UNIVERSITY OF CALIFORNIA
Los Angeles

Fraud Detection with Asymmetric Costs in Online Talent Marketplaces: A Comparative Study of Machine Learning Approaches

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics and Data Science

by

Anshuman Mahalley

2026

ABSTRACT OF THE THESIS

Fraud Detection with Asymmetric Costs in Online Talent Marketplaces: A Comparative Study of Machine Learning Approaches

by

Anshuman Mahalley
Master of Applied Statistics and Data Science
University of California, Los Angeles, 2026
Professor Guang Cheng, Chair

This study compares machine learning approaches for detecting cheating in online interviews from a production dataset using anonymized behavioral features and social network data from ∼272,000 candidates. The dataset presents severe class imbalance, a large volume of unlabeled observations with weak labels, and a sparse social graph with over 1.7 million edges. We engineer 36 predictive features – including missingness indicators and graph-derived metrics – and evaluate three approaches: an XGBoost baseline, a semi-supervised XGBoost extension with differentiated sample weighting, and a GraphSAGE neural network. Models are evaluated under a cost-based metric with asymmetric penalties, where missing a cheater ($600) is twice as costly as incorrectly blocking a legitimate candidate ($300). The XGBoost model achieved the best cost score, outperforming both the semi-supervised model and GraphSAGE. Threshold analysis reveals auto-blocking is only cost-effective above 96.77% confidence. Feature importance analysis shows that feature missingness patterns strongly correlate with cheating, and handcrafted graph features outperform end-to-end graph learning in sparse networks. These results highlight the continued effectiveness of gradient boosting on tabular fraud detection tasks and the importance of aligning model evaluation with real-world cost structures.

The thesis of Anshuman Mahalley is approved.

Yingnian Wu

Yuhua Zhu

David Anthony Zes

Guang Cheng, Committee Chair

University of California, Los Angeles

2026

To my family and friends - thank you for the support, patience, and for tolerating far more statistics talk than anyone should have to.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

The rapid rise of large language models (LLMs) and substantial improvements in artificial intelligence have enabled students and knowledge workers globally to increase their productivity dramatically. Modern foundation models, trained on vast datasets, demonstrate high-level proficiency in specialized domains such as code generation, logical reasoning, and information retrieval. This surge in AI capabilities has resulted in billions of dollars in venture capital investment toward foundational model companies and significantly increased the demand for high-quality data labeling services provided by companies such as Scale AI and Mercor. Mercor, an AI startup that is valued at approximately $10 billion, has a marketplace of highly skilled experts that provide high-quality specialized training data for frontier AI models. Mercor manages a network of over 30,000 contractors and collectively pays over $1M a day to these experts to complete various training tasks.

As such, it is highly important that they onboard skilled domain experts onto the Mercor platform. However, the same AI tools that enhance productivity have also enabled job seekers to utilize AI assistants to bypass technical screening interviews. Consequently, major technology firms have been forced to adapt their hiring processes to mitigate AI-assisted fraud. In this context, cheating detection has emerged as a critical operational challenge; the onboarding of unqualified candidates represents a substantial loss in both capital and time.

The objective of this study is to develop a predictive model to identify whether a candidate in an online talent marketplace is engaging in cheating behavior during an interview.

This analysis utilizes anonymized activity features and behavioral signals. Cheating detection in real-life environments is a challenging problem because most labeled cases of known cheating come from candidates who are flagged by manual review processes. These processes are not only labor-intensive and time-consuming but also subject to human error and high operational costs. As candidate volume scales, marketplaces like Mercor require automated machine learning techniques to augment and enhance traditional detection methods.

While fraud detection is an established domain already, the talent marketplace dynamic presents unique statistical challenges. These include severe class imbalance as confirmed instances of cheating are relatively scarce compared to the broader candidate pool, and the necessity of analyzing social graph data to identify collaborative cheating networks. Furthermore, the problem is constrained by cost optimization. In a real-world production environment, a model cannot simply optimize for standard accuracy; it must account for operational impact. False negatives (cases where the system misses confirmed cheating) and false positives (incorrectly blocking legitimate candidates) have a negative cost associated with them. Finally, the collaborative nature of cheating allows for the exploration of graph-based feature engineering and Graph Neural Network (GNN) architectures to build a fraud detection model.

This work contributes a comparative analysis of supervised, semi-supervised, and graph-based approaches under a realistic cost-based objective function, evidence that handcrafted graph features outperform end-to-end graph learning in sparse networks, and an analytical derivation of decision thresholds from cost asymmetries. By employing state-of-the-art algorithms, such as XGBoost, Semi-Supervised Learning with sample weighting and GNNs, this study seeks to identify the most effective approaches for accurately classifying candidate cheating. The findings of this study have practical implications for talent marketplaces and other industries where user activity data can be analyzed to derive meaningful insights on how to improve the screening process.

CHAPTER 2

Background & Dataset

2.1 Datasets

The primary data utilized in this study is derived from the "Mercor Cheating Detection" competition hosted on Kaggle. The data files include real-world cheating detection data from a production AI interviewing system used at the company. All candidate identifiers and features have been anonymized to protect user privacy and prevent reverse engineering of proprietary detection methods.

2.1.1 Main Dataset Structure

We are provided a training dataset (with labels and optional unlabeled examples), test dataset without labels, and a social graph dataset. Each row in the training data represents a single candidate with the following columns:

- `user_hash` – Anonymized candidate identifier.
- `feature_001` to `feature_018` – Eighteen anonymized behavioral and interview-related features. The specific meanings of these features are hidden to prevent reverse-engineering.
- `high_conf_clean` – Binary flag (1 or NaN) indicating high-confidence "not-cheating" predictions generated by the existing production model. Candidates with `high_conf_clean` = 1 do not have manual review labels and represent a broader sample

of the population.

- `is_cheating` – Target variable:
  - 0 = Not cheating (legitimate candidate)
  - 1 = Cheating (confirmed during manual review)
  - NaN = Unlabeled (only for rows where `high_conf_clean` = 1)

A supplemental `feature_metadata.json` file provides further details regarding feature types, ranges, and missingness statistics (Appendix A).

2.1.2 Social Graph

We are also provided with an edge list representing the complete social network for exploring graph-based detection methods:

- `user_a` – Anonymized ID of the first user in a connection.
- `user_b` – Anonymized ID of the second connected user.

The graph includes all relationship edges in the system. Notably, nodes in the graph do not appear in the main dataset (these are "ghost users").

2.2 Problem Setup

The modeling task is defined as a probabilistic classification problem. For each `user_hash` in the test set, our machine learning model needs to predict the probability (0.0 to 1.0) that the candidate is cheating. Unlike standard classification tasks that prioritize metrics like Accuracy or F1-score, this study utilizes a cost-based metric reflecting real-world operational impacts. This approach penalizes costly errors, such as missing a cheater or wrongly blocking

a legitimate candidate, while rewarding confident, correct decisions. Predicted probabilities are translated into three distinct operational decision regions based on optimal thresholds.

Decision Regions

- **Auto-pass** (Low cheating risk): Candidate passes without review.
- **Manual review** (Medium risk): Candidate is flagged for human review.
- **Auto-block** (High risk): Candidate is removed from consideration without review.

Cost Structure

1. False Negative (cheating passes through): $600
2. False Positive in auto-block region: $300
3. False Positive in manual review region: $150
4. True Positive requiring manual review: $5
5. Correct auto-pass or auto-block: $0

The final score is calculated as the negative of the minimum total cost found across all threshold combinations; therefore, a higher (less negative) score indicates superior model performance.

CHAPTER 3

Technical Background

Modern fraud detection has shifted from the analysis of isolated transactions to the investigation of the relational context in which these fraudulent interactions occur. Properly analyzing the provided dataset, which contains a large-scale social graph with over 1.7 million edges, requires a grounding in graph theory and learning specific approaches to graphical feature engineering.

3.1 Graph Theory Fundamentals

The graph is formally defined as an ordered pair $G = (V, E)$, where V denotes the set of vertices (or nodes) representing individual entities in a graph, and E denotes the set of edges representing the connections between them.

Key Graph Structures

1. **Path**: A simple path is defined as a sequence of vertices where each adjacent pair is connected by an edge.
2. **Cycle**: A cycle is a path that starts and ends at the same vertex.
3. **Connected Graphs**: A graph is connected if there is a path between every pair of vertices.

A key theorem in algebraic graph theory states that the (i, j) entry of A^k (the k-th power

of the adjacency matrix) equals the number of walks of length k from node i to node j. This property allows for the efficient computation of multi-hop relationships, which are vital for uncovering "layering" strategies where fraudsters use intermediate nodes to obscure their connections to known bad actors.

In an undirected graph, edges do not have a direction and simply connect two vertices. This type of graph is used to represent symmetric relationships such as connections in a network where direction is not significant. In an undirected graph, an edge is an unordered pair $\{u, v\}$, implying a bidirectional or mutual relationship. For our analysis, the graph data is provided as an edge list containing pairs of anonymized IDs (`user_a`, `user_b`) representing the entire network. In this specific dataset, we also need to account for the presence of "ghost users" – users that appear in the edge list but are absent from the primary tabular dataset. While we do not have metadata on these ghost users, their presence in the matrix allows the model to capture the "hidden" connectivity links between candidate nodes through common shared resources or social connection, thereby mapping the complete network of potential fraud rings.

