Account Risk Detection in Large-Scale Financial Graphs with Auxiliary Asset Prediction

Jingye Zhao, Jianan Shen, Jianwei Wang, Tianyuan Zhou, Ruijia Wu, Kai Wang, Xuemin Lin

***Abstract*—Account risk detection, which aims to identify accounts at forced liquidation risk within financial account-asset bipartite graphs, is crucial for ensuring financial market stability and economic resilience. Although traditional node-classification-based anomaly detection techniques can be applied to this task, these approaches often exhibit two key limitations: (1) insufficient consideration of asset fluctuations, resulting in unsatisfactory accuracy; and (2) scalability challenges, making them unsuitable for large-scale financial graphs. To address these issues, we propose RiskGuard, a novel framework for account risk detection that integrates auxiliary asset prediction and gradient-based sampling. First, we introduce an auxiliary asset prediction paradigm to capture the critical influence of asset fluctuations on account risk. Rather than solely predicting account risk, our unified model employs a Temporal-Attention Net (TANet) to jointly predict asset fluctuations and account risk. This auxiliary task enables the model to learn fluctuation-aware asset representations, significantly enhancing prediction accuracy. To overcome scalability challenges, we design GLUE, an online graph sampler leveraging gradient entropy. GLUE dynamically adjusts sampling weights based on model gradients and graph structure, prioritizing high-entropy nodes in the neighborhood for improved efficiency. Extensive experiments on four financial datasets demonstrate that RiskGuard outperforms existing techniques in accuracy while achieving high efficiency in processing large financial graphs.**

***Index Terms*—Bipartite Graph, Graph Neural Networks, Financial Data Management**

I. Introduction

Financial networks are widely used to model complex interactions in financial systems, supporting applications like market analysis, credit risk assessment, and systemic risk modeling [19], [21], [24], [31], [37], [45]. A common and practical scenario of financial networks involves account-asset networks, where relationships between accounts and financial assets can reflect critical insights into market behaviors and portfolio dynamics. Leveraged trading, prevalent in such networks, allows investors to amplify returns using borrowed funds but increases account risks. While traditional default risks mainly depend on account behaviors, leveraged trading extends this by introducing forced liquidation risk, a distinct threat driven by both account transactions and market fluctuations. Such accounts are at risk when market movements push their leveraged positions close to margin call levels, requiring real-time monitoring of both account positions and market conditions. Excessive leverage can trigger margin calls, leading to forced liquidations and destabilizing the system [31], [39]. Thus, robust account risk detection techniques are crucial for mitigating losses from margin calls and enhancing financial system stability [39].

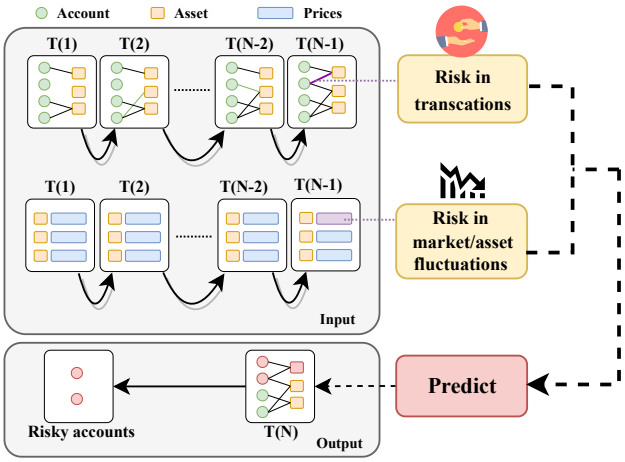


Fig. 1: An illustration of account risk detection, showing risks arising from transactions and market fluctuations.

Numerous related studies concerning account risk or account patterns have emerged in the financial domain [46]–[48]. Early-stage methods [46], [47] typically focus on machine learning techniques to analyze account or asset patterns. For instance, the method in [46] employs multivariate regression analysis to demonstrate that borrower activity information can assist banks in making more informed lending decisions, offering valuable insights for identifying potentially risky accounts. However, simple regression struggles to capture complex temporal and structural patterns within the data, often resulting in unsatisfactory performance.

As illustrated in Figure 1 and highlighted in previous works [49]–[51], account-asset financial networks can be naturally modeled as the temporal bipartite graphs (TBGs), where one set of nodes represents trading accounts and the other set of nodes represents financial assets, with edges capturing their dynamic interactions over time. Then, node-classification-based approaches, such as fraud detection or anomaly detection, can be applied to TBGs for identifying risky nodes [22], [52], [53]. While these approaches have shown promise in various risk detection scenarios, their

Jingye Zhao, Jianan Shen, Tianyuan Zhou, Ruijia Wu, Kai Wang, and Xuemin Lin are with the Antai College of Economics and Management, Shanghai Jiao Tong University, China. E-mail: jingye_zhao@sjtu.edu.cn, s1lent@sjtu.edu.cn, ztyqwq89@sjtu.edu.cn, rjwu@sjtu.edu.cn, w.kai@sjtu.edu.cn, xuemin.lin@sjtu.edu.cn.

Jianwei Wang is with The University of New South Wales, Australia. E-mail: jianwei.wang1@unsw.edu.au.

Manuscript received: February, 2025.

application to forced liquidation risk remains underexplored. The most relevant existing research lies in graph-based fraud and anomaly detection. Many existing methods [19]–[21] directly apply graph neural networks (GNNs) to classify which accounts are at risk. A notable example is HOGRL [19], which utilizes (temporal bipartite) graph neural networks to capture the structural features in the data for accurate pattern detection.

However, since these models are often trained on the entire graph structure, their computational complexity grows exponentially with the graph size increases, making it challenging to scale to large financial graphs. To address this, a mainstream alternative integrates GNNs with sampling strategies to extract a relevant subgraph and aggregate information from it, which enhances scalability [27], [50].

Motivations. While these GNN-based methods achieve outstanding performance by effectively capturing account-asset relationships, they face two major limitations.

Firstly, existing methods focus mainly on account behaviors while overlooking market conditions. In financial graphs, information flows bidirectionally: account activity influences asset behavior, and asset fluctuations, in turn, affect accounts [18]. For instance, a sudden drop in asset prices can trigger margin calls. Although bipartite graph neural networks [52], [54] can model account-asset interactions, they typically rely on static representations, failing to fully capture the dynamic nature of asset price changes.

Secondly, existing methods struggle with large financial datasets effectively. Although sampling-based methods [27], [50] can handle large-scale graphs by sampling context subgraphs and propagating information within them to learn representations, they typically rely on fixed sampling rules. However, the characteristics of temporal account-asset bipartite graphs are evolving over time. These sampling-based approaches may not adequately capture the high dynamism of financial graphs, potentially degrading performance.

Challenges. To design an accurate and efficient account risk detection method in financial graphs, two challenges exist.

Challenge I: How to effectively incorporate asset fluctuation information into account risk detection. The uncertainty of future asset performance and the complex inter-asset dependencies complicate the utilization of fluctuation information. A promising method is to employ financial temporal models, like iTransformer [1], to forecast future asset fluctuations and integrate them for account risk detection. However, besides the inherent difficulty of accurately predicting asset fluctuations, the forecasting models may introduce biases that deviate from the objective of the account risk detection.

Challenge II: How to handle large-scale financial graph while maintaining effectiveness. The dynamic nature of financial data increases the complexity of scaling the model. The existing sampling-based methods for risk detection primarily focus on the fixed sampling rules which inevitably face temporal shifts in the characteristics of accounts and nodes. Although we can use some temporal sampling methods like Zara [40] to enhance its performance, they often rely on predefined assumptions about the patterns of temporal shifts, which may not align with the objective of the model.

Our approaches. Motivated by the above challenges, we propose RiskGuard, a new account risk detection framework with auxiliary asset prediction and gradient-based sampling.

To address Challenge I, we introduce an auxiliary asset prediction paradigm besides the objective of account risk detection. Instead of directly predicting the fluctuation of the asset in the future, we first use the auxiliary asset prediction to learn a fluctuation-aware representation of the asset, and then use the representation to augment the financial graph for better account risk detection. Specifically, at a coarse granularity, we employ a unified model that simultaneously addresses auxiliary asset prediction and account risk detection. Initially, we focus on the objective of asset fluctuation to extract dynamic features, enabling the model to learn fluctuation-aware representations of assets. These representations capture the temporal patterns and fluctuations inherent in the assets. Subsequently, we leverage an updated temporal bipartite graph to conduct account risk detection, allowing the model to identify potential risks based on the latest interrelations between accounts and assets. At a fine granularity, we introduce Temporal-Attention Net (TANet), a module designed to explore the temporal and structural relationships in the graph. It captures the fine-grained interactions by modeling how accounts and assets evolve over time and how they are structurally connected to each other.

To address Challenge II, we propose GLUE, an online Graph sampLer Utilizing gradient Entropy. Inspired by online mirror descent [41], [42], GLUE incorporates two key components: weighted neighbor sampling and the adaptive update of sampling weights. The process begins by computing the sampling probabilities based on the gradient entropy, which captures the uncertainty in the predictions of the model. These probabilities are then used to guide the weighted neighbor sampling, ensuring that samples with higher uncertainty and nodes in the neighborhood are prioritized. After each epoch, the sampling weights are adaptively updated to reflect the latest model gradients and changes in the underlying data distribution of the financial graph. By applying weighted neighbor sampling and adaptive updates at every epoch, GLUE dynamically adjusts its parameters in response to both model updates and evolving data. Furthermore, we prove that GLUE is unbiased, guaranteeing that the learned representations converge to those obtained from the full graph. This property ensures that critical patterns and relationships within the financial graph are preserved, ultimately leading to more accurate account risk detection.

Contributions. Our contributions are summarized as follows.

1) We propose a new framework, RiskGuard, which incorporates auxiliary asset prediction and gradient-based sampling techniques for account risk detection.
2) We propose auxiliary asset prediction to enhance the capture of asset fluctuations and integrate market information more effectively into RiskGuard.
3) We propose GLUE to enable RiskGuard for handling large financial graphs. By leveraging gradient entropy, GLUE effectively captures temporal information and maintains unbiased representations.

4) Extensive experiments on 6 datasets demonstrate the superior performance of RiskGuard in both accuracy and efficiency. Regarding accuracy, RiskGuard achieves an average performance improvement of 15.00% in the AUC score over existing approaches. Moreover, RiskGuard can well handle large-scale graphs with an average speedup of $1.61\times$ compared to existing graph-based neural models.

II. Related Work

A. Account Risk Detection

Market-based methods. Market-based methods assess fluctuations in asset values to directly evaluate their influence on account risks [18], [46]. Factor models [32], [55] explain asset fluctuations by various factors, including metrics such as high prices and opening prices. For instance, a capital asset pricing model (CAPM) proposed in [55] attributes variations in stock revenue to different exposures to the market factor. Recently, machine learning models have become increasingly popular for extracting market information [1]–[3], [23], [24], [28]. The method in [23] simulates the traditional factor model by compressing information into the variational embedding space and leveraging the variational auto-encoders to reconstruct factor returns. The method in [1] incorporates an inverted Transformer that applies self-attention across different assets for forecasting. However, these models primarily focus on assets while overlooking account behaviors, leading to limited performance for account risk detection.

Tabular-based methods. Tabular-based methods can be used to analyze account patterns in the financial domain. One direction involves neural models that learn these relationships. For instance, MLP [47], [56] employs multiple fully connected layers to model complex non-linear relationships within the data, enabling the classification of accounts at risk.

Tree-based models are also commonly employed to model account features and assess risk. LightGBM [25] employs gradient-boosting with tree-based algorithms to model data relationships and assess account risks.

Graph-based methods.

Graph-based methods are promising in capturing both account-asset structures and attribute information. Several approaches are based on identifying specific patterns [20], [52], [54], such as the (α, β)-core. However, these methods fail to utilize attribute information [22]. Several fraud-detection methods are based on GNNs [4], [19], [21], [22], [26], [27], [36]–[38], which can handle both structure and attribute information. Yu et al. [22] propose a group-based fraud detection network(GFDN), which comprises a structural feature generation module and a community-aware fraud detection network to enhance the identification of fraudulent activities within user groups. A Heterogeneous Temporal Graph Transformer (HTGT) framework is proposed in [27] for detecting evolving Android malware.

