbf{W}x_t + \mathbf{U}h_{t-1}) \end{aligned} \tag{6}$$

where $\sigma()$ is the sigmoid function, x_t is the input vector at sequence t, and h_{t-1} is the hidden state at sequence $t-1$. The relationship between the cell's state c_t and the three gates is

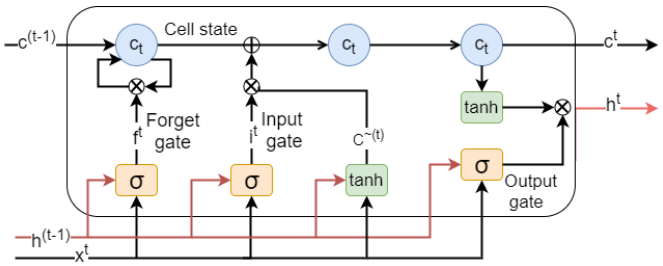

Fig. 5. Long Short Term Memory (LSTM) cell

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(\mathbf{W}x_t + \mathbf{U}h_{t-1}) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{7}$$

3) Mamba: Time series processing has long been a prominent topic in deep learning. Following the development of RNN [12], GRU [13], and LSTM [11], Transformer [14] has garnered significant attention for time series tasks due to its attention mechanism, which effectively captures relationships between any two positions in a sequence. However, there is still considerable debate regarding its effectiveness. For instance, some researchers argue that the Transformer struggles to recognize order due to its positional invariance [15].

Recently, Mamba [16], developed on structured State Space Sequence Models (SSMs) [17], [18], has gained widespread attention and is seen as a potential replacement for the Transformer. In traditional SSMs, the workflow can be expressed as:

$$\begin{aligned} h'(t) &= Ah(t) + Bx(t) \\ y(t) &= Ch(t) \end{aligned} \tag{8}$$

where A, B, C are model parameters, $x(t), y(t)$ represent the model's input and output, $h(t), h'(t)$ denote the hidden state before and after updating, and t indicates time. Mamba enhances this process through a selection mechanism, allowing A, B, C to depend on the input by utilizing a neural network, rather than being fixed matrices. This approach is quite similar to the attention mechanism in Transformers: for different tokens, Mamba employs distinct processing patterns for the hidden state, which incorporates historical information. Compared to Transformer, Mamba offers faster inference speeds and demonstrates excellent performance across various sequence-level tasks, such as audio processing and NLP.

C. Under-sampling

The poor performance of models trained directly on the full dataset arises from an unbalanced data distribution, which causes the model to focus more on the class with a larger number of samples. Therefore, our initial approach involves the implementation of under-sampling techniques to achieve equilibrium among the labels. In our dataset, the proportions of classes -1, 0, and +1 are approximately 1:8:1. Consequently, in each epoch, we randomly drop $\frac{1}{8}$ of the samples from class 0 in each epoch.

D. Cost-sensitive Learning

1) Fixed Cost Matrix: In the context of addressing label imbalance, cost-sensitive learning at the algorithmic level aims to focus the model's attention on classes with fewer samples, similar to the intuition behind resampling methods. Initially, we employed a straightforward method, direct loss weighting, to tackle the label imbalance issue. The loss for a sample can be expressed as:

$$L = \sum_{c=1}^{C} w_c l_c \tag{9}$$

where C is number of classes, l_c is the loss function for samples from class c, and w_c is the assigned weight. For our specific task, we designated the weights for classes -1, 0, and 1 as 8.0, 1.0, and 8.0, respectively. Intuitively, this method serves a similar process to resampling.

Moreover, we adjusted the weights based on the specific size of each class. We incorporated cost into the mean square error (MSE) for each class, building on Castro and Braga's research on imbalanced binary labels [19]. The loss function for each sample can be defined as:

$$L = \sum_{c=1}^{C} \frac{N_{-c}}{(C-1)\cdot N}(1-p_c)^2 l_c \tag{10}$$

where N_{-c} is the number of samples other than class c, and p_c is the probability that the sample is in class c. The cost is normalised by $C-1$ to ensure that $\sum_{c=1}^{C} \frac{N_{-c}}{(C-1)\cdot N} = 1$. Although fixed, the cost is aligned with the true distribution of the labels rather than being assigned intuitively.

