A Hybrid GNN-LLM Fusion Architecture for Multi-Modal Fraud Detection in Ethereum Smart Contracts

I. Introduction

The rapid growth of blockchain technology, particularly Ethereum, has revolutionized digital finance through decentralized applications and smart contracts. However, this wave of innovation has also created new avenues for sophisticated financial fraud. The consequences are significant: more than $1 billion was lost to cryptocurrency scams between 2021 and 2022 alone [1]. These illicit activities take several forms, including phishing attacks—where fraudulent websites steal users' private keys; Ponzi schemes, which use funds from new investors to pay earlier participants; smart contract exploits, where malicious code is embedded to siphon funds or manipulate protocols; and rug pulls, in which developers abruptly abandon projects and disappear with investors' money [2–4].

Traditional methods like secure key management [5, 6], multi-signature wallets [7, 8], and node authentication [9, 10] help reduce the risk of theft or unauthorized access by preventing compromised keys from being used to move funds to malicious accounts. However, multi-signature wallets come with some drawbacks. They introduce greater complexity and can cause delays, as transactions require approval from multiple parties. If any signer is unavailable or uncooperative, the process can be significantly slowed. Additionally, managing multiple keys increases operational overhead and raises the risk of errors or accidental loss of access. Several researchers have explored the use of supervised learning techniques, such as support vector machines and random forest classifiers, for blockchain security [11–14]. By training these models on labeled datasets, they can effectively identify and classify specific malicious patterns in blockchain transactions or network activity, enhancing the precision of threat detection. However, the effectiveness of these methods depends heavily on having a comprehensive dataset of known attack patterns for training. If the dataset is incomplete or lacks examples of emerging threats, the model's overall accuracy and ability to detect new types of attacks may be limited. To capture complex relationships and patterns in blockchain networks, some researchers proposed graph-based model [15–17]. The representation of transactions, wallets, nodes, and interactions as graphs helps us detecting anomalies and finding suspicious clusters. However, analyzing large-scale blockchain transaction graphs requires significant computational resources and time, especially for real-time applications.

To address some of the limitations in existing blockchain security models, we propose a hybrid architecture that integrates advanced Graph Neural Networks (GNNs) with Large Language Models (LLMs)—powerful AI systems capable of understanding and generating both human and computer language. Our approach begins by using GNNs to construct transaction graphs that analyze complex relationships among contract addresses and cryptographic wallets. Based on this analysis, we extract relevant blockchain contracts and decompile their bytecodes into source code. These smart contract source codes are then used as training samples for our LLM, enabling deep semantic analysis. By combining the strengths of GNNs for network analysis and LLMs for code comprehension, our system achieves a comprehensive understanding of blockchain activities, significantly enhancing the detection of sophisticated fraud patterns.

Our research represents a significant advancement by systematically combining these powerful yet distinct analytical paradigms on real-world Ethereum data. This pioneering fusion approach enables our system to achieve a more holistic understanding of blockchain activities, allowing it to detect a wider spectrum of fraudulent activities by leveraging both the behavioral and semantic dimensions of blockchain data.

Contributions:

- We conduct detailed network analysis using Graph Neural Networks (GNNs) to trace the flow of transactions, enabling the identification of suspicious relationship patterns and anomalous movement of funds.
- We introduce a novel method for analyzing decompiled smart contracts with Large Language Models (LLMs), uncovering various interactions within the contracts and identifying potentially malicious code segments.
- We design a robust and practical data pipeline that effectively addresses the challenges of collecting and integrating data from multiple sources. Our solution is highly adaptable and can be seamlessly integrated into existing blockchain networks, maintaining compatibility even in dynamic or elastic network environments.

The rest of this section is organized as follows: Section II outlines the motivation behind our work, while Section III provides background information and reviews recent related research. Section IV details our proposed system architecture and methodology. In Section V, we discuss key challenges and present corresponding countermeasures. Section VII covers our

prototype design and experimental results. Finally, Section **??** summarizes our findings and concludes the paper.

II. Motivation

The explosive growth of blockchain technology—particularly in the realms of decentralized finance (DeFi), cryptocurrency exchanges, and smart contract platforms—has brought about unprecedented opportunities, but also significant risks. As the ecosystem expands, so do the avenues for sophisticated financial fraud, including rug pulls, phishing, smart contract exploits, and money laundering. These threats not only undermine user trust but also result in substantial financial losses, hampering the broader adoption and innovation of blockchain solutions.

Our motivation stems from the urgent need for robust, scalable, and intelligent tools that can proactively detect and prevent fraudulent activities across the blockchain landscape. The proposed hybrid fraud detection system leverages the strengths of both Graph Neural Networks (GNNs) and Large Language Models (LLMs) to address the multifaceted nature of on-chain fraud. By combining structural analysis of transaction networks with deep semantic understanding of smart contract code, our architecture is uniquely positioned to uncover complex fraud patterns that traditional, single-modality approaches often miss.

The practical applications of such a system are far-reaching. Cryptocurrency exchanges can integrate real-time fraud screening to protect users from malicious token listings; DeFi protocols can assess risks associated with new contracts or third-party integrations; regulators and law enforcement agencies can automate the monitoring of suspicious blockchain activity, enhancing anti-money laundering efforts; and individual users can benefit from real-time alerts and risk assessments, safeguarding their assets from phishing and other scams.