However, it primarily focuses on single-type behavior propagation, while neglecting the intricate relationship between market dynamics and customer behaviors, which are crucial for accurately detecting account risks [18].

General dynamic graph prediction methods. Recent studies have proposed various methods for modeling dynamic graphs [5]–[11], which encode structural and temporal information separately or alternately. For instance, SLATE [11] uses supra-Laplacian encoding in spatio-temporal transformers to preserve both spatial and temporal information. However, these methods do not distinguish between different node types in bipartite graphs, updating all nodes simultaneously, which leads to redundant computation. Moreover, they may fail to capture market volatility effectively.

B. Graph Sampling Techniques for GNN and Related Domains

Graph sampling techniques are typically used to accelerate the training of GNN [45]. The early-stage methods usually utilize the fixed sampling policy [44], [45], [57]–[59]. GraphSAGE [44], a sample and aggregate framework, introduces the first sample strategy for accelerating GNN training, which involves sampling 2-hop neighbors for each target node. The method proposed in [57] introduces a sampling approach that assigns and maintains probabilities for sampling each edge. However, these methods do not incorporate feedback from the model, which makes it challenging to align the sampling process with the GNN.

Recently, several sampling methods have been proposed to learn how to sample nodes based on feedback from the GNN [60]–[63]. Some approaches [60], [61] to the problem are by reinforcement-style ways. The method in [62] integrates the sampler with the GNN. However, they inevitably involve a training phase, which increases the computation burden. Method in [63] designs a sampling method using historical representations to regulate the variance from neighbor sampling. However, existing sampling methods often overlook the temporal information when handling temporal graphs.

Beyond graph-specific sampling, adaptive sampling techniques have also been successfully applied in other domains such as recommendation systems [12]–[14], [16]. These methods demonstrate the importance of feedback-driven sampling strategies in improving model performance. The method in [16] samples different types of samples with adaptive sampling weights by updating group sampling distributions, adaptively adjusting the momentum parameters. However, these approaches mainly focus on negative sampling problems and do not optimize for graph sampling techniques.

III. Preliminary

A. Graph Modeling

Detecting risks in financial accounts often requires assessing the asset holdings of an account at a specific time t. To represent account-asset relationships effectively, the Temporal Bipartite Graph (TBG) is introduced and defined as follows:

Definition 1. Temporal Bipartite Graph (TBG). We define a temporal bipartite graph in the time range $[1, T]$ as $\mathcal{G} = \left(\{G^t\}_{t=1}^{T}\right)$, where each G^t represents a bipartite graph at the timestamp $t \in [1, T]$. Formally, we define $G^t = (\mathcal{U}, \mathcal{V}, \mathcal{X}_{\mathcal{U}}^t, \mathcal{X}_{\mathcal{V}}^t, \mathcal{E}^t)$ where: $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ are two disjoint sets of nodes, with m and n representing the sizes of $\mathcal{U}$ and $\mathcal{V}$, respectively; and $\mathcal{E}^t \subset \mathcal{U} \times \mathcal{V}$

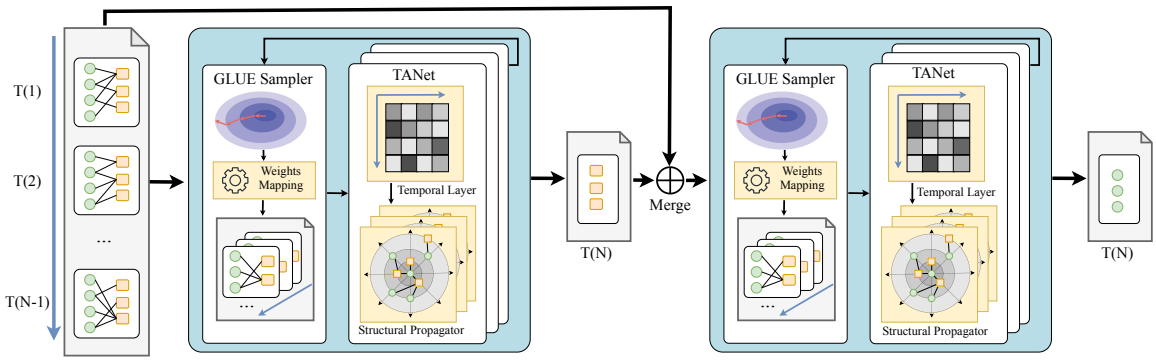


Fig. 2: Illustration of the RiskGuard framework

is the edge set at time t. $\mathcal{X}_{\mathcal{U}}^t$ and $\mathcal{X}_{\mathcal{V}}^t$ are the features for the nodes in sets $\mathcal{U}$ and $\mathcal{V}$, respectively.

Specifically, we use the set $\mathcal{U}$ to represent accounts in the financial network and the set $\mathcal{V}$ to represent assets, where assets refer to blockchain tokens and traditional financial instruments such as stocks and bonds. The edge set $\mathcal{E}^t \subset \mathcal{U} \times \mathcal{V}$ represents the account nodes holding asset nodes at time t, with edge weights indicating holding amounts.

B. Problem Formulation

In TBG, we model all account nodes as leveraged investors who borrow funds from lenders (e.g., financial institutions) to transact with asset nodes. The Maintenance Margin Ratio (MMR) is an indicator of liquidity risk [17], [18] and is widely used to evaluate whether an account can maintain its leverage. It is defined as:

$$\mu = \frac{\rho}{\psi}, \tag{1}$$

where μ is the MMR, ρ denotes the equity value owned by the account, and ψ is the current market value of the loans held in the account. In TBG, ρ is calculated as the sum of the market value of all long positions and the currency of the account. ψ is computed as the total value of associated loans, including the short positions of neighboring asset nodes and borrowed funds. Intuitively, MMR indicates how much the current value of an account can cover its outstanding loans, thereby determining the risk level of this account. Formally, an account is classified as *at risk* on the day t if the predicted MMR $\mu_{t+\delta}$ for the day $t+\delta$ satisfies: $\mu_{t+\delta} < \mu^*$ where μ^* is a threshold and is set to 0 in this study.

Problem statement. Given a temporal bipartite graph $\mathcal{G}$ representing account-asset financial transactions, our goal is to detect the set of at-risk accounts $\mathcal{R} \subseteq \mathcal{U}$ on day t. The TBG structure naturally captures the dynamic account-asset holdings required for computing equity values (ρ) and loan obligations (ψ) in the MMR framework, making it well-suited for modeling leveraged account risks. Formally, an account $u \in \mathcal{U}$ belongs to $\mathcal{R}$ if its predicted MMR $\mu_{t+\delta} < \mu^*$, indicating potential forced liquidation risk.

IV. Our Approach

A. Overview

We first present an overview of RiskGuard, which is illustrated in Figure 2. It comprises two core components: the GLUE sampler for sampling context subgraphs to handle large graphs, and the TANet for information propagation across the graph. The framework integrates an auxiliary asset prediction objective to capture asset fluctuation information. To achieve this, we adopt a unified model consisting of the GLUE sampler and the TANet, enabling it to simultaneously learn the auxiliary task of asset fluctuation and the primary task of account risk detection. Initially, the model is trained on the auxiliary objective of asset fluctuation, encoding fluctuation information in the embedding. Then, we update the embeddings for the asset features. The updated graph is utilized to train for account risk detection.

B. GLUE Sampler

Motivations. Numerous algorithmic acceleration techniques have been developed to scale GNN models for large-scale graphs, including graph-level methods [44], [45], [57]–[59], [64] and model-level methods [65]–[68]. Among these, sampling-based methods demonstrate high efficiency and competitive effectiveness [44], [45], [57]–[59]. However, financial graphs exhibit two properties that limit existing sampling methods: **1): Structure imbalance**. In account-asset bipartite graphs, assets are typically limited (e.g., thousands of assets in the market), while accounts are numerous(e.g., millions of accounts in the market), each holding a small subset of assets. As a result, the degree of assets is significantly higher than that of accounts. Existing methods [26], [36], [44] with fixed sampling rules tend to over-sample high-degree nodes that may represent irrelevant connections while overlooking critical low-degree nodes. **2): Temporal shift**. Account-asset graphs evolve over time, causing non-stationary neighbor distributions. The context subgraph sampled at a given stage may not remain representative or informative over time. Most existing samplers rely on static strategies that fail to account for the dynamic nature of edge connections over time, leading to unsatisfactory performance.

To overcome the challenges discussed above, we introduce GLUE, an online Graph sampLer Utilizing gradient Entropy.

The GLUE comprises two key phases: 1) the edge weight update phase which updates the weight of each edge in the neighborhood based on the gradient entropy and the mirror descent algorithm [41], and 2) the subgraph sampling phase which samples the context subgraph based on both the neighborhood structure and the edge weights from the first phase. Firstly, GLUE receives gradient information of

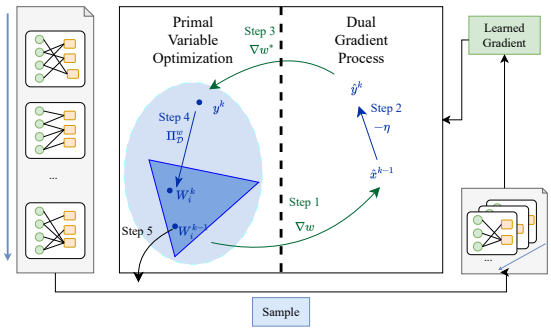


Fig. 3: Illustration of the GLUE sampler

the edge sampled in the previous step. This gradient matrix, denoted as J, is the gradient of sampled first-layer neighbors from the last layer of the model. J is then utilized to update the weight matrix W, which contains the probability of neighbors to sample for each center node. This process aims to maintain the probability of sampling each neighbor. Neighbors with higher probabilities are expected to contribute more to the learning process of RiskGuard by providing more information to the model, resulting in larger gradient updates. These probabilities are expected to evolve as the model is updated, making it natural to model it as an online optimization problem. The mirror descent algorithm [41], [42] generalizes classic gradient descent by using a distance-generating function to map updates into a dual space, making it well-suited for such online optimization settings. Notably, mirror descent inherently handles non-stationary data through its dynamic regret minimization framework—by incorporating a distance-generating function (DGF), it provides adaptive learning rates that allow the model to effectively adapt to abrupt changes in gradient distributions during the learning process. Specifically, we use the entropy setup as follows [41]:

Definition 2 (Entropy Setup)**.** we first define:

- $\|\cdot\| = \|\cdot\|_1$ (The norm under the setting is L1-norm)
- Domain $\mathcal{D}$ is a full-dimensional standard simplex Δ_n^+
- DGF: $\omega(x)$ is the regularized entropy, defined as:

$$\omega(x) = (1+\delta)\sum_{i=1}^{n}\left(x_i + \frac{\delta}{n}\right)\ln\left(x_i + \frac{\delta}{n}\right) : \Delta_n^+ \to \mathbb{R}.$$

where n is the dimension of the vector x in $\mathcal{D}$, Δ_n^+ is the n-dimensional probability simplex defined as:

$$\Delta_n^+ = \{x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = 1, x_i \geq 0 \text{ for all } i\},$$

and $\delta > 0$ is a regularization parameter and is set to 1 here.

Setup 2 has been widely applied [41], [42] and fits our problem through the standard simplex definition. We denote W_i as **Gradient Entropy** for node i, which serves as input to $w(\cdot)$ under the entropy setup, yielding:

Theorem 1. Under the entropy setup and the mirror descent algorithm, we have the update process of W_i:

$$W_i \leftarrow (1+\delta)\cdot\text{softmax}\left(\left(W_i + \frac{\delta}{n}\right)\odot\left(\exp(-\frac{\eta}{1+\delta})\right)\right) - \frac{\delta}{n},$$

where $\odot$ denotes the Hadamard product, η is gradient information, and the softmax function is defined as in [69].