Homophily

Graph homophily is the tendency of nodes with similar attributes to connect via edges in a graph. In graph-based fraud detection, homophily refers to fraudulent users connecting with each other and legitimate users with other legitimate users leading to dense subgraphs of activity. If a node is connected to several known fraudulent entities, the probability of that node being fraudulent increases. This is often modeled as a diffusion process, where known labels act as "taints" that spread through the graph. In graph-based fraud detection, this principle is used to identify suspicious accounts, though fraudsters can actively camouflage themselves by connecting to normal users (heterophily) and making detection more challenging. Therefore, our models must be robust and highly accurate to account for this.

CHAPTER 4

Exploratory Data Analysis

A comprehensive exploratory data analysis (EDA) was conducted to analyze the training dataset, which encompasses 272,819 unique users. This analysis focuses on target distribution, feature missingness, and the structural properties of the social graph.

4.1 Target Distribution and Class Imbalance

To begin with, the target variable `is_cheating` was visualized to determine if there were any class imbalances in the training data. The target variable contains three possible states: 0 (legitimate), 1 (confirmed cheating), and NaN (unlabeled). Note that the target variable is unlabeled (has an NA value) whenever the current production model predicts that the person has a high likelihood of cheating (i.e. `high_conf_clean` $= 1$). As illustrated in Figure 4.1, the dataset exhibits a severe class imbalance. Only 12.6% of the total observations are confirmed cheating cases. Within the manually reviewed subset (41.4% of the total data), 28.8% are confirmed clean observations, while the remaining 12.6% are confirmed cheating. The largest portion of the dataset, 58.6%, consists of unlabeled "weakly clean" samples.

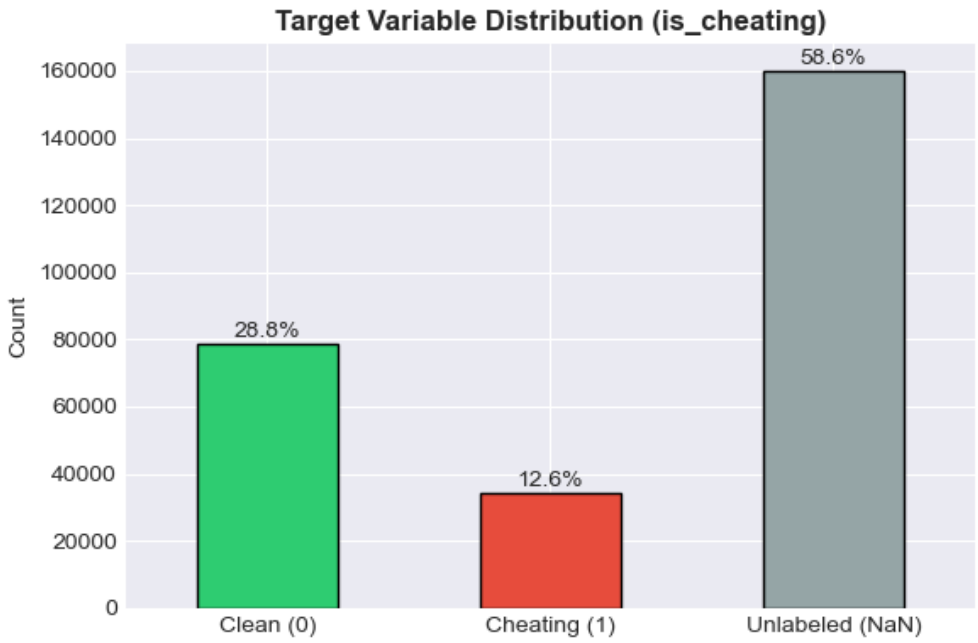

Figure 4.1: Percentage breakdown of the `is_cheating` variable.

Next, a subset of the data was taken to examine the distribution of all the labeled cases (112,966 rows), excluding the unlabeled samples. Figure 4.2 illustrates the class proportions within this labeled subset. Even within this filtered group, imbalance persists: only 30.5% of labeled cases are confirmed cheaters, while 69.5% are legitimate candidates. The abundance of `high_conf_clean` samples ($n = 159{,}853$) suggests that a semi-supervised learning approach may be highly beneficial for model training.

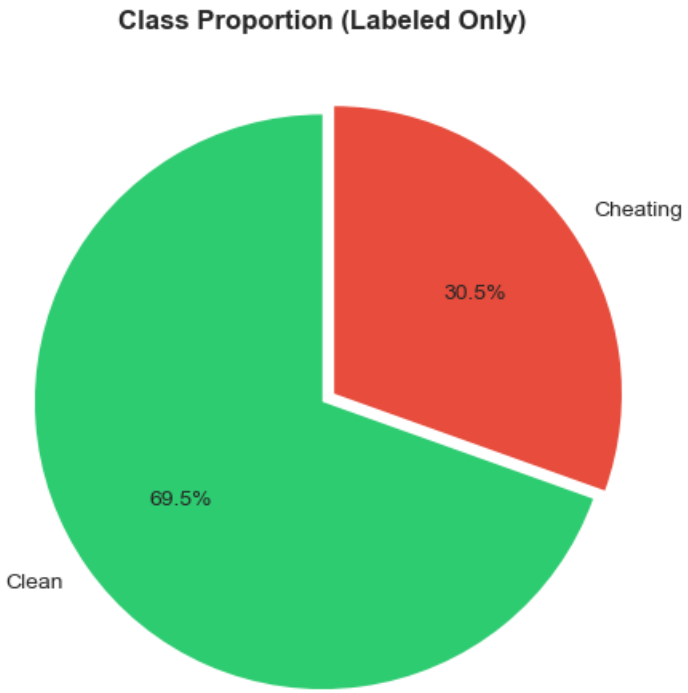

Figure 4.2: Imbalanced class distribution within the labeled subset.

Finally, an initial correlation analysis indicated a lack of significant multi-collinearity within the predictor set. Given that the underlying meaning of the features is hidden due to anonymization, creating interaction terms without domain-informed hypotheses risks overfitting the model to noise. Therefore, the decision was made to treat the feature categories as independent inputs, allowing XGBoost and GNN architectures to identify non-linear relationships autonomously.

4.2 Feature Missingness Analysis

The eighteen anonymized features were analyzed to plot the missing data percentage per feature within the labeled subset. As shown in Figure 4.3, which shows a bar chart of the missing data percentage by feature, five features exhibit missing data rates exceeding 10%,

with `feature_018` missing approximately 15% of its observations.

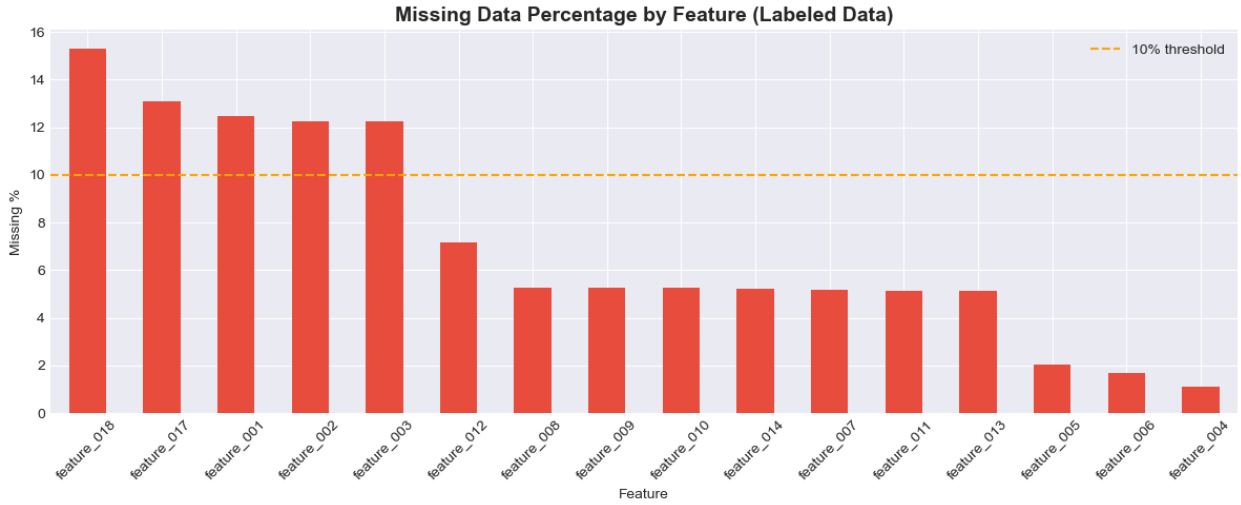

Figure 4.3: Missingness data percentage by feature.

An investigation into the relationship between missingness and cheating behavior reveals that NaN values are highly informative. For each feature, we compute the cheating rate, which is defined as the proportion of labeled users with `is_cheating` $= 1$, separately for observations where the feature is NULL versus where it is present. The results reveal that missingness itself carries strong predictive signals. For several attributes, including `feature_011`, `feature_013`, `feature_007`, and `feature_014`, the cheating rate approaches 98% when the data is null, compared to a baseline of approximately 30% when the data is present. As shown in Figure 4.4, this suggests that missing values are not random but rather systematically associated with cheating behavior, motivating the inclusion of missingness indicators as engineered features in our models.