2) Adaptive Cost Matrix: Focal loss [20] is a method akin to loss weighting, but it incorporates a dynamic cost matrix for each sample. It can be represented as:

$$\begin{aligned} FocalLoss(\boldsymbol{p}, y) &= -\sum_{c=1}^{C}(1-p_c)^{\lambda} log(p_c)\cdot \mathbf{1}\{y=c\} \\ CrossentropyLoss(\boldsymbol{p}, y) &= -\sum_{c=1}^{C} log(p_c)\cdot \mathbf{1}\{y=c\} \end{aligned} \tag{11}$$

where $\boldsymbol{p}$ is the vector output of the probabilities for each class by the model, y is the ground truth, C is the number of classes, λ is a control parameter, and $\mathbf{1}\{y=c\}$ is the indicator function. Compared to traditional cross-entropy loss, focal loss assigns greater weights to samples for which the model has low confidence (p_c) in the correct class. This adaptive weight $(1-p_c)^{\lambda}$ is the cost of misclassifying a sample to another class. The approach is particularly relevant in scenarios with imbalanced labels, where the model typically exhibits lower confidence for classes with fewer samples. Moreover, the dynamic nature of the costs during training ensures they remain relevant throughout the process.

What's more, [21] also propose a similar loss function:

$$\begin{aligned} w_c &= \frac{\frac{1}{a_c}}{\sum_{c=1}^{C}\frac{1}{a_c}} \\ L &= \sum_{c=1}^{C} w_c l_c \end{aligned} \tag{12}$$

where a_c is the average accuracy of the c^{th} class.

V. Experiment

A. Dataset Description

Our data spans from May 4th, 2023, to May 29th, 2023, covering a total of 20 trading days of high-frequency futures data, with a frequency of 0.5 seconds. The dataset includes six varieties: rebar (rb2310), silver (ag2308), fuel oil (fu2309), nickel (ni2306), tin (sn2306), and gold (au2308). Please note that each variety has different trading hours, resulting in varying sample sizes. Silver and gold have larger sample sizes, while rebar and fuel oil have smaller ones.

The raw data contains key information such as trading time, daily prices (opening, closing, highest, and lowest), latest transaction price ($lastPrice$), cumulative transaction amount, cumulative transaction volume, and the price and volume of buy and sell orders at the first to fifth trading positions ($bidPrice_i$ and $askPrice_i$, where $i = 1,2,3,4,5$, which represents 5 positions respectively). Based on this data, we constructed 13 variables to predict returns. These variables include:

- $midPrice$, which is the average of the best buy (buy one) and sell (sell one) prices at each data point.
- $diffBidPrice_i$ and $diffAskPrice_i, i = 1,2,3,4,5$, which are the price differences between each level and the average $midPrice$ at each data point.
- $diffLastPrice$, which is the price difference between the latest transaction price $lastPrice$ and the average $midPrice$.
- $logVolume$, which is the logarithm of transaction volume $volume$ in the past 0.5 seconds (if there is no transaction, it is recorded as 0).

The calculation formula is as follows:

$$\begin{aligned} midPrice &= \frac{bidPrice_1 + askPrice_1}{2}, \\ diffBidPrice_i &= bidPrice_i - midPrice, i = 1,2,3,4,5, \\ diffAskPrice_i &= askPrice_i - midPrice, i = 1,2,3,4,5, \\ diffLastPrice &= lastPrice - midPrice, \\ logVolume &= \begin{cases} log(volume) & \text{if } volume > 0 \\ 0 & \text{if } volume = 0 \end{cases} \end{aligned} \tag{13}$$

For calculations of return, we compute the rate of change of the average $midPrice$ at time t relative to time t-29.5s (An interval of 59 data points, with 0.5 seconds between each data point).

Additionally, we filled forward missing values and recorded the top 59 returns for each transaction segment (23:00 the

previous day, 9:00 am, 10:30 am, and 1:30 pm on the current day) as missing values for each trading day.

As shown in Table II, we also constructed several factors to predict the returns so as to test the quality of our data. The accumulated sum of factors of returns for 8 different factors are shown in Fig. 7. The significant and unstable differences in the effects of factors indicate the need for models to further analyze and estimate data in order to achieve better predictive performance.

B. Experiment Setup

The implementation details of our proposed method are presented in Table III. For our experiments, we utilized an Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz and two NVIDIA RTX 4090 GPUs as the hardware devices. The detail of each neural network is shown as Table IV.

C. Experiment Result & Analysis

The experimental results are shown in Fig. 6. We observed that the model is prone to overfitting when the training set consists of only one item. This may be due to the limited amount of data, which contains a significant amount of noise. Consequently, we utilized all six items to train the models and evaluated their performance across different items.