Algorithm 1: The GLUE Sampler

Input: TBG in a time window $\mathcal{G}_1^t = \{G^1, G^2 \ldots, G^t\}$, sampling depth L, center nodes set C, gradient matrix J from last step, weight matrix W at time t
Output: Sampled subgraph $\mathscr{S}$

```
// The weight update phase.
for each sampled node i in J do
    η ← γ J_i
    W_i ← (W_i + δ/n) ⊙ exp(−η/(1+δ))
    W_i ← (1+δ) · softmax(W_i) − δ/n
// The sampling phase.
S ← Φ; S_0 ← C
for ℓ = 1 to L do
    S_ℓ ← ∅
    for node i in S_{ℓ−1} do
        if ℓ = 1 then
            ▷ Gradient entropy (W_i)-based sampling
            Select the top a fraction of edges (0 < a < 1)
            Randomly select an additional b fraction of
              edges (0 < a + b < 1) from rest of neighbors
            Assign weights: 1 for the top edges and (1−a)/b for
              the remaining edges
        else
            Uniform random sampling for other layers
        Add sampled neighbors to S_ℓ
    S ← S ∪ S_ℓ
𝒮 ← Φ
for j = 1 to t − 1 do
    Induce subgraph S_j by S and G^j
    𝒮 ← 𝒮 ∪ S_j
𝒮 ← 𝒮 ∪ S
return the sampled subgraph 𝒮
```

The proof of Theorem 1 is in the online full version [33].

Secondly, based on the edge weight matrix W at time t, we sample across L hops of the center node. Intuitively, the first-order neighbors of a central node are of greater importance. Therefore, we focus on sampling first-order neighbors. Specifically, we aim to ensure that the distribution of sampled neighbors remains invariant with the full neighborhood. Additionally, to provide more information to our model, we choose edges with high W values. Inspired by [25], we design a gradient entropy-based sampling process to sample first-order neighbors. This approach retains all neighbors with high W values and randomly samples from those with low W values. To compensate for sampling bias during the message propagation, we enhance the propagation weights of nodes exhibiting low W values. Specifically, we select the top a portion of neighbors based on the W weights. Subsequently, we randomly sample an additional b fraction of nodes from the remaining neighbors. During message passing, we amplify the neighbors with low W values by a weight of $\frac{1-a}{b}$. GLUE has several good properties, as Theorems 2 and 3 shows.

Theorem 2. The sampling result of the first layer in Algorithm 1 is an unbiased estimator for the Message Passing (MP) layer in GNNs: $\mathbb{E}\left[\nabla\hat{f}(x)\right] = \nabla f(x)$, where $f(\cdot)$ is the output function of the MP layer with all neighbors as input, $\hat{f}(\cdot)$ is the output function of the MP layer when the input is a gradient entropy-based sampling method.

Theorem 3. Using gradient entropy-based sampling, the error $\|\nabla f(x) - \nabla \hat{f}(x)\|_1$ is upper bounded by $\sqrt{\frac{4(1-a)c^2 \log K}{n}} \cdot \sum_k g_k$ with probability at least $1 - \frac{2}{K}$. Here k is k-th element of the gradient ∇; $c = \max\{1, |\frac{a+b-1}{b}|\}$; g_k is maximum element of $|\nabla_k f(x)|$; and K is the length of $\nabla f(\cdot)$.

The proofs of Theorems 2 and 3 are in subsection A and B of appendix. Theorems 2 and 3 show that our gradient entropy-based sampling is unbiased with bounded approximation error. These properties are important for convergence guarantees.

Our approach addresses the structural imbalance and temporal shifts in financial graphs. For structural imbalance, gradient entropy serves as an adaptive importance measure: neighbors with larger gradients (indicating greater contribution to learning) are sampled with higher probability. This naturally handles structural imbalance by prioritizing informative connections over high-degree but irrelevant nodes. For temporal shifts, mirror descent enables online adaptation to temporal shifts through continuous probability updates based on recent feedback. Moreover, the sampling process ensures both exploitation of stable patterns and exploration of potentially changing relationships. This design is particularly important for capturing evolving market dynamics.

While uniform sampling [30] is also unbiased at each time step, it suffers from high variance in the presence of structural imbalance and temporal shifts, only a small fraction of uniformly sampled neighbors may be truly important. Our method maintains both unbiasedness and controls error bounds across the entire temporal learning process, leading to more stable and faster convergence.

The illustration of the GLUE is in Figure 3, and the overall algorithm is summarized in Algorithm 1. GLUE takes TBG $\mathcal{G}_1^t = \{G^1, G^2 \ldots, G^t\}$, sampling depth L, center nodes set C, gradient matrix from last step $J \in \mathbb{R}^{k \times n}$, weight matrix W at time t as input and output sampled subgraph set $\mathscr{S} = \{S_1, \ldots, S_t\}$. Without loss of generality, we define the time window as [1, t], with nodes to update in the first k rows and each center node having n neighbors. First, GLUE updates the weight matrix based on the gradient entropy (Line 1-4), then samples a subgraph set $\mathscr{S}$ based on W and $\mathcal{G}_1^t$. The weight update phase (Lines 1-4) involves three steps per center node i. Line 2 calculates η as the product of the step size γ and the gradient vector J_i. Then, the weight matrix W_i is updated using the Hadamard product of the historical weight term $\left(W_i + \frac{\delta}{n}\right)$ and the current information term $\exp\left(-\frac{\eta}{1+\delta}\right)$ (Line 3). The updated W_i is then projected onto the probability simplex (Line 4). In the sampling phase, we sample L-hop neighbors(Lines 5-22) by maintaining set $\mathscr{S}$. For the first hop, we apply gradient entropy-based sampling(Lines 10–13); otherwise, uniform random sampling(Line 15). For gradient entropy-based sampling, we use parameters a and b as the ratio of high-weight edges and probability of low-weight edges. GLUE selects all edges with the top a highest weight (Line 11) and then randomly selects an additional b fraction of edges from remaining neighbors (Line 12). The sampled edges are then assigned weights (Line 13). Through this process, GLUE generates a sampled subgraph S for day t. Then, subgraph S_j is induced for each day $j \in [1, t-1]$ (Line 19-22). In this way, we obtain the subgraph set $\mathscr{S}$.

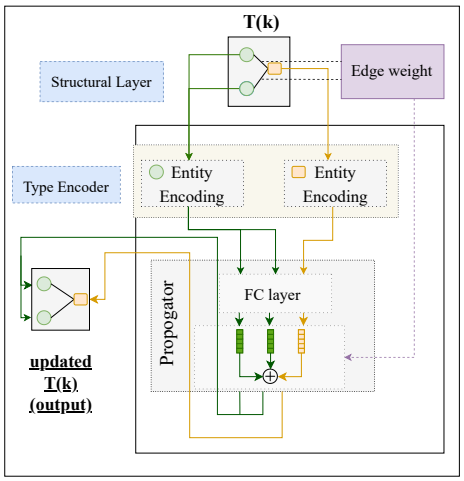


Fig. 4: The structural propagator

C. The Architecture of TANet

Motivations. After the context graph is sampled using the GLUE sampler, the features within the graph need to be aggregated to learn the node representations. For a financial graph, temporal information plays a crucial role in predicting account risk since the patterns in the graph, such as transaction history and market trends. To better capture the temporal dynamics, a common approach is to use Recurrent Neural Networks (RNNs) as the backbone [70]. However, RNNs face challenges with long-range dependencies due to vanishing gradients and computational inefficiencies, which are not suitable for account risk detection [27].

In this paper, we introduce TANet, a Temporal-Attention Net, inspired by transformer-based methods [27]. TANet leverages the attention mechanism to encode time-varying features and capture temporal correlations effectively. Furthermore, to better aggregate features and process accounts and assets within daily bipartite graphs, TANet incorporates a structural propagator, as illustrated in Figure 4.

Specifically, we first implement a one-layer fully connected encoder to map features $\mathcal{X}_{\mathcal{U}}^t$ and $\mathcal{X}_{\mathcal{V}}^t$ to a hidden dimension:

$$z_u^t = \sigma(\Theta_{\Phi(u)} \cdot x_u^t),$$

where $\Theta_{\Phi(u)} \in \mathbb{R}^{d \times d_u}$ is a learnable weight matrix, $\Phi(u) : \mathbb{R} \to \mathbb{R}$ is a fixed type mapping, $\mathbf{x}_u^t \in \mathbb{R}^{d_u}$ are the feature of node u of dimension d_u at time t, and σ is the non-linear activate function (e.g., LeakyReLU [45] as in experiments).

The structural propagator contains a type encoder to encode the type information of both nodes and a message-passing layer to capture neighborhood information. To handle different types of nodes in the account-asset bipartite graph, we first utilize a type encoder g(·) that maps their features into a unified representational space: $g(u) = \sigma(P_{\Phi(u)} \cdot z_u^t)$, where $P_{\Phi(u)} \in \mathbb{R}^{d \times d}$ is a learnable, node-dependent weight matrix. Given the node u and the sampled neighbors $\mathcal{N}_u^t$, the latent representation of u at time t can be computed as:

$$\mathbf{h}_u^t = \mathrm{MP}\left(\mathbf{z}_u^t, \left\{\left(\mathbf{z}_v^t, e_{u,v}^t\right) : v \in \mathcal{N}_u^t\right\}; \Theta_{\mathrm{MP}}\right),$$

where $\mathbf{z}_v^t \in \mathbb{R}^d$ are the input feature of node v with a dimension of d output by the type encoder at time t, and $e_{u,v}^t$ and the sampled edge between u and node v at time t. $\mathcal{N}_u^t$ is the sampled subgraph of node u at time t. Θ_{MP} is the learnable parameters of the message-passing layer and is shareable across all timestamps.

Then, we implement a Propagator layer to facilitate message passing between nodes. Formally, we define it as: $h_u^t = \sigma\left(\sum_{v\in\mathcal{N}_u\cup\{u\}} \gamma_v \frac{1}{\sqrt{d^u d^v}}\Theta g_v\right)$, where σ is an activate function, γ_v is the learning weight of sampled node v (assigned by Algorithm 1 (Line 13)), d^u, d^v is the degree of node u and node v, Θ is the weight matrix.

Moreover, we utilize the attention mechanism to capture the temporal relationship. An attention function maps a query and a set of key-value pairs to an output, where the query, keys, values, and output are represented as vectors [27], [71]. To capture the importance of a historical embedding $\mathbf{h}_u^{t'}$ relative to the current feature vector $\mathbf{h}_u^t$, we transform $\mathbf{h}_u^t$ into a query vector, $\mathbf{h}_u^{t'}$ into a key vector, computing their dot product as the attention. Formally, given the historical representation $\mathbf{h}_u^{t'}$ from the structural propagator, we denote $H_u^t = \|_{t'\le t} h_u^{t'} \in \mathbb{R}^{t\times d_{\Phi(u)}}$ as the concatenated sequences of the historical representations up to timestamp t. The temporal query $\mathbf{q}_u^t$, keys $\mathbf{k}_u^{t'}$ and values $\mathbf{v}_u^{t'}$ are formulated as:

$$\mathbf{Q}_u^t = \mathbf{W}_q^{TL}\cdot H_u^t; \mathbf{K}_u^{t'} = \mathbf{W}_k^{TL}\cdot H_u^t; \mathbf{V}_u^{t'} = \mathbf{W}_v^{TL}\cdot H_u^t \quad (2)$$

where $\mathbf{W}_q^{TL}, \mathbf{W}_k^{TL}, \mathbf{W}_v^{TL} \in \mathbb{R}^{d\times d}$ denote the transformation matrices for query, key, and value, respectively.

After all, the representation of node u at timestamp t where $t \ge t'$ can be computed as:

$$\mathbf{Z}_u^t = \text{Attention}(\mathbf{Q}_u^t, \mathbf{K}_u^{t'}, \mathbf{V}_u^{t'}) = \text{softmax}\left(\frac{\mathbf{Q}_u^t \mathbf{K}_u^{t'}}{\sqrt{d}}\right)\mathbf{V}_u^{t'}.$$

When computing $\mathbf{Z}_u^t$, we first utilize the attention layer to capture its temporal relationship w.r.t. the representation from previous layers and then use the structural layer to aggregate information at the current timestamp.