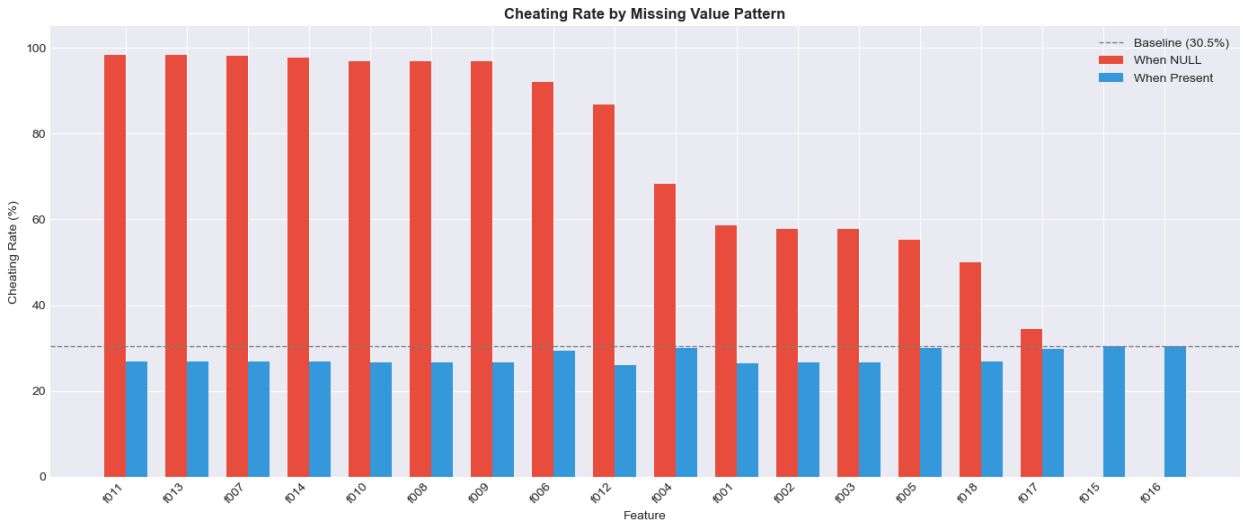

Figure 4.4: Cheating rate by missing value pattern.

4.3 Social Graph Analysis

The provided social graph was also analyzed to understand the characteristics of the candidate pool. Table 4.1 below shows summary statistics for the social graph.

Table 4.1: Social graph summary metrics.

Metric	**Value**
Total nodes	1,727,366
Total edges	1,709,794
Graph density	0.000001
Average degree	1.98
Median degree	1
Connected components	17,624
Largest component size	1,335,920 (77.3%)
Train users in graph	197,920 (72.5%)
Test users in graph	35,437 (73.2%)

The train/test user coverage percentages show that approximately 27% of users do not have a presence in the given social graph (i.e., ghost users), which may require additional feature engineering of graphical features. The social network is also extremely sparse, with a density of 0.000001 and an average degree of just 1.98 connections per user. The median degree of 1 indicates that at least half of all users have only a single connection. This sparsity makes sense given that this is production data from an internal AI interviewing platform at Mercor where users are not actively "friending" each other like other social networks such as social media platforms.

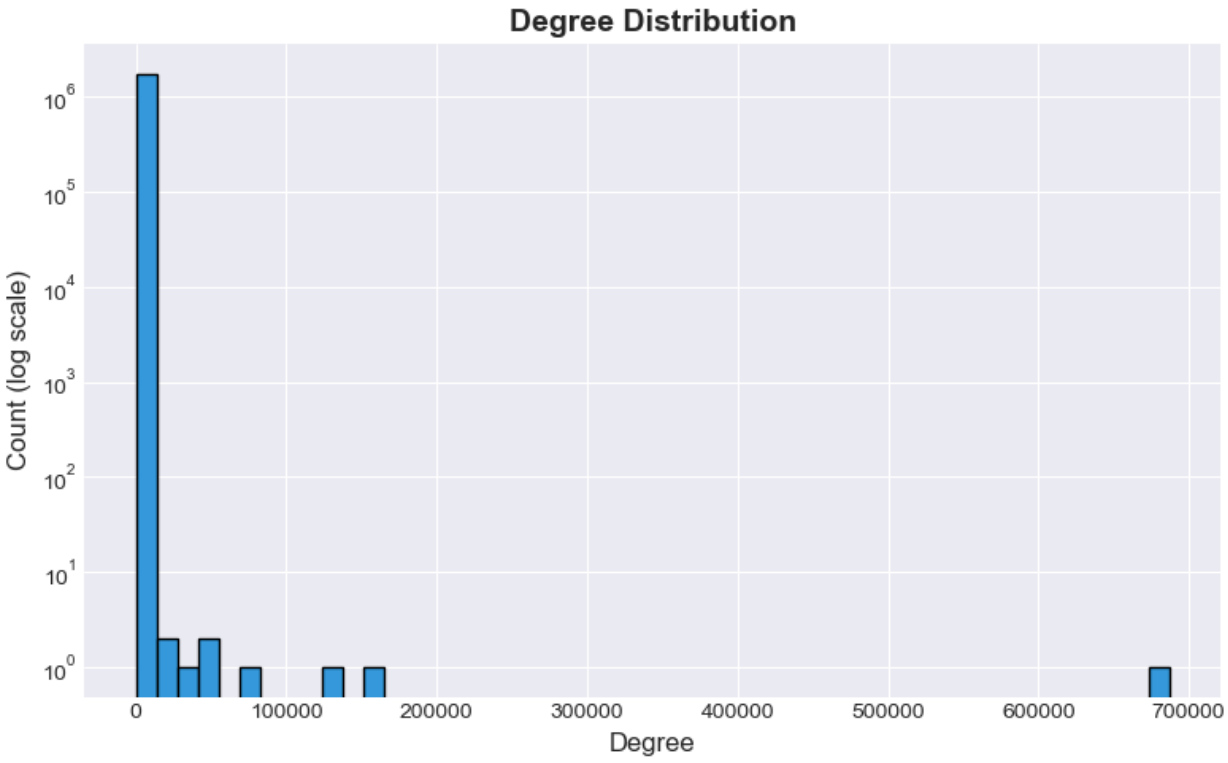

Figure 4.5: Degree distribution of the social network.

The degree distribution exhibits a classic power-law pattern characteristic of real-world social networks. The vast majority of nodes (∼1.7M) have very few connections, while a small number of hub nodes have degrees exceeding 100,000—with at least one node approaching 700,000 connections. This node is an extreme outlier anomaly and likely represents a system artifact rather than an organic hub.

To test the principle of homophily, the average number of "cheater neighbors" was calculated for both cheating and non-cheating candidates. The analysis reveals a key finding: cheaters have an average of 0.18 cheater neighbors compared to just 0.03 for non-cheaters. Despite the sparse network, this supports the idea that cheating behavior clusters socially. Users connected to known cheaters are substantially more likely to be cheaters themselves, which provides a strong basis for graph-based feature engineering.

Lastly, we also visualized and plotted a 2-hop ego network for a randomly selected cheater node. Figure 4.6 below shows the 2-hop social network of a known cheater. Gray nodes represent unlabeled users whose proximity to cheaters could inform semi-supervised labeling or risk scoring. From the plot, we can see in multiple areas that the red labels (i.e. cheaters) are in close proximity to one another and clustered together. This visualization provides further support for our finding that cheaters are more likely to socially cluster together compared to legitimate candidates in the network.

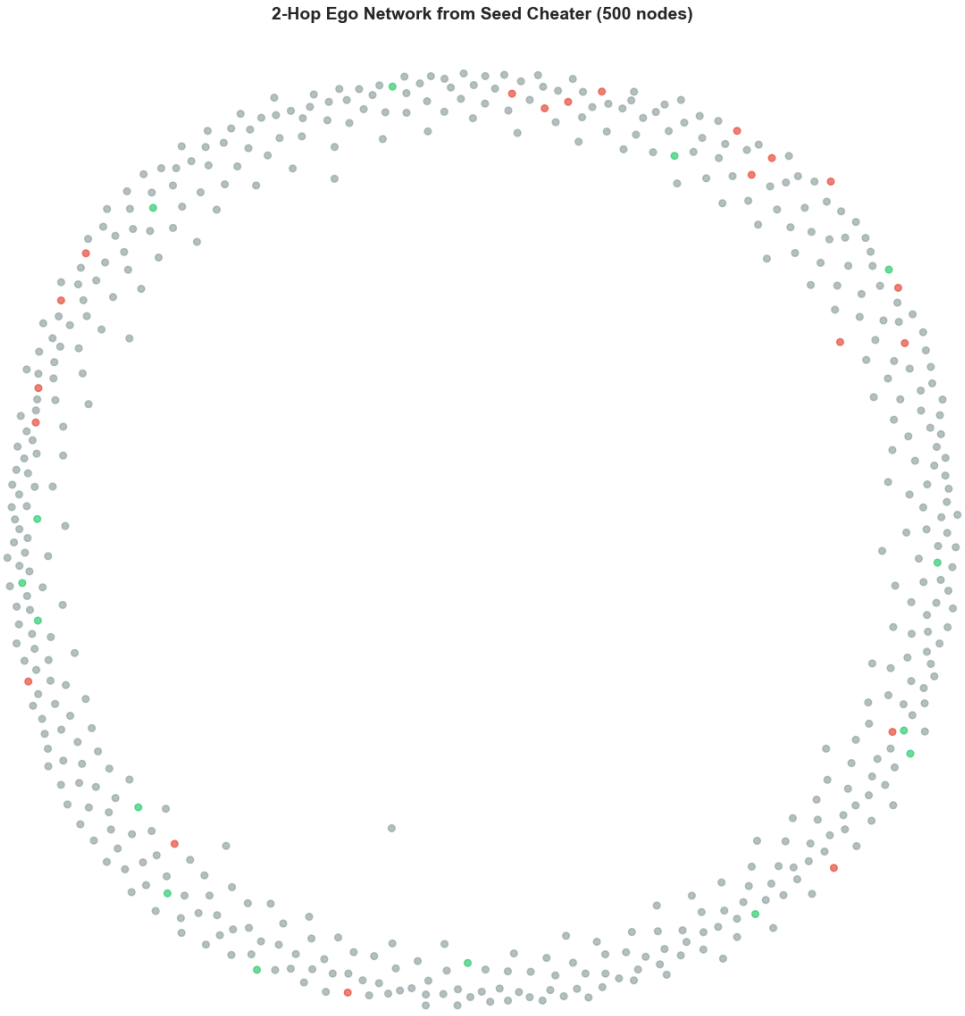

Figure 4.6: 2-Hop ego network visualization from seed cheater.