Regarding the backbone models, it is difficult to determine which one is the best. The LSTM [11] and Mamaba [16] models generally outperform the MLP in most scenarios, likely because they are better suited to capture temporal relationships due to their structural design. However, we noted that the training time for Mamaba is significantly greater than for the others, as it cannot compute in parallel. Although this comparison may be unfair due to its higher number of parameters, further reduction of its scale proves challenging.

For addressing the label imbalance problem, it appears that the sensitive loss (Eq. 10) and loss weighting (Eq. 9) methods yield better performance than others. In our task, both resampling and focal loss (Eq. 11) [20] methods sometimes performed worse than the benchmark, which employed no specific approach for the label imbalance problem. The reasons for this require further investigation.

VI. Discussion & Future Directions

In this project, we primarily illustrate the efficiency of using machine learning methods in high-frequency trading with label imbalance. In addition to the findings reported in this paper, we aim to highlight some problems and challenges encountered during our project to provide insights and experiences for future research.

A. Data Noise

In our experiment, we observed that financial data contains significant noise, which complicates the training process. Although we attempted to alleviate this through normalization (Eq. 1), the performance improvement was minimal. To address this challenge, we consider two approaches. One is to improve the model structure. For instance, [22] suggests using a Gaussian distribution for regression tasks instead of predicting a single value, which has proven effective in high-noise or dynamic data scenarios. The other approach involves employing feature engineering methods. For example, we identified some effective features, as shown in Fig. 7, which may help enhance model robustness and performance.

B. Domain Shift

During the project, we mistakenly used the mean and standard deviation of the entire dataset for normalization instead of using a rolling one-minute window. This oversight led to a noticeable improvement in model performance compared to the current results. This suggests that the testing data may exhibit a significant domain gap from the training data, indicating that the data domain gradually changes over time. To address this problem, we may need to explore cross-domain methods [23], [24].

C. Limitations

This work also has several limitations that can be explored in future research. First, regarding backbone model selection, the currently chosen models can be categorized into MLP and decoder structures (LSTM and Mamba). However, encoder structures are also important in time series analysis. We have developed a BERT model for financial data (following the approach in [25]) as provided in the repository. Due to time constraints, however, we have not fully tested its performance. Second, our current approach to addressing label imbalance primarily focuses on loss functions. Other methods, such as data augmentation and few-shot learning, could also be explored. Third, if data and resources permit, we could attempt to construct a large foundation model for financial data [26]. As mentioned in the last section, we found that model performance was significantly poor when we trained the model on a single item, but this issue was resolved when we used multiple items. We believe that scaling laws can also be beneficial in the field of finance.

VII. Conclusion

In this project, we attempt to learn the prediction for the forward 1 min return in the Chinese future market. We succeed in addressing the substantial challenges in high-frequency trading nature and building a model with the stable predictive power by proper backbone models and label imbalance adjustment methods.

References

[1] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[2] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, pp. 113–141, 2013.

[3] L. Wang, M. Han, X. Li, N. Zhang, and H. Cheng, "Review of classification methods on unbalanced data sets," *Ieee Access*, vol. 9, pp. 64606–64628, 2021.