D. Training Objectives

Motivations. The training objectives guide the model toward achieving the desired patterns and outputs. In the financial context, the fluctuation of asset prices, i.e., the fluctuations of these assets, contributes to the risk of the account. Mathematically, the fluctuations ξ is defined as: $\xi = \frac{P_h - P_l}{P_o}$ where P_h, P_l, and P_o are the highest, lowest, and opening prices during the period, respectively. As shown in Equation 1, the MMR of an account is influenced by the value of the asset it holds. Significant asset fluctuations affect ρ or ψ, which, in turn, impacts the MMR of the account.

Therefore, we employ a unified model to first learn the auxiliary task of asset fluctuation prediction, followed by the primary task of account risk detection. Specifically, for the auxiliary task, we predict whether assets will experience significant future fluctuations as a classification problem, as demonstrated in previous works [32], [48]. Given the highly imbalanced node distribution in the account-asset bipartite graph (Section IV-B), we adopt the Logit Adjustment loss [72] to handle class imbalance. The training objective of the auxiliary task is defined as follows:

$$\mathcal{L}_{aux} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{f(v_i,\mathscr{S},\theta)+\tau\cdot\log\pi_{y_i}}}{\sum_{y'\in\mathcal{Y}} e^{f_{y'}(v_i,\mathscr{S},\theta)+\tau\cdot\log\pi_{y'}}}. \quad (3)$$

Here N is the number of asset samples; $\mathscr{S}$ is the sampled subgraph; θ is the parameter set; $\tau > 0$ is a hyperparameter controlling the strength of the adjustment which is set to 1 following previous work [72]; π_y represents the prior probability of class y, and $f_y(v_i, S, \theta)$ denotes the logit score for class y given input v_i.

After training on the asset fluctuation objective, the features encoded in the model capture key temporal patterns and fluctuations. Thus, we first update the asset representation according to the model learned in the auxiliary task. Specifically, we replace the stock features in the original graph for each day t with the embeddings of the stocks obtained from the last temporal layer of the same day t. Using this reconstructed temporal bipartite graph, we train the model for account risk detection. The account risk task classifies whether a node is likely to be at risk in the future. To address class imbalance, we apply Logit Adjustment loss as in $\mathcal{L}_{aux}$:

$$\mathcal{L}_{main} = -\frac{1}{M}\sum_{i=1}^{M}\log\frac{e^{f(u_i,\mathscr{S},\theta)+\tau\cdot\log\pi_{y_i}}}{\sum_{y'\in\mathcal{Y}} e^{f_{y'}(u_i,\mathscr{S},\theta)+\tau\cdot\log\pi_{y'}}}, \quad (4)$$

where M is the number of customer samples.

TABLE I: Statistics of datasets. ERC-U: ERC-users, ERC-A: ERC-active, ERC-L: ERC-low, SimA-C: SimA-Chat, COV-19: COVID-19 Crisis, Bear: Sustained Bear Market

Metrics	ERC-U	ERC-A	ERC-L	SimA-C	COVID-19	Bear
Acct. nodes/day	1,523,333	816,993	845,031	988	29,090	29,090
Assets/day	6	6	6	297	806	806
Days	29	70	49	201	591	591
Total edges	15.0M	30.2M	34.0M	0.32M	21.1M	22.6M
Avg. label pos (%)	5.4	6.1	0.039	0.056	0.077	0.056
Acc. features	4	4	4	14	10	10
Item features	19	19	19	13	12	12

V. Experimental Evaluations

A. Experimental Setup

Datasets.We use two categories of financial networks in our experiments: blockchain networks and financial asset holdings. Table I provides an overview of the data. Details on data processing can be found in online full version [33].

For *blockchain transaction networks*, we utilize the Stablecoin ERC20 Transactions Dataset [73] as our data source, which contains over 70 million transactions across six tokens on the Ethereum blockchain. We assume each account starts with an initial balance of $\rho_0 = \$10,000$, with a fraction borrowed according to a leverage ratio following a normal distribution ($\mu = 3, \sigma = 1$, bounded between 0 and 10). The MMR for each account is calculated daily using Equation 1, and we define the label as whether the MMR will fall below 0 on the next day. To simulate different market scenarios, we provide three datasets: (1) ERC-users (April 28 - May 25, 2022) encompasses approximately 1.5 million accounts for large-scale analysis; (2) ERC-active (May 18 - July 26,

TABLE II: Effectiveness evaluation results for account risk detection

Metrics	Datasets	Market-based					Tabular-based		Graph-based						
		FactorVAE	THGNN	iTransformer	TimeMixer	STAEformer	MLP	LightGBM	(α,β)-core	GFDN	HTGT	HOGRL	PREM	SLATE	RiskGuard
AUC score	ERC-user	0.8339	0.6917	0.9373	0.9382	0.9046	0.5259	0.9253	0.7425	OOT	0.6972	OOM	0.7428	OOM	**0.9875**
	ERC-active	0.7759	0.6248	0.9273	0.9276	0.7983	0.7248	0.9345	0.7401	OOT	0.6700	OOM	0.7019	OOM	**0.9698**
	ERC-low	0.5556	0.5387	0.5142	0.5208	0.5288	0.5446	0.7284	0.5503	OOT	0.5756	OOM	0.5280	OOM	**0.9470**
	SimA-chat	0.6274	0.6232	0.6250	0.6189	0.5651	0.6933	0.5778	0.5479	0.5930	0.6902	0.6937	0.5233	0.7022	**0.7076**
	COVID-19	0.6640	0.6909	0.7267	0.6601	0.7400	0.5844	0.7353	0.5930	OOM	OOT	OOM	0.5002	OOM	**0.7569**
	Bear	0.6497	0.5243	0.6789	0.5397	0.5632	0.5281	0.5619	0.5272	OOM	OOT	OOM	0.5100	OOM	**0.7223**
Improvement		16.41%	23.29%	11.36%	14.76%	16.52%	24.83%	10.47%	23.17%	11.46%	24.47%	1.39%	26.42%	0.54%	
F1 score	ERC-user	0.7474	0.7857	0.6947	0.6778	0.7081	0.8771	0.9388	0.5049	OOT	0.8821	OOM	0.5430	OOM	**0.9730**
	ERC-active	0.8453	0.8326	0.6245	0.5743	0.7041	0.8858	0.9242	0.5098	OOT	0.9009	OOM	0.1032	OOM	**0.9602**
	ERC-low	0.7510	0.6304	0.6230	0.2184	0.5486	0.8997	0.9459	0.1022	OOT	0.8832	OOM	0.2255	OOM	**0.9471**
	SimA-chat	0.5815	0.6524	0.5238	0.3498	0.6940	0.9171	0.9106	**0.9574**	0.9114	0.8996	0.8362	0.7864	0.7364	0.9467
	COVID-19	0.8448	0.7950	0.8491	0.8966	0.8836	0.7998	0.9208	0.7665	OOM	OOT	OOM	0.0041	OOM	**0.9499**
	Bear	0.8028	0.8619	0.7872	0.8127	0.8842	0.7895	0.8491	0.7522	OOM	OOT	OOM	0.0028	OOM	**0.8997**
Improvement		18.40%	18.64%	26.24%	35.78%	20.90%	8.46%	3.12%	34.73%	3.53%	6.53%	11.05%	66.86%	21.03%	

2022) includes only accounts with at least two transactions to ensure higher activity levels; and (3) ERC-low (August 27 - September 22, 2022) contains only accounts with asset positions below $80 to focus on low-risk scenarios.

For *financial portfolio holdings*, we utilize both real-world and simulated datasets. For real-world data, we employ the FAR-Trans Dataset [15] as our primary data source, which contains pricing data for equities, stocks, and mutual funds, along with 388,049 records of investment actions. We set personalized leverage ratios following different distributions (with μ ranging from 3 to 6 and σ ranging from 1 to 1.6) according to their risk level. We provide two datasets from this source: (1) COVID-19 dataset, extracted from transactions between August 26th, 2019 and April 7th, 2021, when markets experienced a stark decline due to COVID-19; and (2) BEAR dataset, covering transactions from April 8th, 2021 to November 19th, 2022, when markets faced several uncertainties such as the ongoing pandemic and the Russia-Ukraine war. For simulated data, we use real stocks in the CSI 300 index [35] from August 1st, 2023, to May 31st, 2024. Since account-level trading data in the stock market is not publicly available, we simulate transaction behaviors using a Large Language Model (LLM) [43], [74]. Initially, we generate attributes for an account, including age, education level, and financial capacity. These attributes are then provided to an LLM to generate a backstory for the account. Subsequently, another LLM utilizes this backstory to simulate trading on each trading day. In detail, the account-LLM receives information over the past 7 days of his positions and 50 randomly selected stocks. For each stock, features such as prices, trading volumes, and other attributes are provided as input to the LLM. To demonstrate that the LLM-simulated trading behavior resembles human behavior, we evaluate herd behavior using the LSV measure [34], which is widely used in human transactions analysis: $\text{LSV} = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{|\mathcal{A}|}\frac{1}{|\mathcal{A}|}|p(i,t) - \frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|}p(i,t)|$, where LSV is an indicator of herd behavior, $\mathcal{A}$ denotes the asset set, and $p(i,t)$ denotes the ratio of buy transactions to total transactions for asset i at time t. A significantly positive LSV value indicates the presence of herd behavior. We computed LSV across three financial portfolio datasets: COVID-19 (LSV = 32.6691), BEAR (LSV = 43.4913), and SimA-Chat (LSV = 26.7837). All three values are statistically significant ($p < 0.001$), confirming the presence of herd behavior. Notably, the SimA-Chat dataset, generated by our LLM simulation, exhibits a comparable level of herd behavior to the real-world datasets, demonstrating that the simulated dataset successfully captures human-like trading patterns.

Implementation Details. For GLUE, we set the time window to 2, $a = 0.6$, and $b = 0.1$ in most cases. When performing auxiliary asset prediction on ERC datasets, we set $a = 0.1$ and $b = 0.1$ due to GPU memory constraints. We set the hidden embedding dimension to 256, the number of epochs to 6, and the number of TANet layers to 2. We use LeakyReLU as the activation function and float32 operations to accelerate the training procedure and reduce memory usage. We evaluate performance using AUC-ROC and micro-F1 scores [75]. Experiments are conducted on a server with an Intel (R) Xeon (R) Gold 6342 CPU, 2× GeForce RTX 4090 GPUs, and 512 GB of RAM. An algorithm is terminated if its runtime exceeds 8 hours. Our source code is available at: https://github.com/shenmuxing/RiskGuard.

B. Compared Methods

We compare RiskGuard with the following baselines, including 1) market-based methods; 2) tabular-based methods, and 3) graph-based methods.

Market-based methods. We compare the following representative market-based baselines.

- FatorVAE [23] integrates the dynamic factor model with the variational autoencoder to learn the stock risk.
- THGNN [24] uses a temporal and heterogeneous GNN to predict the stock pattern. We build a temporal graph for assets first, in this way we predict asset fluctuations and then aggregate the holdings of all high-risk stocks in an account to calculate its risks [31], [32].
- iTransformer [1] inverts dimensions in the attention and feed-forward network for asset prediction.
- TimeMixer [2] employs an MLP-based architecture with Past-Decomposable-Mixing and Future-Multipredictor-Mixing blocks for asset pattern forecasting.
- STAEformer [3] applies spatio-temporal attention to capture information across assets and time.

We first train these models to predict stock patterns and then aggregate the holdings of all high-risk stocks in an account to calculate its risk, following prior works [31], [32].

Tabular-based approaches. We compare the following two classical models capable of tabular data.

- MLP [26] employs multiple fully connected layers to classify whether an account is at risk.
- LightGBM [25] is a gradient-boosting framework used to classify whether an account is at risk.

Graph-based methods. We include the following graph-based baselines for comparison.