CHAPTER 5

Feature Engineering

Guided by the graph theory fundamentals previously discussed, the raw dataset was transformed into a comprehensive feature set through extensive feature engineering. First, we generated binary missingness indicators for each of the eighteen behavioral features. As the dataset contains realistic missingness patterns where certain signals are only available for specific candidate subsets, these indicators serve to inform the model of potential systematic biases in data collection.

In addition to these indicators, we derived eight graph-based features to extract structural information from the provided social network. These features are designed to capture the relational context of each candidate, as the connectivity and behavior of a node's neighbors may provide critical signals for identifying cheating behavior.

The following graphical features were integrated into the model:

- **Node degree** – The node degree is the most intuitive measure of a node's activity and centrality. It represents the total number of edges connected to a specific candidate, representing their overall connectivity within the network.

- **PageRank** – PageRank is a measure of a node's global importance based on the principle that links from important nodes carry more weight. It provides a centrality score that measures the relative importance or influence of a node based on the quantity and quality of its connections. A node with a high PageRank that is also connected to known cheaters is highly suspicious, as it suggests a primary hub of illicit activity.

- **Triangles** – The count of three-node subgraphs in which the candidate participates, further characterizing local network stability. We computed triangle counts using NetworkX, which efficiently enumerates triangles for each node by iterating through neighbor pairs. A high triangle count indicates that a node is embedded in a dense social fabric. Fraudulent activity often creates "triangle-heavy" structures because coordinated groups maintain redundant connections to ensure the resilience of their communication or resource-sharing network.

- **graph_cheater_neighbors** – This feature provides the raw count of neighbors already labeled as fraudulent:

$$\texttt{graph\_cheater\_neighbors}_i = \left|\{j \in N_i : \text{Label}_j = \text{“Fraud”}\}\right| \tag{5.1}$$

  A high count of fraudulent neighbors is one of the strongest indicators of risk, as it suggests the node is actively interacting with the known fraudulent population.

- **graph_clean_neighbors** – Conversely, this feature counts the absolute number of direct neighbors who are legitimate (have a confirmed `is_cheating` label = 0).

- **graph_unknown_neighbors** – This counts the number of direct neighbors with no available labels. In large-scale real-world graphs, the majority of nodes are typically unlabeled:

$$\texttt{graph\_unknown\_neighbors}_i = \left|\{j \in N_i : \text{Label}_j = \text{“Unknown”}\}\right|. \tag{5.2}$$

  A high count of unknown neighbors relative to confirmed labels indicates that the node exists in a "grey area" of the network, which may require further investigation or the use of semi-supervised learning methods to propagate labels.

- **graph_cheater_neighbor_ratio** – The proportion of a node's neighbors who are confirmed cheaters relative to their total number of neighbors.

- **graph_in_network** – This binary or categorical feature indicates whether the `user_id` from the main dataset is present within the `social_graph.csv`.

5.1 Feature Summary

The final feature set utilized in this study comprises a total of 44 predictors. This ensemble includes the 18 original anonymized behavioral and interview-related features, 18 corresponding binary missingness indicators to account for realistic data patterns, and 8 derived graphical features designed to capture structural network information.

CHAPTER 6

Methodology

This section presents three modeling approaches of increasing complexity to address the cheating detection problem. We begin with a gradient boosting baseline, extend to a semi-supervised framework that accounts for label uncertainty, and conclude with a graph neural network that learns directly from network structure.

6.1 Model Background

6.1.1 Gradient Boosting with XGBoost

XGBoost (Extreme Gradient Boosting) is an ensemble method that sequentially fits decision trees to the residuals of prior iterations, optimizing a regularized objective function. For a dataset containing n samples and m features, the predicted output at iteration t is defined as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{6.1}$$

where f_t represents the incremental tree added at iteration t. The objective function to be minimized combines a differentiable loss function L with a regularization term Ω:

$$\mathcal{O}^{(t)} = \sum_{i=1}^{n} L\big(y_i,\ \hat{y}_i^{(t-1)} + f_t(x_i)\big) + \Omega(f_t) \tag{6.2}$$

The regularization term

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \tag{6.3}$$

penalizes model complexity by considering the number of leaves T and the magnitude of the leaf weights w.

XGBoost was selected for our baseline for three reasons. First, the algorithm handles missing values natively by learning optimal split directions for NaN values during training. This capability is critical for the Mercor dataset, as our analysis indicates missingness rates as high as 11.9% (e.g., `feature_001`), and preliminary analysis suggests that these missing patterns serve as a strong predictive signal. Second, gradient boosting is widely recognized for achieving state-of-the-art performance on tabular datasets characterized by handcrafted features. Third, the `scale_pos_weight` hyperparameter provides an efficient mechanism to address the significant class imbalance—noted in the training data where the labeled cheating rate is approximately 30.48%—without the need for external resampling techniques.

6.1.2 Semi-Supervised Learning with Sample Weighting

Semi-supervised learning leverages both labeled and unlabeled data to enhance model performance, which is particularly beneficial when labeled instances are scarce or costly to acquire. In the provided dataset, approximately 58.6% of training samples (159,853 records) lack manual review labels. These samples are instead assigned "weak labels" based on the `high_conf_clean` flag, which represents high-confidence "not cheating" predictions from Mercor's production model to detect cheating.

Rather than treating these weak labels as absolute ground truth, this study employs a sample weighting approach to assign differential importance to observations during the training process. The weighted loss function is defined as:

$$\mathcal{L}_{\text{weighted}} = \sum_{i \in \mathcal{D}_L} w_L \cdot \ell(y_i, \hat{y}_i) + \sum_{j \in \mathcal{D}_W} w_W \cdot \ell(y_j, \hat{y}_j) \tag{6.4}$$

where $\mathcal{D}_L$ represents the set of manually reviewed samples with a fixed weight $w_L = 1.0$, and $\mathcal{D}_W$ represents the weak-labeled samples with a weight $w_W < 1.0$. By downweighting the weak labels, the model is able to learn from the broader population distribution while

maintaining its primary focus on the confirmed `is_cheating` labels.

This approach was chosen over alternatives such as pseudo-labeling or self-training because it requires no iterative retraining and integrates directly with XGBoost's existing sample weight functionality. The optimal weight w_W is determined empirically through sensitivity analysis on the validation set.

6.1.3 Graph Neural Networks (GNN)

Graph Neural Networks (GNNs) extend deep learning to non-Euclidean domains by iteratively aggregating information from neighboring nodes. Given a social graph $G = (V, E)$ with node features $X \in \mathbb{R}^{|V| \times d}$, a GNN layer updates node representations through message passing:

$$h_v^{(k)} = \sigma\big(W^{(k)} \cdot \text{AGGREGATE}\big(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\big)\big) \tag{6.5}$$

where $h_v^{(k)}$ is the embedding of node v at layer k, $\mathcal{N}(v)$ denotes the set of neighbors for user v in the social graph, $W^{(k)}$ is a learnable weight matrix, and σ represents a nonlinearity.

In this study, we specifically employ the GraphSAGE (Sample and Aggregate) architecture. The update rule for GraphSAGE is defined as:

$$h_v^{(k)} = \sigma\big(W^{(k)} \cdot \text{CONCAT}\big(h_v^{(k-1)},\ \text{MEAN}_{u \in \mathcal{N}(v)} h_u^{(k-1)}\big)\big) \tag{6.6}$$

GraphSAGE was selected over alternative architectures such as Graph Convolutional Networks (GCN) or Graph Attention Networks (GAT) for two primary reasons. First, GraphSAGE is inductive, meaning it can generate embeddings for "ghost users" or new candidates not seen during training by aggregating features from their local neighborhood. This is essential for a scalable production system where the social graph is constantly evolving. Second, the mean aggregator is computationally efficient and robust to the high-variance degree distributions typically found in the power-law structure of social networks. However, it is important to note that the mean aggregator assumes a degree of homophily, i.e., the tendency for similar nodes to connect. In heterophilic settings, where cheating candidates

may attempt to camouflage their behavior by intentionally connecting to legitimate users, mean aggregation may inadvertently hurt model performance by blending signals.

The central hypothesis for including a GNN in this comparison is that end-to-end learning from the graph structure may capture complex relational patterns that handcrafted features like `graph_degree` or `graph_cheater_neighbor_ratio` might fail to express. By comparing this approach to the XGBoost variants, this study aims to evaluate whether the automatic aggregation of neighborhood information provides a superior predictive signal compared to the handcrafted graph features utilized in the baseline. This comparison directly addresses the broader debate in recent fraud detection literature regarding the trade-offs between the complexity of GNNs and the efficiency of gradient boosting on tabular datasets.