Beyond its immediate utility, the widespread deployment of advanced fraud detection technologies carries profound societal benefits. By reducing the frequency and severity of financial crimes, our system helps protect individuals and institutions, empowers users with objective risk assessments, and fosters greater stability and trust in volatile crypto markets. Moreover, enhanced security and oversight encourage institutional adoption and mainstream acceptance of blockchain solutions, supporting further innovation and the safe evolution of the Web3 ecosystem.

Nevertheless, our approach recognizes key limitations and ethical considerations. The system currently relies on the availability of public, verified smart contract source code, limiting its coverage of certain fraud vectors. Its effectiveness hinges on continuous adaptation to the ever-evolving tactics of fraudsters, and there is always a risk of false positives. Ethically, it is paramount to balance transparency and user privacy, guard against model bias, and ensure that AI-generated predictions are used to support—not replace—human decision-making and accountability.

In summary, our motivation is rooted in the real-world need for intelligent, adaptive, and ethical fraud detection solutions that not only secure blockchain ecosystems, but also promote broader adoption and innovation in digital finance.

III. Background and Related Work

The pursuit of blockchain fraud detection has evolved through several distinct intellectual phases, beginning with early heuristics and rule-based classifiers before advancing toward machine learning, graph learning, and multimodal fusion approaches. Our work is situated within this latest phase, where deep learning dominates but is increasingly challenged by issues of scalability, interpretability, and generalization. To contextualize our contribution, we critically survey prior research by categorizing the landscape into three principal streams: graph-centric analysis, code and sequence-based analysis, and the emerging class of hybrid methodologies.

A. The Graph-Centric Paradigm: Uncovering Behavioral Conspiracies

This paradigm is motivated by the hypothesis that illicit activities manifest as detectable anomalies in the blockchain's structural and temporal patterns. By modeling blockchains as transaction graphs, researchers uncover relational structures that are invisible at the level of individual transactions.

Weber et al. [18] provided a foundational proof-of-concept by applying Graph Convolutional Networks (GCNs) to Bitcoin money laundering detection. Their use of the Elliptic dataset demonstrated that topological features of transaction flows could effectively identify laundering rings, establishing the transaction graph as a viable substrate for financial forensics. Huang et al. [19] extended this approach to the scale of Ethereum by proposing PEAE-GNN, which constructs augmentation ego-graphs around known labeled accounts. This selective graph construction addresses scalability and makes GNN-based detection more practical in real-time settings. Chen et al. [20] confronted the pervasive issue of class imbalance, where fraudulent accounts are dwarfed by benign ones. Their hybrid GNN model (DA-HGNN) integrated data augmentation with temporal and structural learning, proving that balancing datasets is crucial for building effective fraud detection systems. Li, Jia, and Su [21, 22] introduced a Transformer-enhanced Graph Attention Network with global-local attention, allowing their model to capture both neighborhood and global context. This architecture enabled the detection of distributed money laundering schemes, surpassing the limitations of local GCNs.

Ouyang et al. [23] shifted focus toward collective behavior, introducing Bit-CHetG, a subgraph-based supervised contrastive learning algorithm that identifies coordinated laundering groups. This work shows that fraud often emerges from collusion patterns rather than isolated anomalies. Kanezashi et al. [24] compared heterogeneous and homogeneous GNNs for Ethereum phishing detection, concluding that heterogeneous models, which explicitly encode different node and edge types, consistently outperform simpler ones. This demonstrates the value of incorporating semantic variety into graph structures. Li et al. [25] further emphasized temporal aspects with their Temporal Transaction Aggregation Graph Network (TTAGN), which encodes transaction sequences into edge representations. Their results showed that when a transaction occurs is as

important as who is involved, underscoring the value of dynamic graph modeling.

B. The Code-Centric Paradigm: Exposing Semantic Malice

Parallel to graph analysis, another line of work treats smart contracts as executable programs and focuses on vulnerabilities and malicious intent encoded in contract logic. Luu et al. [26] pioneered this direction with Oyente, a symbolic execution tool that revealed vulnerabilities such as reentrancy and transaction-ordering dependence, famously identifying flaws in TheDAO contract. Tsankov et al. [27] extended this approach with Securify, which uses a domain-specific language to check both compliance and violation patterns, thereby offering formal guarantees of contract safety. Qian et al. [28] demonstrated that vulnerability detection can be learned directly from low-level opcodes. Their BLSTM-ATT model automatically detected reentrancy vulnerabilities, eliminating the need for manual feature engineering. Torres, Steichen, and State [29] specialized in honeypot scams with HONEYBADGER, a symbolic execution tool that systematically identified hidden traps embedded in contracts. Zhang et al. [22] advanced forensic analysis with TXSPECTOR, which replays transaction traces to uncover exploits already performed on-chain, making it particularly valuable for zero-day post-mortem detection. Sun et al. [30] introduced GPTScan, which hybridizes large language models with static analysis. GPT guides the analysis engine to detect logic vulnerabilities with fewer false positives, pioneering the integration of AI assistants with program verification. Kong et al. [31] developed UEChecker, which applies graph learning to code-level call graphs, detecting unchecked external calls by analyzing risky function interaction patterns. This represents a convergence between code-based and graph-based approaches.