- (α, β)-core [22] is the maximal subgraph where upper and lower nodes have degrees at least α and β, respectively. We optimize parameters on training data and classify all accounts within the subgraph as at risk.
- GFDN [22] captures community-level account features and classifies accounts.
- HTGT [27] leverages dynamic heterogeneous graphs and a heterogeneous graph neural network to classify nodes. [1]
- HOGRL [19] constructs a high-order transaction graph to capture multi-hop features and classify accounts.
- PREM [4] compromises a pre-processing module and an egoneighbor matching module to eliminate training cost on learning accounts at risk.
- SLATE [11] transforms TBG into multi-layer graphs and incorporates the spectral properties of their associated supra-Laplacian matrix to classify accounts.

C. Effectiveness Evaluation

Exp-1: Effectiveness results. In this experiment, we compare RiskGuard with the baselines using both the AUC score and the F1-score. The results are in Table II. Among market-based models, iTransformer outperforms FactorVAE, THGNN, Time-Mixer, and STAEformer by 5.05%, 11.93%, 3.4%, and 5.16% in terms of AUC score, respectively. However, in terms of F1-score, FactorVAE surpasses THGNN, iTransformer, Time-Mixer, and STAEformer by 0.24%, 7.84%, 17.38%, and 2.5%, respectively. For tabular-based methods, LightGBM consistently outperforms MLP across both metrics, achieving 14.36% higher AUC score and 5.34% higher F1-score. For the graph-based methods. GFDN, HOGRL, and SLATE are unable to handle large-scale graphs. GFDN fails to complete within 8 hours (OOT) on three large-scale datasets, ERC-user, ERC-active, and ERC-low, and runs out of memory (OOM) on COVID-19 and Bear. HOGRL runs out of memory on these datasets (OOM). Among graph-based methods that can handle most datasets, HTGT achieves relatively better performance on ERC and SimA-chat datasets with average AUC and F1-score, though it also experiences timeout issues on COVID-19 and Bear datasets. (α, β)-core maintains moderate but consistent performance, achieving 3.26% higher AUC and 32.13% F1-score than PREM. Overall, RiskGuard significantly outperforms all baselines. In terms of AUC, RiskGuard achieves an average performance gain of 11.36%, 10.47%, and 23.17% compared to iTransformer, LightGBM, and (α, β)-core. For F1-score, RiskGuard average surpasses FactorVAE, LightGBM, and (α, β)-core by 18.40%, 3.12%, and 34.73%, respectively.

[1]HOGRL does not provide official code, so we implement it by https://github.com/AI4Risk/antifraud

TABLE III: Efficiency results (in seconds)

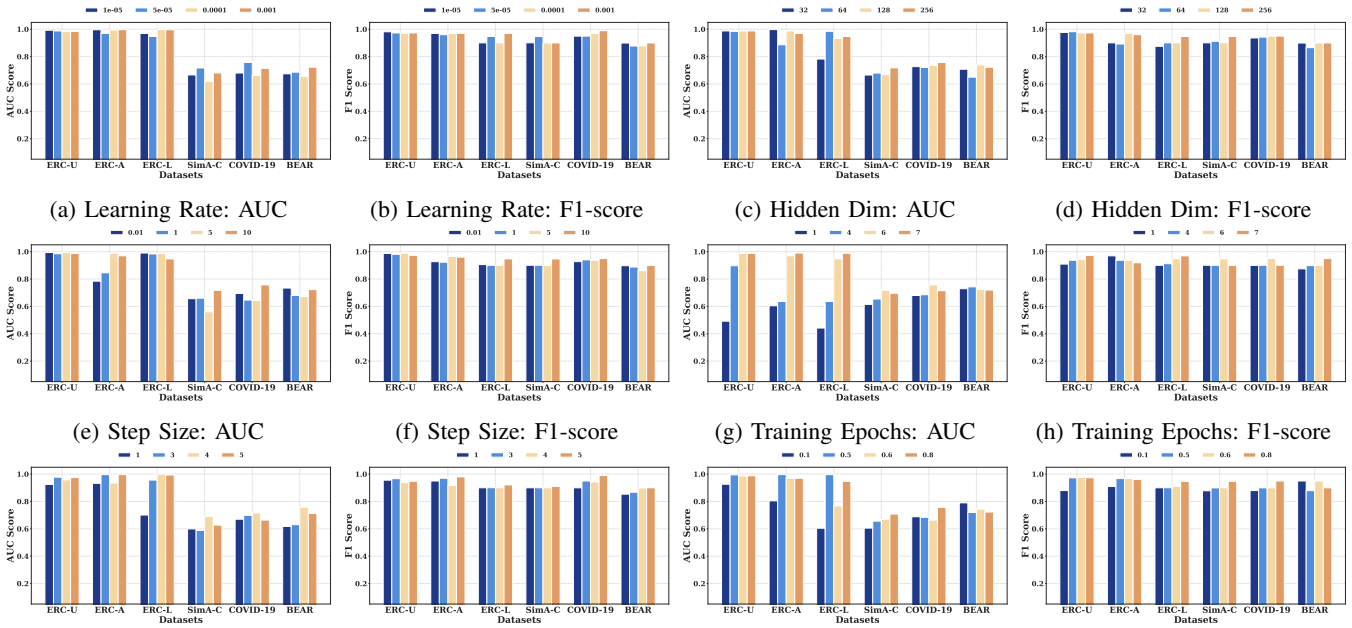
Time (s)	(α, β)-core	GFDN	HTGT	HOGRL	PREM	SLATE	RiskGuard
ERC-user	194.969	OOT	13668.78	OOM	3607.88	OOM	7520.4
ERC-active	330.229	OOT	24495.45	OOM	5580.29	OOM	17634.05
ERC-low	288.823	OOT	20649.27	OOM	4209.2	OOM	11349.18
SimA-chat	1.6622	244.77	1527.71	1385.31	32.4	1926.89	1004.22
COVID-19	224.83	OOM	OOT	OOM	2343.74	OOM	14496.36
BEAR	233.93	OOM	OOT	OOM	2407.84	OOM	15332.4

TABLE IV: Ablation study

Metrics	Datasets	Framework Variants					
		RiskGuard-S	RiskGuard-U	RiskGuard-AD	RiskGuard-A	RiskGuard-T	RiskGuard
AUC	ERC-U	0.9043	0.9425	0.9818	0.9543	0.9771	**0.9875**
	ERC-A	0.5434	0.9498	0.9569	0.8695	0.9603	**0.9698**
	ERC-L	0.7859	0.9327	0.934	0.7581	0.8609	**0.947**
	SimA-C	0.6139	0.6676	0.6899	0.6427	0.6798	**0.7076**
	COVID-19	0.5576	0.646	0.6004	0.708	0.661	**0.7569**
	Bear	0.6225	0.6161	0.7187	0.7074	0.6891	**0.7223**
F1 Score	ERC-U	0.8607	0.937	0.9719	0.9247	0.9309	**0.973**
	ERC-A	0.8341	0.9004	0.9496	0.9207	0.9294	**0.9602**
	ERC-L	0.8996	0.9003	0.8804	0.9001	0.8841	**0.9471**
	SimA-C	0.8924	0.8844	0.9	0.9015	0.9001	**0.9467**
	COVID-19	0.8706	0.8921	0.8766	0.8996	0.898	**0.9499**
	Bear	0.8658	0.8663	0.8995	0.8892	**0.9493**	0.8997

D. Efficiency Evaluation

Exp-2: Efficiency results. In this experiment, we evaluate the efficiency of RiskGuard. We compare RiskGuard with existing graph-based approaches, including the (α, β)-core, GFDN, HTGT, and HOGRL. The results are summarized in Table III. The (α, β)-core records the average time required for querying a single core, while the other methods report the total time of training. As shown in the table, RiskGuard shows a competitive efficiency. The (α, β)-core is an algorithm-based method for computing subgraphs without training, while PREM trains based on (α, β)-core and contrastive learning, allowing them to achieve higher efficiency. However, their accuracy is lower than that of RiskGuard. GFDN encounters OOT errors on ERC and OOM errors on COVID-19 and BEAR. This is because GFDN aggregates (α, β)-core information as feature columns, facing scalability challenges in temporal graphs and structural imbalance. HOGRL fails with OOM errors across all five datasets due to its high-order neighborhood aggregation through adjacency matrix multiplication, limiting its capacity to approximately 100,000 nodes. SLATE also encounters OOM errors on all datasets as it requires constructing a memory-intensive Supra-Laplacian Matrix for temporal graph embedding. Among the learning-based methods, RiskGuard demonstrates superior efficiency with an average speedup of 1.61× over HTGT on ERC and SimA-chat datasets. HTGT encounters OOT errors on COVID-19 and BEAR due to the low efficiency of heterogeneous spatial attention and temporal attention. This performance gain primarily results from the efficiency of GLUE, which selects a small yet representative set of nodes, enabling faster training.

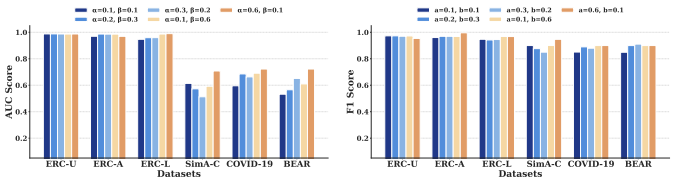


(a) Learning Rate: AUC (b) Learning Rate: F1-score (c) Hidden Dim: AUC (d) Hidden Dim: F1-score

(e) Step Size: AUC (f) Step Size: F1-score (g) Training Epochs: AUC (h) Training Epochs: F1-score

(i) Time Windows: AUC (j) Time Windows: F1-score (k) Training Size: AUC (l) Training Size: F1-score

Fig. 5: Sensitivity analysis of hyperparameters and training configurations.



(a) Sampling weights: AUC (b) Sampling weights: F1-score

Fig. 6: Hyperparameter analysis for sampling weights (a, b)

E. Ablation Study

Exp-3: Ablation study. Here we perform an ablation study to assess the effectiveness of the core components proposed in RiskGuard, including the GLUE sampler, the auxiliary learning paradigm, and TANet. Specifically, we define the following variants:1)RiskGuard-S, which replaces the GLUE with a random walk sampling method [29]; 2) RiskGuard-U, which replaces the GLUE with the uniform sampling method [30]; 3) RiskGuard-AD, which replaces the GLUE with the adaptive sampler [16] by sample different groups of assets predicted as high or low variance, and dynamically adjusts sampling weights for each group using momentum-based updates on group losses. 4) RiskGuard-A, which frees the model from the process of the auxiliary task; and 5) RiskGuard-T, which replaces the temporal layers with GRU, following [7]. The results are presented in Table IV. The table shows that all these components contribute to the high performance of RiskGuard. Specifically, the GLUE can bring an average performance gain of 8.94% and 5.27% in terms of AUC-ROC score and F1 score, respectively. The auxiliary learning paradigm can bring an average performance gain of 7.52% and 4.02% in terms of AUC-ROC score and F1 score, respectively. The TANet can bring an average performance gain of 4.38% and 3.08% in terms of AUC-ROC score and F1 score, respectively. These results collectively highlight the effectiveness of the components designed in RiskGuard.

F. Hyper-parameter Analysis

Exp-4: Learning rate analysis. We evaluate RiskGuard with learning rates of 1.00e-05, 5.00e-05, 1.00e-04, and 1.00e-03, respectively. The AUC and F1 results are shown in Figure 5(a), (b). For all four datasets, learning rates of 1.00e-05, 5.00e-05, 1.00e-04, and 1.00e-3 show stable performance in terms of both AUC and F1-score. The results show the robustness of RiskGuard to various learning rates.

Exp-5: Hidden layer dimension analysis. We evaluate RiskGuard with hidden layer dimensions of 32, 64, 128, and 256, respectively. The AUC and F1 results are shown in Figure 5(c), (d). Across all four datasets, the AUC and F1 scores demonstrate relatively stable performance for hidden layer dimensions of 32, 64, 128, and 256.