6.2 Experimental Setup

The full training dataset comprises 272,819 candidates, of which 112,966 (41.4%) have confirmed labels from manual review and 159,853 (58.6%) carry weak labels derived from the `high_conf_clean` flag. We construct the target variable by assigning `is_cheating` = 0 to all weak-labeled samples, acknowledging this assumption as a limitation of our analysis.

We employ a stratified 80/20 train/validation split on the full dataset, preserving the class distribution across partitions. This yields approximately 218,000 training samples and 54,000 validation samples. Stratification ensures both partitions maintain the same cheating prevalence, preventing sampling bias from affecting model comparison.

The three models differ in which portion of the training data they utilize. The XGBoost model trains exclusively on labeled samples within the training partition, approximately 90,000 samples. This represents the conservative approach of relying only on confirmed ground truth. The semi-supervised approach and GNN model train on the full training partition, incorporating both labeled and weak-labeled samples. We employ a single stratified split and bootstrap confidence intervals (1,000 resamples) on the validation set to provide

variance estimates for model comparison.

Critically, all models are evaluated on the labeled portion of the validation set only. This ensures performance metrics reflect predictions against confirmed ground truth rather than assumed labels. The labeled validation set contains approximately 22,500 samples, providing sufficient statistical power for model comparison while maintaining evaluation integrity.

6.2.1 Model Configuration

XGBoost Baseline. The baseline model uses XGBoost with up to 500 trees, early stopping after 50 rounds without validation AUC improvement, maximum depth of 6, learning rate of 0.05, and 80% row and column subsampling. Missing values are handled natively through learned split directions.

Semi-Supervised XGBoost. The semi-supervised model uses an identical architecture to the baseline but trains on the full dataset with differentiated sample weights: labeled samples receive weight 1.0, while weak-labeled samples receive a reduced weight w_W. We conduct sensitivity analysis over candidate weights $\{0.1, 0.3, 0.5, 0.7, 1.0\}$, selecting the weight that minimizes cost on the labeled validation set.

GraphSAGE. The GNN employs a two-layer GraphSAGE architecture with mean aggregation, projecting the 44 input features to a 64-dimensional hidden representation with ReLU activation and 30% dropout. Training uses Adam optimizer (learning rate 0.01, weight decay 5×10^{-4}) with early stopping after 20 epochs without improvement, up to 200 maximum epochs.

6.2.2 Class Imbalance Handling

The dataset exhibits significant class imbalance, though the degree varies by training subset. Among labeled samples, approximately 30% are cheaters, yielding a 2.3:1 ratio of clean to

cheating candidates. When weak-labeled samples are included, the cheating rate drops to approximately 12.6%, producing a 7.0:1 imbalance—the assumed-clean weak labels dilute the positive class prevalence.

To address this imbalance, we apply the `scale_pos_weight` parameter in XGBoost, which upweights the positive class during loss computation. The baseline model uses a weight of 2.3 reflecting the labeled-only class ratio, while the semi-supervised model uses 7.0 corresponding to the full dataset distribution. For GraphSAGE, we apply equivalent class weighting through the `pos_weight` parameter in the binary cross-entropy loss function.

This approach ensures each model appropriately accounts for the class distribution in its respective training data, preventing the majority class from dominating gradient updates.

6.3 Evaluation Framework

6.3.1 Standard Classification Metrics

To provide a comprehensive view of model performance and establish a baseline for comparison, several standard statistical metrics are reported. These metrics allow for a diagnostic assessment of model behavior across different dimensions. Notably, simple classification accuracy is omitted from this setup, as it is largely uninformative in the context of the severe class imbalance present in this dataset. A model predicting all candidates as "clean" would achieve high accuracy while failing entirely to detect cheating behavior.

Precision measures the proportion of candidates flagged as cheaters who were confirmed to be engaging in cheating behavior during manual review:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.7}$$

Recall (sensitivity) captures the proportion of actual cheaters who were correctly identified by the model:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.8}$$

The **F1 Score** provides the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.9}$$

AUC-ROC measures the model's discriminative ability to separate classes across all possible classification thresholds. In contrast, AUC-PR (Area Under the Precision-Recall Curve) is often more informative for the imbalanced distributions found in this dataset, as it focuses exclusively on the performance of the positive class (cheating). Finally, the **Brier Score** assesses probability calibration by measuring the mean squared error between predicted probabilities ($\hat{p}_i$) and actual outcomes (y_i):

$$\text{Brier} = \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_i - y_i)^2 \tag{6.10}$$

While these standard metrics provide essential diagnostics, they assume a symmetric cost for all classification errors. In a real-world interview environment, missing a confirmed instance of cheating (a False Negative) carries significantly different operational consequences than incorrectly blocking a legitimate candidate (a False Positive). For a company like Mercor who wants to onboard specialized candidates to train foundational models, onboarding fraudulent candidates can result in declining data quality provided to their clients. Therefore, this high operational cost motivates the use of a specialized cost-based evaluation framework defined below.

6.3.2 Official Competition Metric Analysis

The competition employs a cost-based metric that reflects operational realities. The implementation details of this cumulative cost optimization function can be found in Appendix B. As discussed in the problem setup, the system must assign each candidate to one of three decision regions based on predicted cheating probability.

The cost structure reveals key asymmetries. Specifically, missing a cheater ($600) is twice as costly as wrongly blocking a legitimate user ($300), reflecting the reputational and

operational damage of allowing fraudulent candidates through. Manual review incurs a baseline cost ($150 for clean users, $5 for cheaters) representing human labor.

The official competition metric identifies optimal thresholds t_1 (the boundary between auto-pass and manual review) and t_2 (the boundary between manual review and auto-block) to minimize total operational cost. Rather than grid search, it employs an efficient cumulative sum approach. After sorting the predicted probabilities in ascending order, two cumulative cost vectors, c_1 and c_2, are computed based on the true labels (y_j):

$$c_1[i] = \sum_{j=1}^{i} (745 \cdot y_j - 150) \tag{6.11}$$

$$c_2[i] = \sum_{j=1}^{i} (155 \cdot y_j - 150) \tag{6.12}$$

The optimal thresholds correspond to the indices minimizing c_1 and c_2 respectively. The total cost is subsequently calculated as:

$$\text{Cost} = \min(c_1) + \min(c_2) + 300 \cdot N_{\text{clean}} \tag{6.13}$$

where N_{clean} is the total number of clean users. The metric returns the negative of the total cost so that higher values indicate superior model performance.

6.3.3 Threshold Analysis: Break-Even Points

The cost structure implies specific probability thresholds at which the optimal decision changes. These break-even points can be derived analytically.

Auto-Pass vs. Manual Review. A candidate should be automatically passed rather than sent for manual review when the expected cost of an auto-pass is lower than the expected

cost of review. Let P denote the model's predicted probability of cheating:

$$\begin{aligned} \text{Cost(Pass)} &< \text{Cost(Review)} \\ P \times 600 + (1-P) \times 0 &< P \times 5 + (1-P) \times 150 \\ 600P &< 5P + 150 - 150P \\ 745P &< 150 \\ P &< \frac{150}{745} \approx 0.2013 \end{aligned} \tag{6.14}$$

Therefore, candidates with predicted cheating probability below 20.1% should be auto-passed.

Manual Review vs. Auto-Block. Similarly, auto-blocking is preferred over manual review when:

$$\begin{aligned} \text{Cost(Block)} &< \text{Cost(Review)} \\ (1-P) \times 300 + P \times 0 &< P \times 5 + (1-P) \times 150 \\ 300 - 300P &< 5P + 150 - 150P \\ 150 &< 155P \\ P &> \frac{150}{155} \approx 0.9677 \end{aligned} \tag{6.15}$$

This reveals a critical insight: auto-blocking is only cost-effective when the model is at least 96.77% confident a candidate is cheating. Given the inherent noise in behavioral interview data, achieving this level of precision represents a significant modeling challenge.

CHAPTER 7

Model Results and Discussion

This section presents the performance results for each of the three modeling approaches, followed by model comparison and feature importance analysis.

7.1 Experimental Results

XGBoost

The XGBoost model was trained on 90,411 labeled samples and was evaluated on 22,555 labeled validation samples. The model achieved strong performance with an AUC-ROC of 0.854 and AUC-PR of 0.785. At the optimal threshold, precision reached 0.646—meaning approximately 65% of candidates flagged as cheaters were confirmed cheaters—while recall of 0.719 indicates the model captured ∼72% of actual cheaters. The Brier score of 0.144 suggests reasonably well-calibrated probability estimates.

Using the competition's cost-based evaluation metric, the baseline achieved a score of −1,462,105 (95% CI: [−1,498,030, −1,418,860]), the best among all three approaches.

Table 7.1: XGBoost model results.

Metric	**Value**
Cost Score	-1,462,105
AUC-ROC	0.8539
AUC-PR	0.7852
F1	0.6804
Precision	0.6459
Recall	0.7187
Brier	0.1439

Figure 7.1 presents the top 20 features ranked by importance (gain-based). Features are categorized as behavioral (original features), missingness (indicator variables), or graph (engineered network features). From the plot, we can see that only 2 of the engineered graphical features were in the top 10—`graph_cheater_neighbor_ratio` and `graph_unknown_neighbors`—with 2 other graph features `graph_degree` and `graph_pagerank` in the top 20. Notably, `feature_012` and its corresponding missingness indicator both rank highly, suggesting this feature carries strong predictive signal whether present or absent.