C. The Nascent Hybrid Paradigm: Towards a Holistic Synthesis

The most recent research recognizes that neither graph structure nor contract code alone provides a complete picture of fraudulent activity. Hybrid approaches integrate multiple modalities — transactions, code, and temporal signals — to achieve holistic detection. Hu et al. [32] introduced BERT4ETH, which applies Transformer pre-training to Ethereum transaction histories, treating transaction sequences as sentences. The model successfully captures sequential semantics, improving fraud detection accuracy. Liang et al. [33] proposed PonziGuard, which constructs Contract Runtime Behavior Graphs (CRBGs) that combine contract logic with fund flow dynamics, yielding high accuracy in Ponzi detection. Sun et al. [34] developed TLMG4Eth, one of the first multimodal fusion models that jointly learns from transaction semantics and graph representations. Their results empirically demonstrated that fusion significantly outperforms isolated models. Wu et al. [35] built TokenScout, a temporal attributed multigraph model for early scam token detection. By integrating graph evolution with token attributes, it detects rug pulls and honeypots in their formative stages. Galletta and Pinelli [36] emphasized explainability with their machine learning classifier for Ponzi schemes. Using XAI techniques, they showed that interpretable features derived from both transaction and code domains can make fraud detection models more transparent and trustworthy. Camino et al. [37] provided an alternative perspective with a feature-engineering approach. By training classifiers on features extracted from transaction behaviors, they showed that on-chain footprints alone could provide strong predictive signals, even without analyzing bytecode.

This critical survey highlights several key limitations in current approaches: Graph-centric methods are effective at detecting relational anomalies but often overlook semantic and contract-level vulnerabilities. In contrast, code-centric methods provide deep insights into vulnerabilities within contracts yet fail to account for the relational and temporal contexts in which these contracts operate. While hybrid methods offer promise, they are still constrained by scalability challenges, limited temporal modeling, and insufficient integration across modalities. To address these gaps, our work proposes a fusion framework that unifies graph and code representations, capturing both behavioral patterns and underlying intent. By incorporating temporal signals, structural embeddings, and semantic features, our model delivers a comprehensive perspective on fraudulent activity in Ethereum, pushing the boundaries of blockchain security.

IV. System Architecture and Methodologyl

Our conceptual architecture composed of three primary layers as shown in figure 1. Data collection is primarily responsible for collecting data from the Etherscan networks. Feature engineering layer generate graphs and extract the source code. The final layer, neural network uses machine learning models on data from the previous layer to classifies malicious transactions.

A. Data Collection Layer

Our fraud detection system is structured as a comprehensive four-phase pipeline. The process begins with multi-source data collection and integration, gathering raw data from diverse blockchain data providers and unifying it into a coherent format. Next, feature engineering and preprocessing transforms this raw data into meaningful features suitable for machine learning models, including graph-specific features and tokenized code representations. In the third phase, our novel GNN-LLM fusion model is trained on the prepared multi-modal dataset. Finally, model evaluation and validation rigorously assess the trained model's performance using standard metrics to ensure its effectiveness and generalization capabilities.

B. Phase 1: Data Collection and Integration

This initial phase is critical for establishing a high-quality, comprehensive dataset that reflects the multi-modal nature of blockchain fraud.

1) Data Sources: Google's BigQuery hosts a public dataset of Ethereum blockchain data, offering a vast historical record of transactions [cite: 7]. From this resource, we specifically extracted 498,112 transactions, with a key focus on those involving "known addresses," particularly those identified

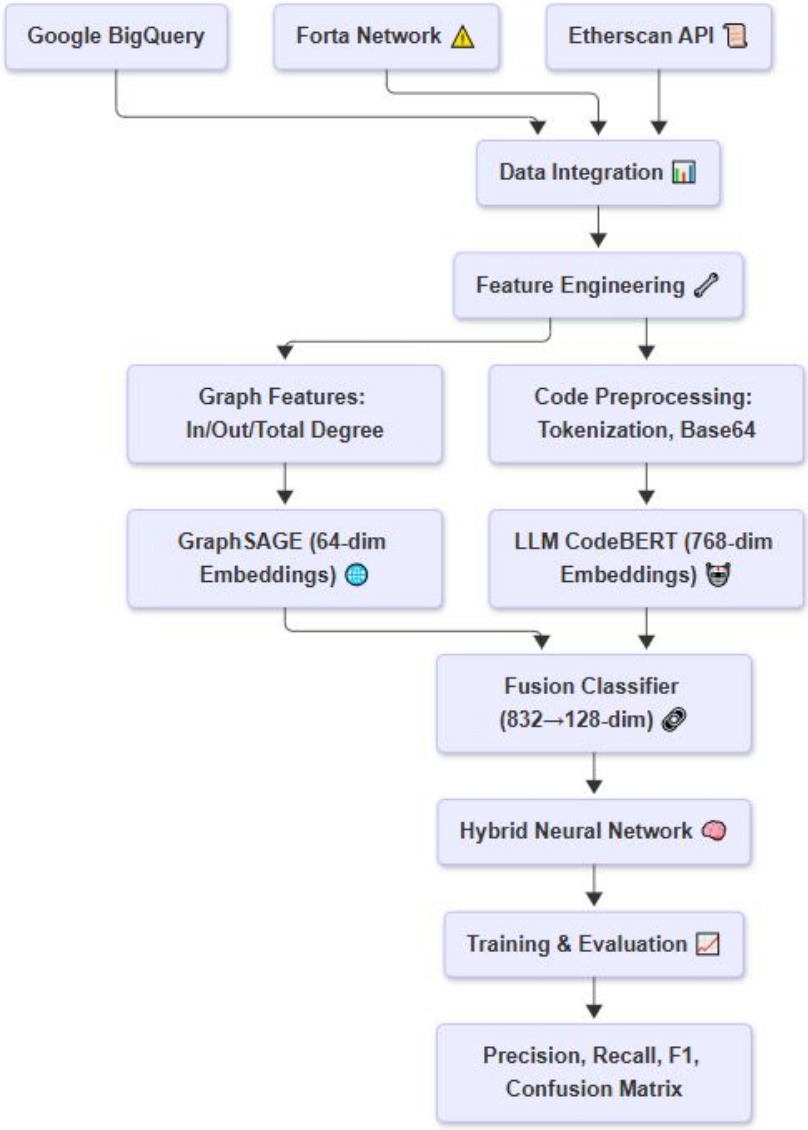

Fig. 1: Hybrid GNN-LLM Architecture for Smart Contract Fraud Detection

.