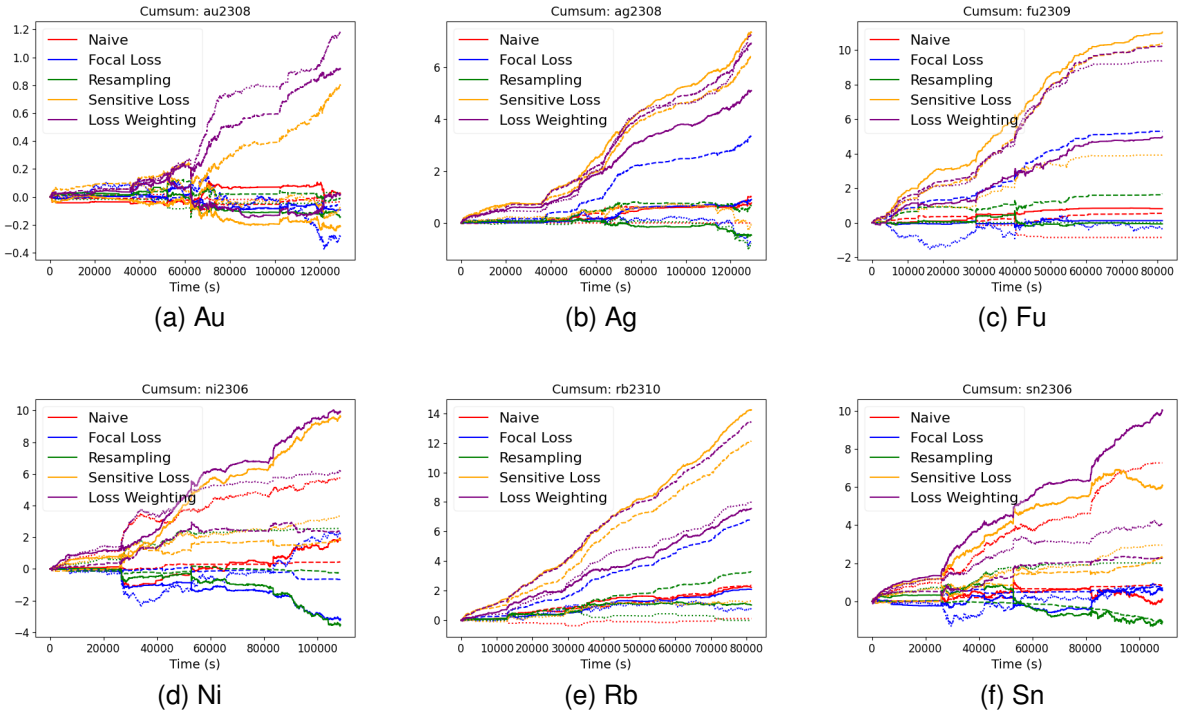

Fig. 6. Experimental Results for 6 Different Items: Different colors represent various methods for addressing label imbalance, while different line styles indicate different backbone models. The solid line represents Mamba [16], the dashed line represents LSTM [11], and the dotted line represents MLP.

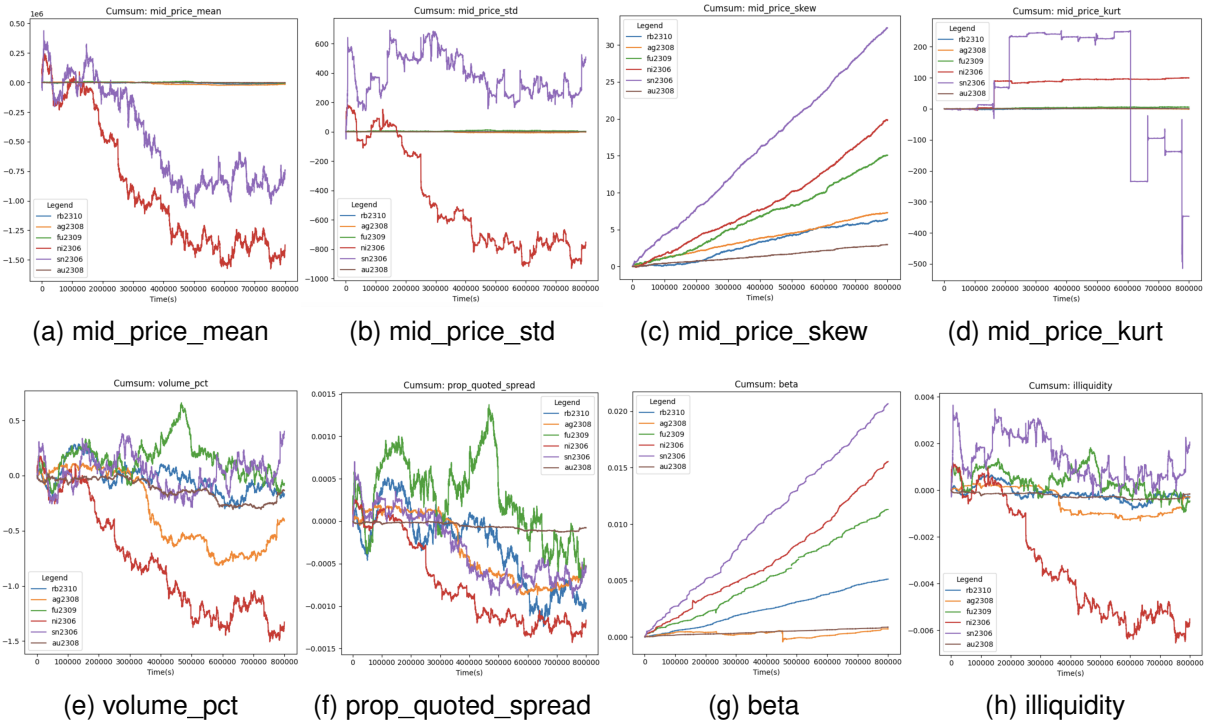

Fig. 7. Accumulated Sum of Factors Multiplied by Returns for 8 Different Factors: Different colors represent 6 different futures varieties.