The relatively modest contribution of graph features likely reflects the sparsity of the social network. With an average degree of approximately 2 and a hub-dominated structure, most node-level metrics collapse to near-zero for the majority of users. Additionally, neighbor-based features such as `graph_cheater_neighbors` are limited by the prevalence of "ghost nodes" which carry no label information to propagate.

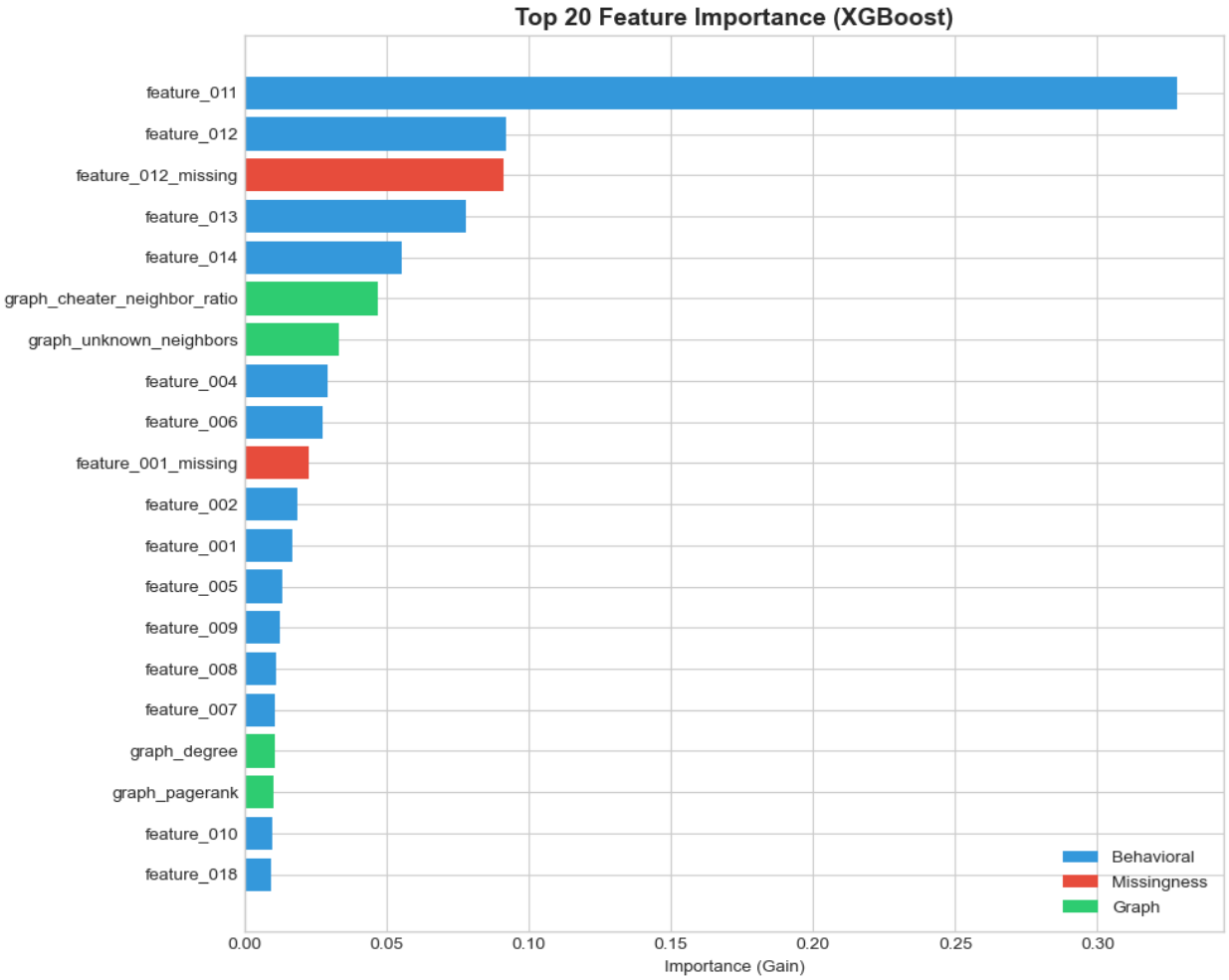

Figure 7.1: XGBoost feature importance.

Semi-Supervised XGBoost

The semi-supervised model was trained on the full dataset of 218,255 samples, applying differentiated sample weights to account for label uncertainty. We conducted sensitivity analysis across five candidate weights for weak-labeled samples, evaluating each configuration on the labeled validation set.

Table 7.2: Weight sensitivity analysis results.

Weight	**Cost Score**	**AUC-ROC**	**AUC-PR**
0.1	−1,488,275	0.8517	0.7832
0.3	−1,496,480	0.8496	0.7817
0.5	−1,508,115	0.8484	0.7807
0.7	−1,498,275	0.8478	0.7805
1.0	−1,505,010	0.8470	0.7806

Figure 7.2 shows the cost score plotted against the label weight for our semi-supervised XGBoost model. The model achieves the best cost score performance at the label weight of 0.1, illustrated by the cost score of −1,488,275. Performance degrades monotonically as weight increases, with AUC-ROC declining from 0.8517 at weight 0.1 to 0.847 at weight 1.0. This pattern suggests the weak labels introduce noise rather than useful signal and heavily discounting them yields the best results.

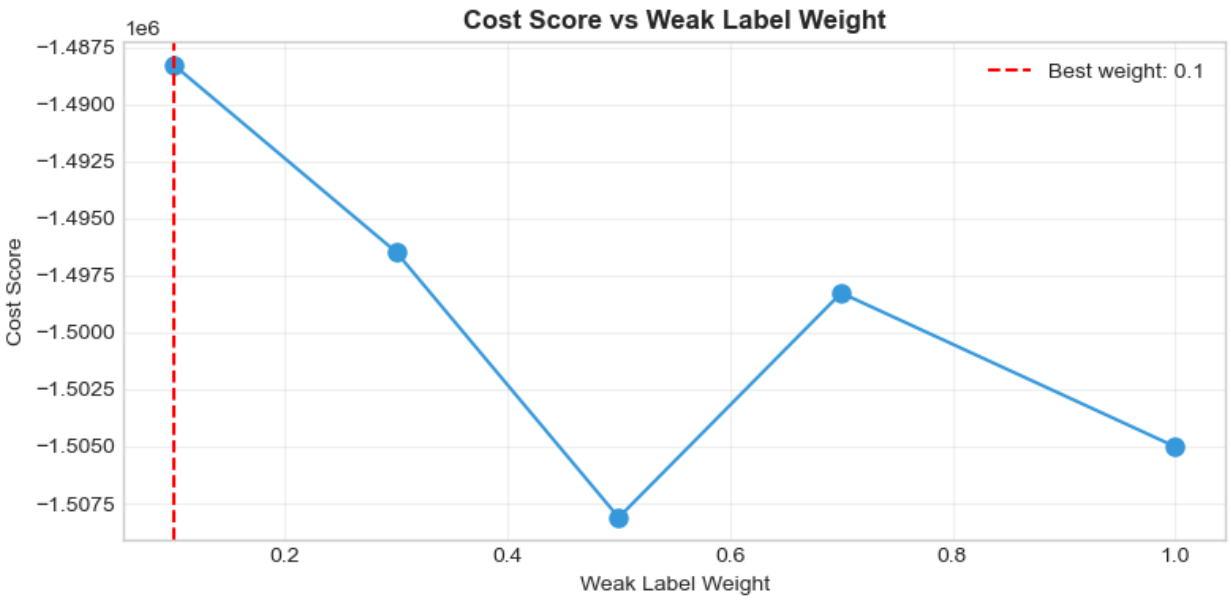

Figure 7.2: Cost score vs. label weight.

Notably, the model exhibits a different precision–recall trade-off than the baseline: precision dropped to 0.468 while recall increased to 0.889. This shift toward higher recall but lower precision indicates the model became more aggressive in flagging candidates as cheaters, likely influenced by the expanded training set. The elevated Brier score (0.220 vs. 0.144) indicates poorer probability calibration, suggesting the weak labels degraded the reliability of predicted probabilities.

GraphSAGE

The graph neural network was trained on 218,255 nodes with only 24,235 edges connecting users within the training set—a substantial reduction from the original 1.7 million edges, as only connections between training users are included. Training proceeded for 117 epochs before early stopping, reaching a peak validation AUC of 0.739.

Final evaluation revealed significantly weaker performance than the tabular approaches,

with AUC-ROC of 0.720 and AUC-PR of 0.594. The cost score of −2,081,030 (95% CI: [−2,108,478, −2,042,325]) represents an increase in cost of approximately $619,000 compared to the baseline, which is a statistically significant difference given non-overlapping confidence intervals. The Brier score of 0.235 indicates poor probability calibration.

The underperformance likely stems from graph sparsity. With only 24,235 edges among 218,255 nodes, the average degree drops below 1, severely limiting message passing.

7.2 Model Comparison

Table 7.3 summarizes performance across all three approaches. The XGBoost baseline achieves the best cost score (−1,462,105), followed by the semi-supervised model (−1,488,275) and GraphSAGE (−2,081,030). The two XGBoost variants perform similarly on discrimination metrics (AUC-ROC of 0.854 vs. 0.852), while GraphSAGE lags substantially at 0.720.