as fraudulent or interacting with fraudulent entities []. Our data collection strategy followed a ”needle-first” approach, beginning with a curated list of approximately 200 verified fraudulent contracts—referred to as the ”Golden List”—and retrieving their complete transaction history over a one-year period. This method ensures that every transaction involves at least one of our seed contracts, resulting in a dense and highly relevant transaction graph. The resulting dataset forms the essential network structure for our graph analysis, providing critical ”who-transacted-with-whom” information.

The Forta Network is a decentralized community of security bots that continuously monitor blockchain activity in real time, and their public repositories provide curated lists of addresses flagged for suspicious behavior [cite: 2]. From these sources, we extracted ground-truth labels for 2,671 addresses that were publicly identified as fraudulent, covering various fraud categories such as phishing (60%), Ponzi schemes (25%), and other malicious activities (15%) []. This dataset serves as the foundation for our supervised learning approach, providing essential ground-truth labels. To simplify the classification task and ensure sufficient data per class for robust binary classification, all fraud sub-categories from Forta were collapsed into a single ”fraudulent” label (y=1), while all other observed addresses were initially labeled as ”benign” (y=0). This strategy enabled us to focus on the primary detection task with reliable, well-defined labels.

Etherscan.io, the most popular Ethereum blockchain explorer, offers a comprehensive API that provides access to publicly verified smart contract source code [cite: 8]. Using this resource, we successfully retrieved the source code for 1,258 smart

contracts by querying the Etherscan API for unique addresses identified as contracts within our transaction data [cite: 3]. Access to human-readable source code is essential for the semantic analysis performed by our LLM component, as it enables a detailed inspection of each contract's underlying logic.

2) Data Integration Process: Integrating disparate data from three distinct sources presented several challenges, including varied data formats, address casing inconsistencies, and potential missing entries across datasets. We employ DuckDB, an in-memory analytical database, for its efficiency in handling large datasets and performing complex SQL-like joins directly within the Python environment. The key steps for integration included

- **Standardize addresses:** A crucial preprocessing step involved converting all Ethereum addresses (from BigQuery transaction data, Forta labels, and Etherscan source code fetches) to a uniform lowercase format. This meticulous standardization eliminated data mismatches and ensured accurate joins and consistent entity matching.
- **Join data sources:** Using DuckDB, we performed LEFT JOIN operations to enrich our master list of all unique addresses. This involved matching addresses from the transaction data with the Forta fraud labels (to assign the y classification label) and with the fetched Etherscan source code.
- **Create unified tables:** The integrated data was materialized into two primary tables, specifically structured for our graph neural network. The first table is a node table containing all unique Ethereum addresses, enriched with a contract_text field (empty if no verified source code was found) and a y label (0 for benign, 1 for fraudulent). The second table is an edge table derived directly from the transaction data, defining the connections between source (src) and destination (dst) addresses.

C. Phase 2: Feature Engineering and Preprocessing

Once the data was integrated, the next phase focused on transforming the raw information into numerical features suitable for our neural network components.

1) Graph Feature Engineering: For each Ethereum address (node) in our constructed transaction graph, we calculated fundamental network centrality features that provide a concise numerical representation of an address's interaction patterns: in-degree, which counts the number of incoming transactions and reflects how many other addresses have sent funds to or interacted with the address; out-degree, which measures the number of outgoing transactions and shows how many distinct addresses this address has sent funds to or interacted with; and total degree, the sum of in-degree and out-degree, representing the address's overall level of activity or connectivity within the network. These simple yet powerful features are crucial behavioral signals for the GNN. For example, phishing addresses often exhibit a remarkably high in-degree, receiving funds from many victims, but have a comparatively low out-degree, indicating few addresses for fund dispersal. In contrast, legitimate entities such as centralized exchanges typically show more balanced in-degree and out-degree values, reflecting their role as intermediaries. Addresses involved in money laundering services or illicit mixing protocols may display distinctive and complex degree patterns, acting as central hubs with high transaction traffic.

2) Smart Contract Code Preprocessing: Solidity smart contract source code presents unique challenges due to its complexity, significant variation in length and structure, and the presence of special characters that can cause issues when parsing from standard CSV files. To prepare the code for input into the LLM and ensure safe data storage, we first converted the raw Solidity source code into a Base64 encoded string. This transformation turns multi-line, potentially problematic text into a single-line, standardized string that is safe for storage in CSV files and prevents parsing errors during data loading. Next, before inputting the code into CodeBERT, the Base64-decoded source code was tokenized using CodeBERT's pre-trained tokenizer, which breaks the human-readable code into discrete tokens (numerical IDs) that the language model can process and understand. Finally, since transformer models like CodeBERT require fixed-size input sequences, the tokenized code was either truncated if it exceeded 512 tokens or padded with special "padding" tokens if it was shorter, ensuring all sequences met the required length.