TABLE II
FACTOR CONSTRUCTION

Factor Name	Factor Description
mid_price_mean	mean of the mid price in the past 30 seconds
mid_price_std	standard deviation of the mid price in the past 30 seconds
mid_price_skew	skewness of the mid price in the past 30 seconds
mid_price_kurt	kurtosis of the mid price in the past 30 seconds
volume_pct	the total transaction amount in the past 30 seconds divided by the total transaction amount in the past 5 minutes
prop_quoted_spread	(lowest ask price - highest bid price) / mid price
beta	the slope of the regression between return and transaction volume
illiquidity	absolute value of yield divided by transaction volume

TABLE III
EXPERIMENT CONFIGURATIONS

Configuration	Our Setting
Sample Dimension	60×13
Class Number	3
Batch Size	512
Optimizer	Adam
Learning Rate	0.0001
Early Stop Epochs	10

TABLE IV
MODEL CONFIGURATIONS

Model	Scale	Layer	Important Parameters	GPU Occupation
MLP	54K	4	structure=[780,64,64,3]	2.82G
LSTM [11]	26K	4	hidden_dim=64	3.66G
Mamba [16]	339K	4	state_size=4	4.73G

[4] L. Zhou, "Performance of corporate bankruptcy prediction models on imbalanced dataset: The effect of sampling methods," *Knowledge-Based Systems*, vol. 41, pp. 16–25, 2013.

[5] O. Loyola-González, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, and M. García-Borroto, "Study of the impact of resampling methods for contrast pattern based classifiers in imbalanced databases," *Neurocomputing*, vol. 175, pp. 935–947, 2016.

[6] A. Ghazikhani, R. Monsefi, and H. Sadoghi Yazdi, "Online cost-sensitive neural network classifiers for non-stationary and imbalanced data streams," *Neural computing and applications*, vol. 23, pp. 1283–1295, 2013.

[7] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2011.

[8] M. Sugiyama, M. Krauledat, and K.-R. Müller, "Covariate shift adaptation by importance weighted cross validation.," *Journal of Machine Learning Research*, vol. 8, no. 5, 2007.

[9] Z. Zhao, T. Chen, F. Meng, H. Li, X. Li, and G. Zhu, "Finding the missing data: A bert-inspired approach against package loss in wireless sensing," *arXiv preprint arXiv:2403.12400*, 2024.

[10] G. James, D. Witten, T. Hastie, R. Tibshirani, *et al.*, *An Introduction to Statistical Learning*, vol. 112. Springer, 2013.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001.

[13] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, pp. 11121–11128, 2023.

[16] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[17] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[18] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, "Combining recurrent, convolutional, and continuous-time models with linear state space layers," *Advances in neural information processing systems*, vol. 34, pp. 572–585, 2021.

[19] C. L. Castro and A. P. Braga, "Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data," *IEEE transactions on neural networks and learning systems*, vol. 24, no. 6, pp. 888–899, 2013.

[20] T. Lin, "Focal loss for dense object detection," *arXiv preprint arXiv:1708.02002*, 2017.

[21] Z. Zhao, "Adversarial-midibert: Symbolic music understanding model based on unbias pre-training and mask fine-tuning," *arXiv preprint arXiv:2407.08306*, 2024.

[22] N. Piovesan, D. López-Pérez, A. De Domenico, X. Geng, and H. Bao, "Power consumption modeling of 5g multi-carrier base stations: A machine learning approach," in *ICC 2023-IEEE International Conference on Communications*, pp. 3633–3638, IEEE, 2023.

[23] Z. Zhao, Z. Cai, T. Chen, X. Li, H. Li, and G. Zhu, "Knn-mmd: Cross domain wi-fi sensing based on local distribution alignment," *arXiv preprint arXiv:2412.04783*, 2024.

[24] Z. Zhao, T. Chen, Z. Cai, X. Li, H. Li, Q. Chen, and G. Zhu, "Crossfi: A cross domain wi-fi sensing framework based on siamese network," *arXiv preprint arXiv:2408.10919*, 2024.

[25] Z. Zhao, F. Meng, H. Li, X. Li, and G. Zhu, "Mining limited data sufficiently: A bert-inspired approach for csi time series application in wireless communication and sensing," *arXiv preprint arXiv:2412.06861*, 2024.

[26] H. Chen, H. Chen, Z. Zhao, K. Han, G. Zhu, Y. Zhao, Y. Du, W. Xu, and Q. Shi, "An overview of domain-specific foundation model: key technologies, applications and challenges," *arXiv preprint arXiv:2409.04267*, 2024.