Table 7.3: Model performance comparison.

Model	**Cost Score**	**AUC-ROC**	**Precision**	**Recall**	**F1**	**Brier**
XGBoost	−1,462,105	0.8539	0.6459	0.7187	0.6804	0.1439
Semi-Supervised	−1,488,275	0.8517	0.4677	0.8892	0.6130	0.2195
GraphSAGE	−2,081,030	0.7200	0.4327	0.6474	0.5187	0.2351

Figures 7.3 and 7.4 present the ROC and precision–recall curves respectively. The baseline and semi-supervised models produce nearly overlapping curves, confirming their comparable discriminative ability. GraphSAGE falls well below both, consistent with its weaker AUC scores.

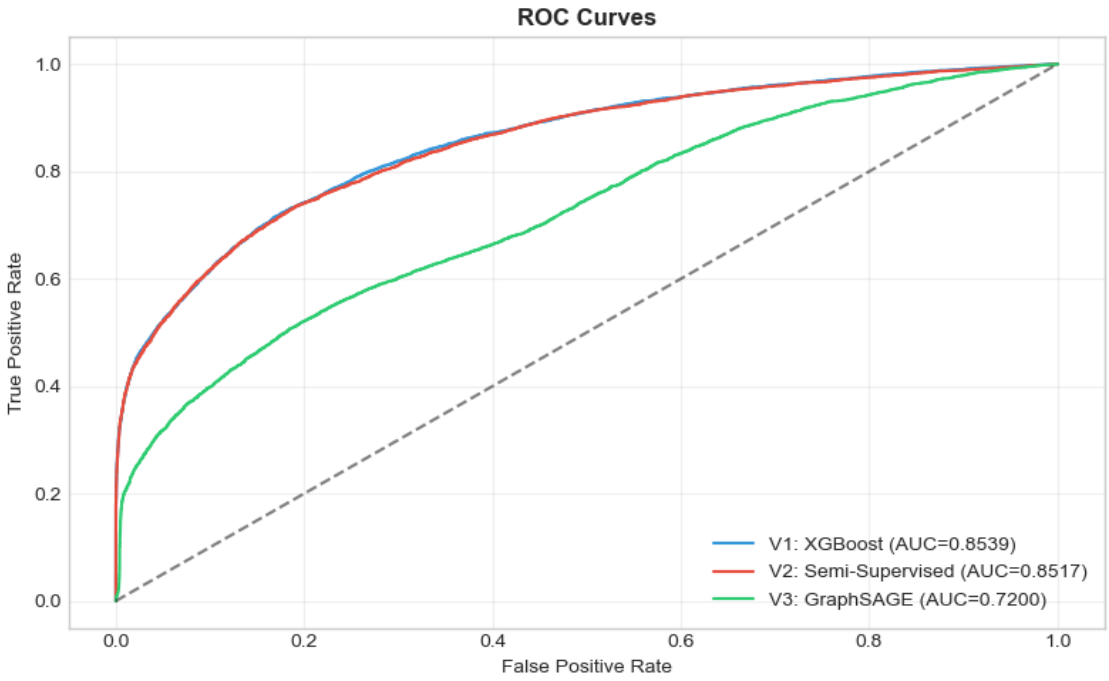

Figure 7.3: ROC curve comparison.

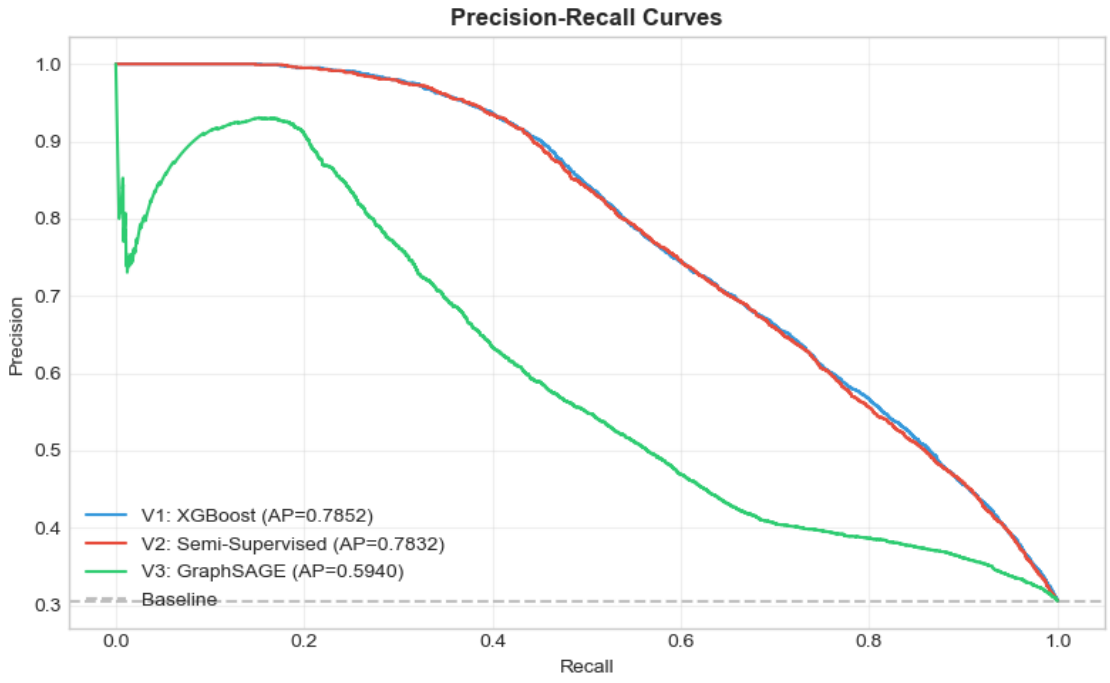

Figure 7.4: Precision–recall curve comparison.

Decision Region Analysis

The optimal thresholds and resulting decision region distributions reveal distinct operational profiles across models. Both XGBoost variants set similar review thresholds (0.346 and 0.592) and block thresholds near the theoretical break-even point (0.975 and 0.990). GraphSAGE, however, sets its block threshold at 1.0—meaning it never auto-blocks candidates, as the model cannot achieve sufficient confidence to justify the risk.

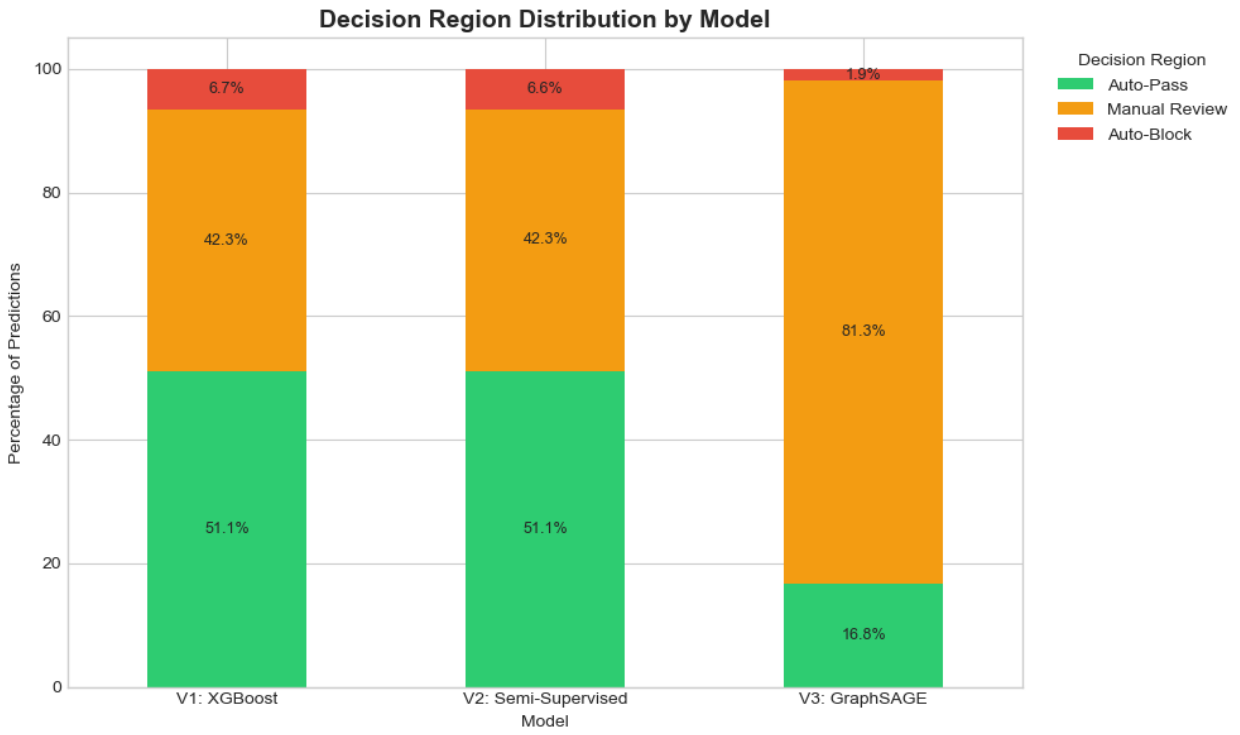

Figure 7.5: Decision region distribution breakdown by model.

The distribution of candidates across decision regions reflects these thresholds. The baseline and semi-supervised models route approximately 51% of candidates to auto-pass, 42% to manual review, and 7% to auto-block. GraphSAGE produces a starkly different distribution: only 17% auto-pass, 81% manual review, and 2% auto-block.