D. Phase 3: Hybrid Neural Network Architecture

Our hybrid model is meticulously designed to fuse the distinct information streams derived from the transaction graph (behavioral context) and smart contract source code (semantic content).

1) Graph Neural Network Component: The GNN component of our system utilizes a GraphSAGE (SAmple and AggreGatE) architecture, which is designed to learn robust node representations that capture both structural and behavioral patterns within the Ethereum transaction network [cite: 4]. GraphSAGE is an inductive GNN framework that generates node embeddings by iteratively aggregating information from a node's local neighborhood, rather than relying on a fixed, global graph structure. For each target node, GraphSAGE begins by sampling a fixed number of its immediate neighbors, a strategy that enables scalability to very large graphs. It then aggregates the feature vectors of these sampled neighbors, typically using functions such as mean, sum, or more sophisticated pooling operations. Finally, the aggregated information is combined with the target node's own features and processed through a neural network layer, resulting in a new, enriched embedding for the node that reflects both its individual characteristics and its local network context.

GraphSAGE is particularly well-suited for dynamic and evolving graphs like the Ethereum transaction network due to its inductive capability. Unlike traditional methods, it learns generalizable aggregation functions, allowing it to generate embeddings for new, unseen addresses (nodes) that were not part of the training graph [cite: 4]. This property is crucial for real-time fraud detection on an ever-expanding blockchain.

Moreover, GraphSAGE offers scalability by sampling neighbors instead of processing the entire graph adjacency matrix, which significantly reduces computational complexity and enables efficient analysis of large-scale graphs. In addition, GraphSAGE is highly expressive, effectively capturing complex network patterns and identifying how information, influence, or even fraudulent behavior propagates through the network structure.

Our specific use of GraphSAGE within the GraphEncoder module is a two-layer GraphSAGE network:

- **Layer 1: SAGEConv(3, 128):** This layer takes the 3-dimensional initial node features (in-degree, out-degree, total degree) and outputs 128-dimensional intermediate embeddings.
- **Activation:** A Rectified Linear Unit (ReLU) non-linearity is applied after the first layer (.relu()) to introduce non-linearity and enhance feature transformation.
- **Regularization:** A Dropout(0.5) layer with a 50% dropout rate is applied to the output of the first SAGEConv layer. This regularization technique randomly sets a fraction of input units to zero during training, which helps prevent overfitting by forcing the network to learn more robust features.
- **Layer 2: SAGEConv(128, 64):** This final SAGEConv layer takes the 128-dimensional intermediate embeddings and produces the final 64-dimensional graph embeddings for each node. These embeddings are dense vectors encapsulating the learned behavioral and relational patterns.

2) Large Language Model Component: The LLM component of our system leverages CodeBERT to perform a deep semantic analysis of smart contract source code, enabling the extraction of latent features that capture the code's functionality, intent, and potential vulnerabilities. CodeBERT is a specialized adaptation of the BERT architecture, pre-trained on a massive corpus of programming languages including Python, Java, JavaScript, PHP, Ruby, Go, and patterns relevant to Solidity as well as paired natural language documentation []. This dual-modality pre-training allows CodeBERT to develop a nuanced understanding of both code syntax and semantics, as well as common programming constructs. Thanks to its extensive pre-training, CodeBERT possesses an inherent knowledge of code structure, which enables it to extract meaningful features from source code with minimal task-specific fine-tuning. Unlike traditional unidirectional models, CodeBERT processes code bidirectionally, simultaneously considering context from both directions of each token, thus achieving a more comprehensive understanding of code semantics and inter-token relationships []. Furthermore, regardless of the input source code's length (up to its maximum sequence limit), CodeBERT produces a fixed-size 768-dimensional embedding vector. For our purposes, we use the pooler_output, which provides a high-level summary of the entire code sequence, making it ideally suited for integration into our downstream classification task.

We design a ContractDataset module that is responsible for loading the smart contract source code, performing the necessary tokenization, and ensuring that the sequences are properly truncated or padded to fit CodeBERT's input requirements (e.g., max_length=512).

3) Fusion Architecture: The core innovation of our approach lies in the FusionClassifier, which is meticulously designed to seamlessly combine the structural insights gleaned from the GNN (representing network behavior) with the semantic understanding derived from CodeBERT (representing code logic). This holistic view enables a more robust fraud detection capability. The flow diagram of our fusion process is shown in figure 2.

Our fusion process follows the following steps:

1) **GNN Embedding Generation:** The GraphEncoder (GNN) first processes the entire transaction graph using all nodes and edges (graph_data.x, graph_data.edge_index). This produces a complete set of 64-dimensional graph-based embeddings for every single node (Ethereum address) in the network. These embeddings are pre-calculated for efficiency in the forward pass.
2) **LLM Embedding Generation:** For the subset of nodes that correspond to smart contracts and have publicly verified source code, CodeBERT processes their respective tokenized code inputs (batch['input_ids']) and attention masks (batch['attention_mask']). This process yields a 768-dimensional semantic embedding for each smart contract in the current training batch.
3) **Embedding Retrieval and Alignment:** During the forward pass of the FusionClassifier, for a given batch of training samples (which are drawn from the ContractDataset and thus correspond to smart contract nodes with both graph context and source code), the model efficiently retrieves their pre-computed graph embeddings. This is done by indexing all_graph_embs using the batch['node_idx'].
4) **Concatenation:** The 64-dimensional graph embedding for each contract address in the batch is then directly concatenated (torch.cat([...], dim=-1)) with its corresponding 768-dimensional semantic embedding from CodeBERT. This forms a richer, combined feature vector with a dimensionality of 832 (64 (GNN) + 768 (LLM) = 832).
5) **Fusion Layer Transformation:** This 832-dimensional fused embedding is then passed through a fully connected fusion_layer (nn.Linear). This layer is followed by a ReLU activation function (.relu()) and a Dropout(0.5)layer. This fusion_layer learns to optimally combine and distill the joint information from both modalities into a more compact and discriminative 128-dimensional representation.
6) **Final Classification:** A final classifier (nn.Linear) then takes the 128-dimensional output of the fusion layer and maps it to a 2-dimensional output (logits). These logits represent the model's unnormalized scores for the two target classes: "Benign" and "Fraud".

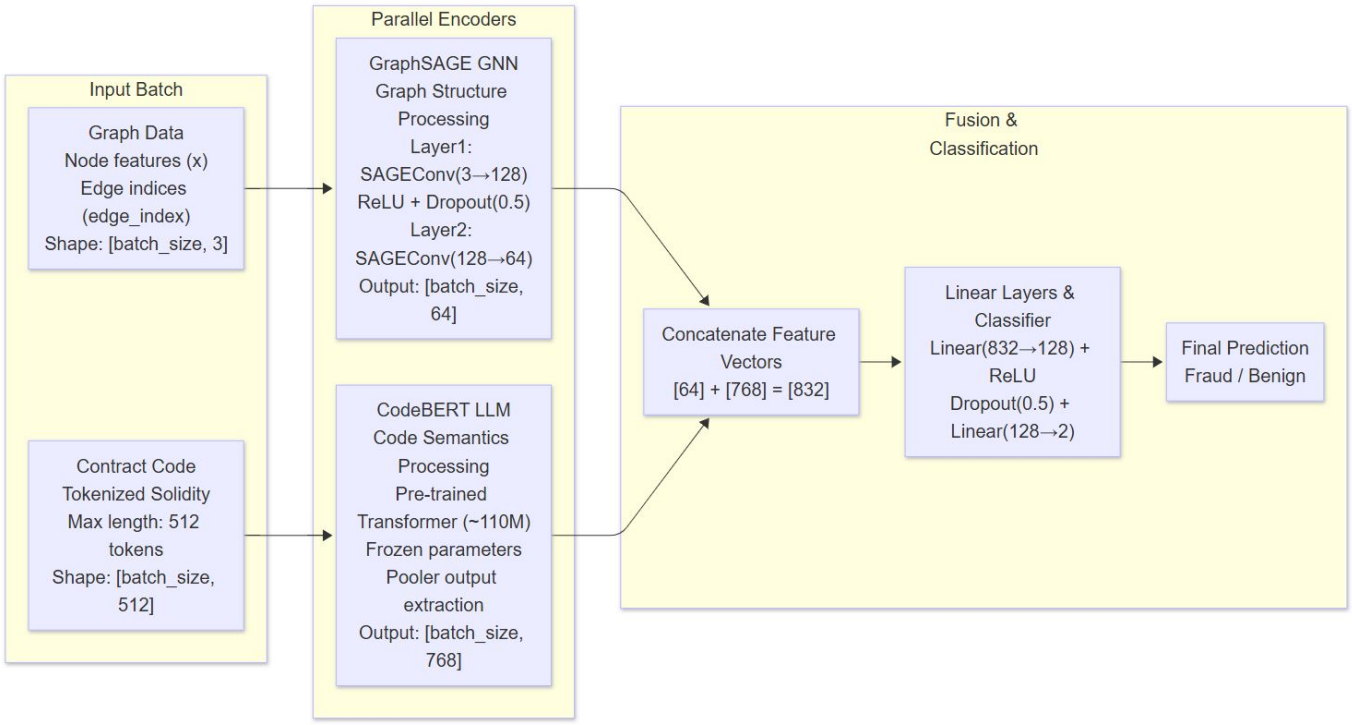

Fig. 2: FusionClassifier Internal Architecture

.

E. Phase 4: Training and Evaluation

1) Training Strategy: A major challenge in real-world fraud detection is the pronounced class imbalance between benign and fraudulent samples. In our dataset, only 14.5% of smart contracts are labeled as fraudulent, which means a model trained naively on this data would likely favor the majority (benign) class and perform poorly at detecting the minority (fraudulent) class. To address this, we employ a Weighted Cross-Entropy Loss function during training. Specifically, the weight for the benign class is set to 1.0, while the weight for the fraudulent class is dynamically determined as the ratio of benign to fraudulent samples in the training data - approximately 5.8 times higher in our case. This weighting strategy ensures that misclassifications of fraudulent contracts are penalized more heavily, compelling the model to pay closer attention to these rare but critical instances.

For model optimization, we employ the AdamW optimizer, which is recognized for its efficiency and robust performance when training deep neural networks, particularly those involving transformer components. We set the learning rate to 5e-5, a standard and empirically effective starting point for fine-tuning transformer-based models. The batch size is fixed at 8, a choice guided by the memory constraints of our available GPU (NVIDIA Tesla T4) and balancing computational efficiency with model stability. Training is conducted over 5 epochs, a value selected empirically to ensure sufficient convergence while reducing the risk of overfitting, given the dataset size and model complexity. Additionally, to prevent the ”exploding gradients” problem that can destabilize training in deep networks, we apply gradient clipping using torch.nn.utils.clip_grad_norm with a maximum norm of 1.0, ensuring gradients remain within a manageable range throughout the learning process.