Exp-6: Step size of GLUE analysis. We investigate the step size γ by testing values of 0.01, 1, 5, and 10. Results in Figure 5(e), (f) show that while ERC-user and ERC-low exhibit slight AUC decreases with increasing γ, ERC-active and SimA-chat benefit from larger step sizes, with higher γ values generally improving performance across datasets.

Exp-7: Number of training epoch analysis. We set the number of training epochs as 1, 4, 6, and 7, respectively, and show results in Figure 5(g), (h). The AUC score suggests that increasing the number of epochs improves performance.

Exp-8: Time window analysis. We set the time windows with values of 1, 3, 4, and 5, respectively, and report the results in Figure 5(i), (j). Generally, longer time windows are associated with higher AUC scores, indicating improved performance.

Exp-9: Training size analysis. We set the training ratio of 0.1, 0.5, 0.6, and 0.8 and show results in Figure 5(k), (l). AUC and F1 scores generally increase with higher training ratios.

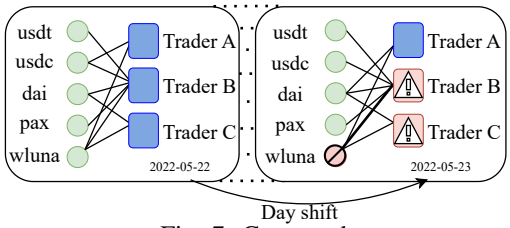


Fig. 7: Case study

Exp-10: Sampling weights (a, b) analysis. We evaluate sampling weight combinations (a, b) across (0.1, 0.1), (0.2, 0.3), (0.3, 0.2), (0.1, 0.6), and (0.6, 0.1) and show results in Figure 6 (a) and (b). Higher a and b values improve performance by increasing the neighborhood sampling ratio.

Exp-11: Case study. To better illustrate the effectiveness of RiskGuard, we perform case studies on the ERC-user dataset. Figure 7 shows that accurate account risk detection requires consideration of both market fluctuations and account holdings. Specifically, on May 22, 2022, five tokens were identified as normal, with three accounts holding these tokens. On the following day, however, the wluna token experienced significant fluctuations. Trader B, who increased his holdings of wluna, became exposed to greater risk. Though Trader C did not change his position, he remained exposed to risk. This case shows that RiskGuard can effectively identify account risks by analyzing both transactions and market fluctuations.

VI. Conclusion

In this paper, we present RiskGuard, a novel framework for account risk detection within financial bipartite graphs. In RiskGuard, an auxiliary asset prediction paradigm is integrated to jointly predict asset fluctuations and account risks. This enables the model to learn fluctuation-aware representations of assets, significantly improving prediction accuracy. For scalability, we design GLUE, an online graph sampling mechanism that dynamically adjusts sampling weights based on gradient entropy and graph structure. Our extensive experiments on four financial datasets demonstrate that RiskGuard achieves superior accuracy compared to existing methods with high efficiency for handling large graphs.

References

[1] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, "itransformer: Inverted transformers are effective for time series forecasting," in *ICLR*. OpenReview.net, 2024.
[2] S. Wang, H. Wu, X. Shi, T. Hu, H. Luo, L. Ma, J. Y. Zhang, and J. Zhou, "Timemixer: Decomposable multiscale mixing for time series forecasting," in *ICLR*. OpenReview.net, 2024.
[3] H. Liu, Z. Dong, R. Jiang, J. Deng, J. Deng, Q. Chen, and X. Song, "Spatio-temporal adaptive embedding makes vanilla transformer SOTA for traffic forecasting," in *CIKM*. ACM, 2023, pp. 4125–4129.
[4] J. Pan, Y. Liu, Y. Zheng, and S. Pan, "PREM: A simple yet effective approach for node-level graph anomaly detection," in *ICDM*. IEEE, 2023, pp. 1253–1258.
[5] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *WSDM*. ACM, 2020, pp. 519–527.
[6] Y. Wang, P. Li, C. Bai, and J. Leskovec, "TEDIC: neural modeling of behavioral patterns in dynamic social interaction networks," in *WWW*. ACM / IW3C2, 2021, pp. 693–705.
[7] J. You, T. Du, and J. Leskovec, "ROLAND: graph learning framework for dynamic graphs," in *KDD*. ACM, 2022, pp. 2358–2366.
[8] Y. Zheng, Z. Wei, and J. Liu, "Decoupled graph neural networks for large dynamic graphs," *Proc. VLDB Endow.*, vol. 16, no. 9, pp. 2239–2247, 2023.
[9] Y. Zhu, F. Cong, D. Zhang, W. Gong, Q. Lin, W. Feng, Y. Dong, and J. Tang, "Wingnn: Dynamic graph neural networks with random gradient aggregation window," in *KDD*. ACM, 2023, pp. 3650–3662.
[10] X. Qin, N. Sheikh, C. Lei, B. Reinwald, and G. Domeniconi, "SEIGN: A simple and efficient graph neural network for large dynamic graphs," in *ICDE*. IEEE, 2023, pp. 2850–2863.
[11] Y. Karmim, M. Lafon, and N. Thome, "Supra-laplacian encoding for transformer on dynamic graphs," in *NeurIPS*, 2024.
[12] D. Li, R. Jin, Z. Liu, B. Ren, J. Gao, and Z. Liu, "Towards reliable item sampling for recommendation evaluation," in *AAAI*. AAAI Press, 2023, pp. 4409–4416.
[13] X. Wang, Y. Xu, X. He, Y. Cao, M. Wang, and T. Chua, "Reinforced negative sampling over knowledge graph for recommendation," in *WWW*. ACM / IW3C2, 2020, pp. 99–109.
[14] Z. Zhang, Q. Liu, Z. Hu, Y. Zhan, Z. Huang, W. Gao, and Q. Mao, "Enhancing fairness in meta-learned user modeling via adaptive sampling," in *WWW*. ACM, 2024, pp. 3241–3252.
[15] J. Sanz-Cruzado, N. Droukas, and R. McCreadie, "Far-trans: An investment dataset for financial asset recommendation," *CoRR*, vol. abs/2407.08692, 2024.
[16] X. Chen, W. Fan, J. Chen, H. Liu, Z. Liu, Z. Zhang, and Q. Li, "Fairly adaptive negative sampling for recommendations," in *WWW*. ACM, 2023, pp. 3723–3733.
[17] 4210. Margin Requirements — FINRA.org. [Online]. Available: https://www.finra.org/rules-guidance/rulebooks/finra-rules/4210
[18] M. Bakoush, E. H. Gerding, and S. Wolfe, "Margin requirements and systemic liquidity risk," *J. Int. Financial Markets Inst. Money*, vol. 58, pp. 78–95, 2019.
[19] Y. Zou and D. Cheng, "Effective High-order Graph Representation Learning for Credit Card Fraud Detection," in *Proc. Int. Joint Conf. Artif. Intell.*, 2024, pp. 7581–7589.
[20] J. Li, Z. Li, J. Huang, J. Zhang, X. Wang, X. Lu, and J. Zhou, "Large-scale fake click detection for e-commerce recommendation systems," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 2595–2606.
[21] D. Wang, Y. Qi, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, and S. Yang, "A semi-supervised graph attentive network for financial fraud detection," in *Proc. IEEE Int. Conf. Data Mining*. IEEE, 2019, pp. 598–607.
[22] J. Yu, H. Wang, X. Wang, Z. Li, L. Qin, W. Zhang, J. Liao, and Y. Zhang, "Group-based fraud detection network on e-commerce platforms," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. ACM, 2023, pp. 5463–5475.
[23] Y. Duan, L. Wang, Q. Zhang, and J. Li, "Factorvae: A probabilistic dynamic factor model based on variational autoencoder for predicting cross-sectional stock returns," in *AAAI*, 2022, pp. 4468–4476.
[24] S. Xiang, D. Cheng, C. Shang, Y. Zhang, and Y. Liang, "Temporal and heterogeneous graph neural network for financial time series prediction," in *CIKM*. ACM, 2022, pp. 3584–3593.
[25] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017, pp. 3146–3154.
[26] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 249–270, 2022.
[27] Y. Fan, M. Ju, S. Hou, Y. Ye, W. Wan, K. Wang, Y. Mei, and Q. Xiong, "Heterogeneous temporal graph transformer: An intelligent system for evolving android malware detection," in *KDD*. ACM, 2021, pp. 2831–2839.
[28] T. Li, Z. Liu, Y. Shen, X. Wang, H. Chen, and S. Huang, "Master: Market-guided stock transformer for stock price forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 162–170.
[29] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. ACM SIGKDD*, 2014, pp. 701–710.
[30] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.
[31] S. Motie and B. Raahemi, "Financial fraud detection using graph neural networks: A systematic review," *Expert Syst. Appl.*, vol. 240, p. 122156, 2024.
[32] E. F. Fama and K. R. French, "Size, value, and momentum in international stock returns," *J. Financial Econ.*, vol. 105, no. 3, pp. 457–472, 2012.
[33] J. Zhao *et al.* (2025) Riskguard: Full. [Online]. Available: https://github.com/shenmuxing/RiskGuard/full_version
[34] S. Bikhchandani and S. Sharma, "Herd behavior in financial markets," *IMF Staff papers*, vol. 47, no. 3, pp. 279–310, 2000.