This conservative posture carries significant operational implications. Routing 81% of candidates to manual review would overwhelm human reviewers and largely defeats the purpose of automated screening. In contrast, the XGBoost models achieve a more balanced workload, where they auto-passed half of the candidates while reserving manual review for genuinely uncertain cases. The 7% auto-block rate, while small, represents high-confidence predictions that can be actioned without human intervention, further reducing operational burden.

CHAPTER 8

Limitations

While the proposed methods and models have demonstrated promising results in predicting whether candidates are engaging in cheating behavior, some limitations regarding the data structure and experimental design must be acknowledged.

Feature Anonymization and Interpretability. A primary constraint of this study is the anonymized nature of the dataset. To protect proprietary detection methods, Mercor provided behavioral and interview-related features without semantic descriptions. This lack of domain transparency precludes the application of domain-informed feature engineering and restricts the interpretability of model outputs. For instance, while feature importance analysis identifies `feature_011` as the most influential predictor, it is impossible to provide a substantive explanation for this relationship. This limits the actionable insights that can be derived for production environments.

Weak Label Assumptions. The most significant limitation concerns our treatment of weak-labeled samples. We assigned `is_cheating` = 0 to all 159,853 candidates flagged as `high_conf_clean`, representing 58% of the training data. While these labels derive from a production model's high-confidence predictions, they remain unverified assumptions. The semi-supervised model's optimal weight of 0.1—effectively discounting weak labels to near-zero—suggests these assumed labels may introduce more noise than signal. An alternative approach would be to train exclusively on labeled data, as our baseline results demonstrate this yields superior performance.

Selection Bias in Evaluation. Our evaluation is restricted to manually reviewed candidates, which constitute a non-random subset of the population. These candidates were flagged for review precisely because they exhibited suspicious behavior, introducing selection bias. Model performance on the broader population, including candidates who would never trigger manual review, remains unknown. This limits the generalizability of our findings to actual production deployment scenarios.

Probability Calibration. The elevated Brier scores for the semi-supervised model and GraphSAGE relative to the baseline indicate poor probability calibration. We did not apply calibration techniques in our modeling pipeline, which represents a missed opportunity. Isotonic regression, a non-parametric calibration method, could adjust predicted probabilities to better reflect true class frequencies while preserving prediction rankings.

CHAPTER 9

Conclusion

This thesis investigated machine learning approaches for detecting cheating in online interviews, addressing several challenges that make this problem unique: a cost-based evaluation metric reflecting real operational impacts, semi-supervised learning with labeled cheating cases and high-confidence unlabeled examples, graph-based detection opportunities using social network structure, and class imbalance and sampling bias mirroring real-life conditions.

We evaluated three modeling strategies: an XGBoost baseline trained on labeled data, a semi-supervised extension with weak labels, and a GraphSAGE model learning from graph structure. The XGBoost baseline achieved the best cost score (−1,462,105), demonstrating that gradient boosting with handcrafted features remains highly competitive for tabular fraud detection. Incorporating weak labels through sample weighting did not improve performance – the optimal weight of 0.1 effectively discarded most weak-labeled samples, suggesting label noise outweighed any benefit. GraphSAGE substantially underperformed (−2,081,030), limited by sparse connectivity that prevented effective message passing.

Feature importance analysis revealed behavioral features dominate predictive signal, with `feature_011` contributing 33% of total importance. Missingness indicators proved informative, confirming that null values correlate with cheating behavior. Graph features contributed modestly, constrained by network sparsity and unlabeled neighbors. Threshold analysis demonstrated that auto-blocking is only cost-effective when confidence exceeds 96.77% – a bar models rarely clear. Both XGBoost variants routed approximately half of candidates

to auto-pass and $\sim$42% to manual review, balancing automation against the risk of missed cheaters.

These findings suggest that interpretable models with engineered features can match or exceed complex approaches, label quality matters more than quantity, and cost asymmetries must guide threshold selection in operational deployment.

Appendix A

Feature Metadata

The following feature metadata was provided for the eighteen anonymized features.

Table A.1: Feature metadata summary.

Feature	**Type**	**Range**	**Missing %**
feature_001	numeric	1–5	11.9
feature_002	numeric	1–10	11.7
feature_003	numeric	1–10	11.7
feature_004	numeric	0–10	0.9
feature_005	numeric	1–10	2.1
feature_006	numeric	0–10	0.9
feature_007	binary	0–1	2.6
feature_008	numeric	0–10	2.7
feature_009	numeric	0–10	2.7
feature_010	numeric	0–39,128,914	2.7
feature_011	binary	0–1	2.6
feature_012	numeric	0–3	4.6
feature_013	binary	0–1	2.6
feature_014	binary	0–1	2.6
feature_015	numeric	−7–898	0.0
feature_016	numeric	0–581	0.0
feature_017	numeric	0–24	10.9
feature_018	numeric	0–1	10.9

Appendix B

Evaluation Metric Code

The following Python code implements the custom cost-based evaluation metric used in the Mercor Kaggle competition.

```python
import numpy as np
import pandas as pd
from pandas.api.types import is_numeric_dtype

def score(solution: pd.DataFrame, submission: pd.DataFrame,
          row_id_column_name: str) -> float:
    """
    Calculate cost-based score for cheating detection.

    Finds optimal decision thresholds that divide predictions
    into three regions:
    - Auto-pass (low confidence cheating):
        $0 if correct, $600 if missed cheating
    - Manual review (medium confidence cheating):
        $5 if cheating, $150 if wasted on legitimate user
    - Auto-block (high confidence cheating):
        $0 if correct, $300 if wrongly blocked legitimate user
    """
    merged = solution.merge(
        submission, on=row_id_column_name, how='inner')
```

```
21
22      if merged.empty:
23          raise ValueError(
24              "No matching IDs between solution and submission")
25
26      if 'prediction' not in merged.columns:
27          raise ValueError(
28              "Submission must have 'prediction' column")
29
30      target_candidates = [
31          c for c in merged.columns
32          if c not in [row_id_column_name, 'prediction', 'Usage']]
33
34      if len(target_candidates) != 1:
35          raise ValueError(
36              f"Cannot identify target column. "
37              f"Found: {target_candidates}")
38
39      target_col = target_candidates[0]
40
41      y_true = merged[target_col].values
42      y_pred = merged['prediction'].values
43
44      if not is_numeric_dtype(merged['prediction']):
45          raise ValueError("Predictions must be numeric")
46      if not np.all((y_pred >= 0) & (y_pred <= 1)):
47          raise ValueError(
48              "Predictions must be between 0 and 1")
49
```

```
    y_true = np.array(y_true, copy=False)
    sort_order = np.argsort(y_pred)
    y_true = y_true[sort_order]

    c1 = (y_true * 745 - 150).cumsum()
    c2 = (y_true * 155 - 150).cumsum()

    total_cost = (c1.min() + c2.min()
                  + (y_true == 0).sum() * 300)

    return float(-total_cost)
```

REFERENCES

[An26] Jiaying An. "How AI Is Forcing Companies To Rethink Coding Interviews." https://www.forbes.com/councils/forbestechcouncil/2026/01/02/how-ai-is-forcing-companies-to-rethink-coding-interviews/, 2026. Last accessed 2026/01/06.

[CG16] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[Dat] DataCamp. "An Introduction to Graph Theory." https://www.datacamp.com/tutorial/introduction-to-graph-theory. Last accessed 2026/02/23.

[Gra] "Graph Homophily and Network Learning." https://www.emergentmind.com/topics/graph-homophily. Last accessed 2026/02/24.

[Jil] O. L. Jilbert. "2.10 Ego-Centric Networks — Social Networks: An Introduction." Online textbook.

[Mer26] Mercor. "Mercor Cheating Detection." https://kaggle.com/mercor-cheating-detection, 2026. Last accessed 2026/02/28.

[MMi] MMiDS Textbook. "5.2. Background: basic concepts in graph theory — MMiDS Textbook." https://mmids-textbook.github.io/chap05_specgraph/02_graph/roch-mmids-specgraph-graph.html. Last accessed 2026/02/23.

[MSC01] Miller McPherson, Lynn Smith-Lovin, and James M. Cook. "Birds of a Feather: Homophily in Social Networks." *Annual Review of Sociology*, **27**:415–444, 2001.

[POK20] Tahereh Pourhabibi, Kok-Leong Ong, Booi H. Kam, and Yee Ling Boo. "Fraud detection: A systematic literature review of graph-based anomaly detection approaches." *Decision Support Systems*, **133**:113303, 2020.

[Sel] Robert Sellmair. "Does Semi-Supervised Learning Help to Train Better Models?" https://towardsdatascience.com/does-semi-supervised-learning-help-to-train-better-models-338283d1f4e9/. Last accessed 2026/02/28.

[WPC21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. "A Comprehensive Survey on Graph Neural Networks." *IEEE Transactions on Neural Networks and Learning Systems*, **32**(1):4–24, 2021.

[Yip25] Jonathan Yip. "AI startup Mercor now valued at $10 billion with new $350 million funding round." https://www.cnbc.com/2025/10/27/ai-hiring-startup-mercor-funding.html, 2025. Last accessed 2026/01/06.

[ZYZ20] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. "Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs." In *Advances in Neural Information Processing Systems*, pp. 7793–7804. Curran Associates, Inc., 2020.