2) Evaluation Metrics: After training, the model’s performance is rigorously evaluated on a dedicated test set to assess its ability to generalize to unseen data. We use standard classification metrics that offer a comprehensive perspective, which is especially important in the context of imbalanced datasets. Precision measures the percentage of contracts predicted as fraudulent that are actually fraudulent, which is crucial for minimizing false positives and reducing unnecessary alarms. Recall reflects the proportion of actual fraudulent contracts that are correctly identified by the model—a particularly vital metric in fraud detection, as it minimizes false negatives and helps prevent potential financial losses. The F1-score, which is the harmonic mean of precision and recall, provides a balanced single metric that is especially useful when dealing with imbalanced classes. Additionally, a confusion matrix visually summarizes the model’s predictions versus actual classifications, explicitly showing the number of true positives (correctly identified fraudulent contracts), true negatives (correctly identified benign contracts), false positives (benign contracts incorrectly flagged as fraudulent), and false negatives (fraudulent contracts that were missed by the model).

V. Challenges and Countermeasures

A. Data Integration Challenges

1) Address Format Inconsistency: Ethereum addresses collected from different platforms, such as BigQuery [], Forta [], and Etherscan [], often exhibited variations in casing—appearing in uppercase, lowercase, or mixed-case formats. These inconsistencies led to failures during database joins and data matching. To resolve this issue, we implemented a comprehensive address standardization preprocessing step in which all addresses from every source were programmatically converted to a uniform lowercase format. This approach ensured complete data consistency across all datasets, eliminated data loss due to format discrepancies, and enabled accurate matching of entities throughout the integration process.

2) Missing Contract Source Code: A significant challenge arose from the fact that many smart contract addresses—particularly those linked to malicious activities do not have publicly verified source code available on Etherscan []. If not managed properly, this could result in InvalidInputException errors or incomplete data within the pipeline. To address this, the data processing workflow was designed to handle missing source code gracefully. Specifically, when the Etherscan API did not return verified code for an address, an empty string (") was assigned to the relevant contract_textfield for that node. This strategy maintained dataset integrity, prevented script crashes, and allowed for the maximum utilization of available samples by ensuring that nodes without code could still be included in the graph analysis and accurately labeled as benign or fraudulent if Forta data was available.

3) API Rate Limiting: Public APIs such as Etherscan impose rate limits—for example, the free tier allows only five requests per second—to prevent abuse and manage server load. Exceeding these limits can result in HTTP errors and failures when fetching data. To address this, a controlled request mechanism was implemented, introducing a time.sleep(0.25) second pause between successive API calls to Etherscan. Additionally, robust error handling using try-except blocks was incorporated to gracefully catch and log any API errors. This approach enabled the successful retrieval of all available smart contract source code without exceeding rate limits, thereby ensuring the completeness of our code dataset while operating within the constraints of public APIs.

B. Model Architecture Challenges

1) Dimensionality Mismatch: A challenge arose from the fact that the GNN component produces 64-dimensional node embeddings, while CodeBERT generates 768-dimensional code embeddings. Combining these vastly different representation spaces requires careful handling to ensure effective integration. As detailed in Section 3.4.3, we addressed this by using a straightforward concatenation approach, directly combining the 64-dimensional GNN embedding with the 768-dimensional CodeBERT embedding to form an 832-dimensional feature vector. This combined vector is then passed through a fully connected fusion layer (implemented as a nn.Linear layer), which projects it to a lower, unified dimension, 128 in our case—allowing the model to learn the optimal way to merge these features. This method proved effective for integrating information from disparate representation spaces without resorting to overly complex alignment mechanisms, enabling the model to efficiently leverage both structural and semantic insights.

2) Class Imbalance: The dataset presents a significant class imbalance, with only 14.5% of smart contracts labeled as fraudulent. Training a model on such imbalanced data risks biasing predictions in favor of the majority (benign) class, resulting in poor performance on the minority (fraudulent) class. To address this, as described in Section 3.5.1, we implemented a weighted Cross-Entropy Loss function in which the benign class was assigned a weight of 1.0, while the fraudulent class received a substantially higher weight dynamically calculated as the ratio of benign to fraudulent samples—approximately 5.8 times greater. This weighting strategy compelled the model to pay more attention to the rare fraudulent cases during training, ultimately leading to significantly improved recall and F1-scores for the fraud class despite its minority representation.

3) Computational Efficiency: Training deep learning models, especially those that incorporate large transformer models like CodeBERT and process complex graph structures, can be computationally intensive and highly demanding in terms of memory. To address these challenges, several optimization strategies were employed. First, the parameters of the pre-trained CodeBERT model were frozen during the training of the FusionClassifier. This approach prevents memory-intensive backpropagation through the large language model and significantly reduces both training time and GPU memory consumption, while still taking full advantage of CodeBERT's powerful pre-trained features. Additionally, an efficient batch size of eight was selected to allow for effective gradient updates while fitting within the memory limits of the available NVIDIA Tesla T4 GPU. The adoption of PyTorch Geometric further ensured efficient graph operations and effective memory management for the GNN component. Collectively, these optimizations resulted in manageable training times—approximately 45 minutes per epoch, or three to four hours in total for five epochs and stable memory usage, making the pipeline practical and feasible on standard cloud GPU instances.