[35] H. Wang, T. Wang, S. Li, and S. Guan, "HATR-I: hierarchical adaptive temporal relational interaction for stock trend prediction," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 6988–7002, 2023.
[36] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," in *CIKM*, 2018, pp. 2077–2085.
[37] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12 012–12 038, 2023.
[38] Y. Yang, Y. Xu, Y. Sun, Y. Dong, F. Wu, and Y. Zhuang, "Mining fraudsters and fraudulent strategies in large-scale mobile social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 1, pp. 169–179, 2021.
[39] H. Chen, S. Joslin, and S. X. Ni, "Demand for Crash Insurance, Intermediary Constraints, and Risk Premia in Financial Markets," *The Review of Financial Studies*, vol. 32, no. 1, pp. 228–265, 2019.
[40] Y. Li, Y. Shen, L. Chen, and M. Yuan, "Zebra: When temporal graph neural networks meet temporal personalized PageRank," *Proc. VLDB Endow.*, vol. 16, no. 6, pp. 1332–1345, 2023.
[41] H. Fang, N. J. A. Harvey, V. S. Portella, and M. P. Friedlander, "Online Mirror Descent and Dual Averaging: Keeping Pace in the Dynamic Case," *J. Mach. Learn. Res.*, vol. 23, no. 121, pp. 1–38, 2022.
[42] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107–194, 2012.
[43] DeepSeek-AI, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," 2024.
[44] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
[45] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, 2022.
[46] L. Norden and M. Weber, "Credit Line Usage, Checking Account Activity, and Default Risk of Bank Borrowers," *The Review of Financial Studies*, vol. 23, no. 10, pp. 3665–3699, 2010.
[47] Y. Li, C. Stasinakis, and W. M. Yeo, "A hybrid xgboost-mlp model for credit risk assessment on digital supply chain finance," *Forecasting*, vol. 4, no. 1, pp. 184–207, 2022.
[48] L. Liu, Z. Pei, P. Chen, H. Luo, Z. Gao, K. Feng, and Z. Gan, "An Efficient GAN-Based Multi-classification Approach for Financial Time Series Volatility Trend Prediction," *Int. J. Comput. Intell. Syst.*, vol. 16, no. 1, p. 40, 2023.
[49] P. K. Nath, G. Waghmare, N. Tumbde, N. Kumar, and S. Asthana, "Learning temporal representations of bipartite financial graphs," in *ICAIF*. ACM, 2023, pp. 202–209.
[50] D. Zhou, A. Uddin, X. Tao, Z. Shang, and D. Yu, "Temporal bipartite graph neural networks for bond prediction," in *Proc. ACM Int. Conf. AI Finance*, 2022, pp. 308–316.
[51] H. Zhang, K. Zeng, and S. Lin, "Federated Graph Neural Network for Fast Anomaly Detection in Controller Area Networks," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1566–1579, 2023.
[52] Y. Ren, H. Zhu, J. Zhang, P. Dai, and L. Bo, "Ensemfdet: An ensemble approach to fraud detection based on bipartite graph," in *Proc. IEEE Int. Conf. Data Eng.* IEEE, 2021, pp. 2039–2044.
[53] H. Wang, K. Wang, W. Zhang, and Y. Zhang, "Bipartite graph analytics: Current techniques and future trends," in *Proc. IEEE Int. Conf. Data Eng.* IEEE, 2024, pp. 1–7.
[54] J. Yu, H. Wang, X. Wang, Z. Li, L. Qin, W. Zhang, J. Liao, Y. Zhang, and B. Yang, "Temporal Insights for Group-Based Fraud Detection on e-Commerce Platforms," *IEEE Trans. Knowl. Data Eng.*, vol. 37, no. 2, pp. 951–965, 2025.
[55] E. F. Fama and K. R. French, "The capital asset pricing model: Theory and evidence," *Journal of economic perspectives*, vol. 18, no. 3, pp. 25–46, 2004.
[56] X. Zhu, K. Tan, and Y. Wang, "Research on the identification and analysis of risk sources of financial enterprises based on mlp contribution analysis," in *Proc. Int. Conf. Mach. Learn. Big Data Bus. Intell.*, 2021, pp. 585–590.
[57] M. G. Morshed, T. Sultana, and Y.-K. Lee, "Lel-gnn: Learnable edge sampling and line based graph neural network for link prediction," *IEEE Access*, vol. 11, pp. 56 083–56 097, 2023.
[58] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," in *NeurIPS*, 2019, pp. 11 247–11 256.
[59] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *KDD*, 2019, pp. 257–266.
[60] M. Yoon, T. Gervet, B. Shi, S. Niu, Q. He, and J. Yang, "Performance-adaptive sampling strategy towards fast and accurate graph neural networks," in *KDD*, 2021, pp. 2046–2056.
[61] Z. Liu, Z. Wu, Z. Zhang, J. Zhou, S. Yang, L. Song, and Y. Qi, "Bandit samplers for training graph neural networks," in *NeurIPS*, 2020.
[62] W. Zhao, T. Guo, X. Yu, and C. Han, "A learnable sampling method for scalable graph neural networks," *Neural Networks*, vol. 162, pp. 412–424, 2023.
[63] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *ICML*, 2018, pp. 942–950.
[64] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, "Sparsification - a technique for speeding up dynamic graph algorithms," *J. ACM*, vol. 44, no. 5, pp. 669–696, 1997.
[65] X. He, K. Deng, X. Wang, Y. Li, Y.-D. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *Proc. SIGIR*. ACM, 2020, pp. 639–648.
[66] F. Wu, A. H. S. Jr, T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proc. ICML*, vol. 97. PMLR, 2019, pp. 6861–6871.
[67] K. Mao, J. Zhu, X. Xiao, B. Lu, Z. Wang, and X. He, "Ultragcn: Ultra simplification of graph convolutional networks for recommendation," in *Proc. CIKM*. ACM, 2021, pp. 1253–1262.
[68] W. Zhang, Z. Yin, Z. Sheng, Y. Li, W. Ouyang, X. Li, Y. Tao, Z. Yang, and B. Cui, "Graph attention multi-layer perceptron," in *Proc. KDD*. ACM, 2022, pp. 4560–4570.
[69] "Softmax function," Wikipedia, 2024.
[70] H. Xue, L. Yang, W. Jiang, Y. Wei, Y. Hu, and Y. Lin, "Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal rnn," in *Proc. ECML PKDD*, vol. 12457. Springer, 2020, pp. 282–298.
[71] J. Wang, K. Wang, X. Lin, W. Zhang, and Y. Zhang, "Efficient unsupervised community search with pre-trained graph transformer," *Proc. VLDB Endow.*, vol. 17, no. 9, pp. 2227–2240, 2024.
[72] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, "Long-tail learning via logit adjustment," in *Proc. ICLR*, 2021.
[73] K. Shamsi, F. Victor, M. Kantarcioglu, Y. Gel, and C. G. Akcora, "Chartalist: Labeled Graph Datasets for UTXO and Account-based Blockchains," *Proc. NeurIPS*, vol. 35, pp. 34 926–34 939, 2022.
[74] J. Wang, K. Wang, Y. Zhang, W. Zhang, X. Xu, and X. Lin, "On llm-enhanced mixed-type data imputation with high-order message passing," *CoRR*, vol. abs/2501.02191, 2025.
[75] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

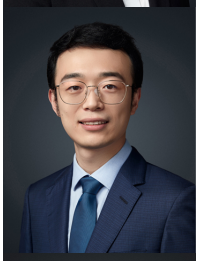
Jingye Zhao is currently a master's degree student at Antai College of Economics and Management, Shanghai Jiao Tong University. His research is focused on graph data analytics and reinforcement learning.

Jianan Shen is currently a bachelor's degree student at Antai College of Economics and Management, Shanghai Jiao Tong University. His research is focused on graph data analytics and financial applications.

Jianwei Wang is currently a PhD candidate in the School of Computer Science and Engineering, University of New South Wales. His research is focused on data quality, graph analytics, and scalable algorithms for dense subgraph discovery.

Tianyuan Zhou is currently a bachelor's degree student at Antai College of Economics and Management, Shanghai Jiao Tong University. His research is focused on graph data analytics and financial applications.

Ruijia Wu is currently an assistant professor at Antai College of Economics and Management, Shanghai Jiao Tong University. She received the BS and MS degrees in Mathematics from the University of Oxford, and the PhD degree in Statistics from the University of Pennsylvania in 2022. Her research interests lie in statistical machine learning, high-dimensional statistics, text analytics, and their applications.



Kai Wang is currently an associate professor at Antai College of Economics and Management, Shanghai Jiao Tong University. He received the BEng degree in Computer Science from Zhejiang University in 2016, and the PhD degree in Computer Science from the University of New South Wales in 2020. His research interests lie in big data analytics, especially for the graph/network and spatial data.

Xuemin Lin is a Chair Professor at Antai College of Economics and Management, Shanghai Jiao Tong University. He is Fellow of the IEEE, Member of Academia Europaea, and Fellow of Asia-Pacific Artificial Intelligence Association. His principal research areas are databases and graph visualization.

VII. APPENDIX

A. Proof of Theorem 1

For convenience, we restate Theorem 1 as follows:

Theorem 1. Under the entropy setup and the mirror descent algorithm, we have update process of W_i:

$$W_i \leftarrow (1+\delta)\cdot\text{softmax}\left(\left(W_i+\frac{\delta}{n}\right)\odot\left(\exp(-\frac{\eta}{1+\delta}\right)\right)-\frac{\delta}{n},$$

where $\odot$ denotes the Hadamard product, η is gradient information, and the softmax function is defined as in [69].

To prove Theorem 1, we derive the update rule for W_i using the mirror descent algorithm under the entropy setup.

Proof of Theorem 1. First, we recall some notations from the main text:

- We are working with the probability simplex $\Delta_n^+ = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = 1, x_i \geq 0 \text{ for all } i\}$;
- The distance-generating function (DGF) is the regularized entropy:

$$\omega(x) = (1+\delta)\sum_{i=1}^{n}\left(x_i+\frac{\delta}{n}\right)\ln\left(x_i+\frac{\delta}{n}\right); \tag{5}$$

- The Bregman divergence induced by ω is:

$$V_x(y) = \omega(y) - \omega(x) - \langle\nabla\omega(x), y-x\rangle. \tag{6}$$

Our goal is to solve the optimization problem:

$$W_i \leftarrow \underset{y\in\Delta_n^+}{\text{argmin}}\left\{V_{W_i}(y) + \langle\eta_i, y\rangle\right\}, \tag{7}$$

where η_i is the gradient of the loss function with respect to W_i.

Then, compute the gradient $\nabla\omega(x)$. For each component x_j of x, the partial derivative is:

$$\frac{\partial\omega(x)}{\partial x_j} = (1+\delta)\left(\ln\left(x_j+\frac{\delta}{n}\right)+1\right). \tag{8}$$

Next, compute $V_{W_i}(y)$:

$$\begin{aligned}
V_{W_i}(y) &= \omega(y) - \omega(W_i) - \langle\nabla\omega(W_i), y - W_i\rangle \\
&= (1+\delta)\sum_{j=1}^{n}\left(y_j+\frac{\delta}{n}\right)\ln\left(y_j+\frac{\delta}{n}\right) \\
&\quad - (1+\delta)\sum_{j=1}^{n}\left(W_{i,j}+\frac{\delta}{n}\right)\ln\left(W_{i,j}+\frac{\delta}{n}\right) \\
&\quad - (1+\delta)\sum_{j=1}^{n}\left(\ln\left(W_{i,j}+\frac{\delta}{n}\right)+1\right)(y_j - W_{i,j}) \\
&= (1+\delta)\sum_{j=1}^{n}\left[\left(y_j+\frac{\delta}{n}\right)\ln\left(\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}}\right) - (y_j - W_{i,j})\right]
\end{aligned}$$

Since $\sum_{j=1}^n(y_j - W_{i,j}) = 0$ (because both $y, W_i \in \Delta_n^+$), the last term sums to zero.

Therefore, the Bregman divergence simplifies to:

$$V_{W_i}(y) = (1+\delta)\sum_{j=1}^{n}\left(y_j+\frac{\delta}{n}\right)\ln\left(\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}}\right) \tag{9}$$

This is the generalized Kullback-Leibler divergence.

Substitute $V_{W_i}(y)$ back into the optimization problem (7):

$$\begin{aligned}
W_i \leftarrow \underset{y\in\Delta_n^+}{\text{argmin}}\Bigg\{&(1+\delta)\sum_{j=1}^{n}\left(y_j+\frac{\delta}{n}\right) \\
&\ln\left(\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}}\right) + \langle\eta_i, y\rangle\Bigg\}
\end{aligned} \tag{10}$$

1) Set Up the Lagrangian: Since we're minimizing over the simplex Δ_n^+, we need to consider the constraint $\sum_{j=1}^n y_j = 1$ and $y_j \geq 0$.

Form the Lagrangian $\mathcal{L}$ with the constraint incorporated using a multiplier λ:

$$\begin{aligned}
\mathcal{L}(y,\lambda,\gamma) = (1+\delta)&\sum_{j=1}^{n}\left(y_j+\frac{\delta}{n}\right)\ln\left(\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}}\right) \\
&+\sum_{j=1}^{n}\eta_{i,j}y_j - \lambda\left(\sum_{j=1}^{n}y_j - 1\right) - \sum_{j=1}^{n}\gamma_j y_j
\end{aligned} \tag{11}$$

where $\gamma_j \geq 0$ are the Lagrange multipliers for the non-negativity constraints $y_j \geq 0$.

Take the derivative of $\mathcal{L}$ with respect to y_j and set it to zero:

$$\frac{\partial\mathcal{L}}{\partial y_j} = (1+\delta)\left(\ln\left(\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}}\right)+1\right) + \eta_{i,j} - \lambda - \gamma_j = 0 \tag{12}$$

Since $\gamma_j \geq 0$ and $y_j \geq 0$, by complementary slackness, if $y_j > 0$, then $\gamma_j = 0$. And obviously, $y_j + \frac{\delta}{n} > 0$.

Thus, for $y_j > 0$, we have:

$$(1+\delta)\ln\left(\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}}\right) + 1 + \delta + \eta_{i,j} - \lambda = 0 \tag{13}$$

Solving for y_j:

$$\ln\left(\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}}\right) = \frac{\lambda-\eta_{i,j}}{1+\delta} - 1 \tag{14}$$

$$\frac{y_j+\frac{\delta}{n}}{W_{i,j}+\frac{\delta}{n}} = \exp\left(\frac{\lambda-\eta_{i,j}}{1+\delta} - 1\right) \tag{15}$$

$$y_j + \frac{\delta}{n} = \left(W_{i,j}+\frac{\delta}{n}\right)\exp\left(\frac{\lambda-\eta_{i,j}}{1+\delta} - 1\right) \tag{16}$$

Then we solve for the Lagrange Multiplier λ. Let $Z = \exp\left(\frac{\lambda}{1+\delta}\right)$ (we'll solve for Z instead of λ), then:

$$y_j + \frac{\delta}{n} = \left(W_{i,j}+\frac{\delta}{n}\right)\exp\left(-\frac{\eta_{i,j}}{1+\delta} - 1\right)Z \tag{17}$$

Sum both sides over j:

$$\sum_{j=1}^{n}\left(y_j+\frac{\delta}{n}\right)=Z\sum_{j=1}^{n}\left(W_{i,j}+\frac{\delta}{n}\right)\exp\left(-\frac{\eta_{i,j}}{1+\delta}-1\right)$$

Since $\sum_{j=1}^{n} y_j = 1$ and $\sum_{j=1}^{n}\frac{\delta}{n}=\delta$, and similarly for $W_{i,j}$:

$$1+\delta=Z(1+\delta)\sum_{j=1}^{n}\tilde{W}_{i,j}\exp\left(-\frac{\eta_{i,j}}{1+\delta}-1\right) \tag{18}$$

where $\tilde{W}_{i,j}=\frac{W_{i,j}+\frac{\delta}{n}}{1+\delta}$.
Simplify:

$$1+\delta=Z(1+\delta)\sum_{j=1}^{n}\tilde{W}_{i,j}\exp\left(-\frac{\eta_{i,j}}{1+\delta}-1\right) \tag{19}$$

$$1=Z\sum_{j=1}^{n}\tilde{W}_{i,j}\exp\left(-\frac{\eta_{i,j}}{1+\delta}-1\right) \tag{20}$$

Therefore:

$$Z=\left(\sum_{j=1}^{n}\tilde{W}_{i,j}\exp\left(-\frac{\eta_{i,j}}{1+\delta}-1\right)\right)^{-1} \tag{21}$$

Substitute back to get y_j:

$$y_j=\left(W_{i,j}+\frac{\delta}{n}\right)\exp\left(-\frac{\eta_{i,j}}{1+\delta}-1\right)Z-\frac{\delta}{n} \tag{22}$$

Recall that $\tilde{W}_{i,j}=\frac{W_{i,j}+\frac{\delta}{n}}{1+\delta}$, so:

$$y_j=(1+\delta)\tilde{W}_{i,j}\exp\left(-\frac{\eta_{i,j}}{1+\delta}-1\right)Z-\frac{\delta}{n} \tag{23}$$

Similarly, since $Z=1/\sum_{k=1}^{n}\tilde{W}_{i,k}\exp\left(-\frac{\eta_{i,k}}{1+\delta}-1\right)$:

$$y_j+\frac{\delta}{n}=\frac{(1+\delta)\tilde{W}_{i,j}\exp\left(-\frac{\eta_{i,j}}{1+\delta}\right)}{\sum_{k=1}^{n}\tilde{W}_{i,k}\exp\left(-\frac{\eta_{i,k}}{1+\delta}\right)} \tag{24}$$

Multiply the numerator and the denominator by $(1+\delta)$:

$$\begin{aligned}y_j+\frac{\delta}{n}&=\frac{(1+\delta)\left(W_{i,j}+\frac{\delta}{n}\right)\exp\left(-\frac{\eta_{i,j}}{1+\delta}\right)}{\sum_{k=1}^{n}(W_{i,k}+\frac{\delta}{n})\exp\left(-\frac{\eta_{i,k}}{1+\delta}\right)}\\&=\frac{(1+\delta)(W_{i,j}+\frac{\delta}{n})\exp\left(-\frac{\eta_{i,j}}{1+\delta}\right)}{\sum_{k=1}^{n}(W_{i,k}+\frac{\delta}{n})\exp\left(-\frac{\eta_{i,k}}{1+\delta}\right)}\end{aligned} \tag{25}$$

Therefore, the updated W_i is:

$$W_{i,j}\leftarrow\frac{(1+\delta)(W_{i,j}+\frac{\delta}{n})\exp\left(-\frac{\eta_{i,j}}{1+\delta}\right)}{\sum_{k=1}^{n}(W_{i,k}+\frac{\delta}{n})\exp\left(-\frac{\eta_{i,k}}{1+\delta}\right)}-\frac{\delta}{n} \tag{26}$$

This completes the proof. □

B. Proof of Theorem 3

For convenience, we restate Theorem 3 as follows:

Theorem 3. Using gradient entropy-based sampling, the error $\|\nabla f(x)-\nabla\hat{f}(x)\|_1$ is upper bounded by $\sqrt{\frac{4(1-a)c^2\log K}{n}}\cdot\sum_k g_k$ with probability at least $1-\frac{2}{K}$. Here k is k-th element of the gradient ∇; $c=\max\{1,|\frac{a+b-1}{b}|\}$; g_k is maximum element of $|\nabla_k f(x)|$; and K is the length of $\nabla f(\cdot)$.

Proof of Theorem 3. Considering the error in the k-th component:

$$\mathscr{E}_A(d)_k=\left|\frac{1}{n}\sum_{i\in B}\nabla_k f_i(x)-\frac{1-a}{bn}\sum_{j\in B_r}\nabla_k f_j(x)\right|. \tag{27}$$

Let $Y_j=\nabla_k f_j(x)$ for $j\in B$, $\widetilde{Y}_j=\frac{1}{n}\nabla_k f_i(x)$ for $j\in B-B_r$ and $\widetilde{Y}_j=\left(\frac{1}{n}-\frac{1-a}{bn}\right)\nabla_k f_i(x)$ for $j\in B_r$. Since $\mathbb{E}(\widetilde{Y}_j)=0$ for all $j\in B$, applying Hoeffding's inequality for independent random variables $\widetilde{Y}_j\in\left[-\frac{cg_k}{n},\frac{cg_k}{n}\right]$, we obtain:

$$P\left(|\mathscr{E}_A(d)_k|\geq t\right)=P\left(|\sum_{i\in B}\tilde{Y}_i|\geq t\right) \tag{28}$$

$$\leq 2\exp\left(-\frac{2t^2}{(1-a)n\left(\frac{2cg_k}{n}\right)^2}\right). \tag{29}$$

Setting $t=\sqrt{\frac{4(1-a)c^2g_k^2\log K}{n}}$, where K is the total number of components, implies that with probability $1-\frac{2}{K^2}$, $|\mathscr{E}_A(d)_k|$ is upper bounded by $\sqrt{\frac{4(1-a)c^2g_k^2\log K}{n}}$ for any component k. In addition, notice that

$$\|\nabla f(x)-\nabla\hat{f}(x)\|_1=|\sum_{k=1}^{K}\mathscr{E}_A(d)_k|.$$

Therefore, with probability $1-\frac{2}{K}$, the error term is

$$\mathscr{E}_A(d)=\sum_k|\mathscr{E}_A(d)_k|\leq\sqrt{\frac{4(1-a)c^2\log K}{n}}\cdot\sum_k g_k, \tag{30}$$

as required. □

C. More details on Experiment

For *blockchain transaction networks*, we utilize the Stablecoin ERC20 Transactions Dataset [73] as our data source, which contains over 70 million transactions across six tokens on the Ethereum blockchain. We assume that each account starts with an initial currency balance, denoted as ρ_0, at the beginning of each simulation. For this study, ρ_0 is set to \$10,000. A fraction of this currency is borrowed, with the leverage ratio following a normal distribution ($\mu=3,\sigma=1$), bounded between 0 and 10. We simulate account activities by transaction records. The MMR for each account is calculated daily using Equation 1. We define the label as whether the MMR of an account will fall below 0 on the next day. To simulate different market scenarios, e.g., many participants, high activity, or a low probability of margin calls, we provide three datasets as follows:

Algorithm 2: Algorithm for LLM-based Trading Simulation Generator

Input: Stock market data from Aug 1st, 2023 to May 31st, 2024, Number of participants N
Output: Trading simulation results for all participants
// **Stage 1: User Profile Generation**
// Original prompts are in Chinese for DeepSeek optimization
Prepare system prompt and few-shot examples for persona generation
for $i = 1$ **to** N **do**
  Sample $age_i \sim$ Poisson($\mu = 35$), $gender_i \sim$ Bernoulli($p = 0.35$)
  Sample $wealth_i$, $return\_expectation_i$, $risk\_tolerance_i$ from predefined distributions
  Create user description combining all characteristics
  Generate detailed persona backstory using LLM with system prompt
  Assign $initial\_capital_i$ based on wealth level (10K-5M CNY)
  Store complete user profile
// **Stage 2: Trading Simulation**
for $i = 1$ **to** N **do**
  Initialize $cash_i \leftarrow initial\_capital_i$, empty holdings, trade history
  for *each trading day* t **do**
    Collect past 7 days stock data and randomly select 50 stocks for observation
    Get data for any existing holdings
    Calculate margin requirements for short positions
    if $cash_i < margin\_requirement_i$ **then**
      Break simulation (bankruptcy)
    Format market data, portfolio info, and cash status into trading prompt
    Add margin warning if $cash_i < 1.5 \times margin\_requirement_i$
    Send combined persona and market context to LLM
    Parse LLM response to get JSON-formatted trading actions
    for *each trading action* **do**
      Execute buy/sell/short orders based on action direction and stock
      Update holdings, cash, and margin requirements accordingly
      Record transaction in trade history
    Calculate total portfolio value and record daily status
  Compile simulation results for participant i
return *All participants' trading histories and portfolio values*

1) ERC-users: This dataset is extracted using transactions between April 28th, 2022, and May 25th, 2022. This dataset is noteworthy for its scale, encompassing approximately 1.5 million accounts.
2) ERC-active: This dataset is extracted using transactions between May 18th, 2022, and July 26th, 2022. Accounts with fewer than two transactions were excluded from the dataset, ensuring that the remaining accounts are relatively more active.
3) ERC-low: This dataset includes transactions from August 27 to September 22, 2022. Accounts with any asset position exceeding $80 are excluded, ensuring only low-position accounts with relatively low risk.

We provide the details of the SimA-Chat dataset in Appendix of the revised paper, *For financial portfolio holdings, we use real stocks in the CSI 300 index [35] from Aug 1st, 2023 to May 31st, 2024. Since account-level trading data in the stock market is not publicly available, we simulate transaction behaviors using a two-stage Large Language Model (LLM) approach [43].*

Stage 1: User Profile Generation. *We first generate diverse user profiles by sampling demographic and financial characteristics from predefined distributions. Specifically, we sample age from a Poisson distribution (μ=35), gender with 35% probability of being female, wealth levels from five categories (poor, average, comfortable, wealthy, extremely wealthy) with weights [0.1, 0.25, 0.45, 0.15, 0.05], return expectations from four levels ranging from "no concern about returns" to "must pursue high returns" with weights [0.05, 0.2, 0.45, 0.3], and risk tolerance from four levels with weights [0.1, 0.3, 0.3, 0.3]. These sampled attributes are then fed to an LLM with a carefully designed system prompt that instructs it to generate a detailed persona backstory. The system prompt includes few-shot examples and explicitly requests diverse characterizations to avoid repetitive patterns. For instance, a 28-year-old wealthy male seeking high returns might be characterized as either a lottery winner seeking to prove himself or an experienced investor with a systematic approach.*

Stage 2: Trading Simulation. *Each generated persona serves as a system prompt for the trading simulation. On each trading day, the account-LLM receives: (1) the persona backstory, (2) information about the past 7 days of the account's positions, (3) data for 50 randomly selected stocks, including prices, trading volumes, and other market attributes, and (4) current cash and margin requirements. The LLM is prompted to respond in a structured JSON format specifying trading actions (buy/sell/short), target stocks (by code), and position directions. We implement realistic trading constraints, including margin requirements for short positions, position sizing based on portfolio value, and forced liquidation when margin falls below 150% of requirements. Initial capital is assigned based on wealth levels, ranging from 10,000-50,000 CNY for "poor" users to 1,000,000-5,000,000 CNY for "extremely wealthy" users.*