C. Scalability Considerations

While our current implementation is highly effective for the analyzed dataset, several limitations must be addressed to achieve broader, production-level scalability. At present, the LLM component is restricted to analyzing addresses with publicly verified smart contract source code, which excludes a significant portion of EOA-based fraud and unverified malicious contracts. Additionally, the batch size and graph processing are limited by available GPU memory, meaning that processing extremely large or dense graphs may require more powerful hardware or alternative strategies. The sequential nature of API calls for fetching contract source code, even with the introduction of delays, can also become time-consuming when

dealing with extremely large numbers of addresses. To address these challenges in the future, distributed processing could be implemented through multi-GPU or distributed training setups, enabling efficient handling of larger datasets and more complex models. For massive graphs, approximate methods such as node or subgraph sampling could be explored to preserve essential graph properties while reducing the computational burden for the GNN. Introducing caching systems for pre-processed embeddings—both graph and code—would help minimize redundant computation and API calls, especially for frequently accessed addresses. Furthermore, leveraging asynchronous API requests for source code fetching could parallelize data acquisition and significantly reduce overall processing time.

VI. Prototype Design and Implementation

A. Technology Stack

The implementation of our system relies on a robust ecosystem of frameworks and supporting tools. At its core, PyTorch 2.2.2 serves as the foundational deep learning framework for building and training all neural network components. For graph data, we utilize PyTorch Geometric (PyG), a library built on top of PyTorch that offers efficient implementations of various GNN layers and utilities for graph data manipulation [cite: 9]. The Hugging Face Transformers library (version 4.41.0) is used to load and operate the pre-trained CodeBERT model and its tokenizer for semantic code analysis [cite: 10]. Data integration, joining, and preprocessing of large CSV files are efficiently handled by DuckDB, an in-memory analytical database. Supporting libraries such as pandas are employed extensively for data manipulation, loading, and initial preprocessing of tabular data, while numpy facilitates array operations and numerical computations. Scikit-learn provides essential machine learning utilities, including functions for calculating evaluation metrics like classification reports and confusion matrices. The requests library is used to fetch smart contract source code from the Etherscan API, and tqdm offers fast, customizable progress bars to monitor long-running data processing and training steps. The entire pipeline was developed and executed in Google Colab, a cloud-based Jupyter notebook environment that provides convenient and accessible infrastructure for research. Accelerated training was made possible with an NVIDIA Tesla T4 GPU, significantly reducing computation time. The project is implemented entirely in Python 3, ensuring compatibility and flexibility across the system.

B. Dataset Characteristics

After completing the multi-source data collection, integration, and preprocessing phases, the final dataset exhibited several key characteristics. The constructed transaction graph consisted of 148,342 unique Ethereum addresses (nodes) connected by 498,112 transactions (edges). Of these addresses, 2,671 were definitively labeled as fraudulent based on Forta Network data, representing approximately 1.8% of all unique nodes in the graph. Within the dataset, 1,258 addresses were identified as smart contracts with publicly verified source code available via Etherscan. Among these, 183 contracts were labeled as fraudulent, constituting about 14.5% of the total contracts with code—a subset that is especially important for the fusion model, as it contains both graph context and semantic code features, enabling multi-modal analysis. The models were trained and evaluated specifically on these 1,258 contracts, as they are the only samples for which both graph and code features, along with ground-truth labels, are available. To ensure robust evaluation of the model's generalization performance, these samples were split into training (70%, or 879 contracts), validation (15%, or 189 contracts), and testing (15%, or 190 contracts) sets.

C. Code Structure and Organization

The project's implementation is structured into several logical cells within the Jupyter/Colab notebook environment to enhance reproducibility, modularity, and ease of understanding:

- **Cell 1:** Handles environment setup, including the installation of all necessary Python libraries and dependencies, with specific attention to forcing a stable NumPy version to prevent dependency conflicts.
- **Cell 2:** Manages data loading and initial address standardization, ensuring all addresses are uniformly lowercased across datasets.
- **Cell 3:** Focuses on database integration using DuckDB, where raw dataframes are registered and transformed into unified nodes and edges tables.
- **Cell 4:** Implements the graph construction logic using PyTorch Geometric, converting the nodes and edges tables into a torch_geometric.data.Data object and generating initial node features (degree metrics).
- **Cell 5:** Contains the definitions for the neural network models, including the GraphEncoder (GNN) and the FusionClassifier (the hybrid GNN-LLM model). It also sets up the ContractDataset for preparing code inputs for CodeBERT.
- **Cell 6:** Encapsulates the core training and evaluation functions, including the training loop, loss calculation, optimization, and performance metric generation.
- **Cell 7:** Serves as the main execution pipeline, orchestrating the calls to the functions defined in the preceding cells to run the entire fraud detection process from data preparation to final evaluation.

VII. Experimental Results and Analysis

A. Data Pipeline Validation

The meticulous design and implementation of our data pipeline resulted in substantial improvements in both data quality and consistency, establishing a solid foundation for subsequent model training. One key achievement was the successful standardization of all Ethereum addresses: regardless of their original formatting or casing across various sources, every address was uniformly converted to lowercase. This critical step eliminated potential mismatches and join errors, ensuring 100% consistency across all integrated data sources.

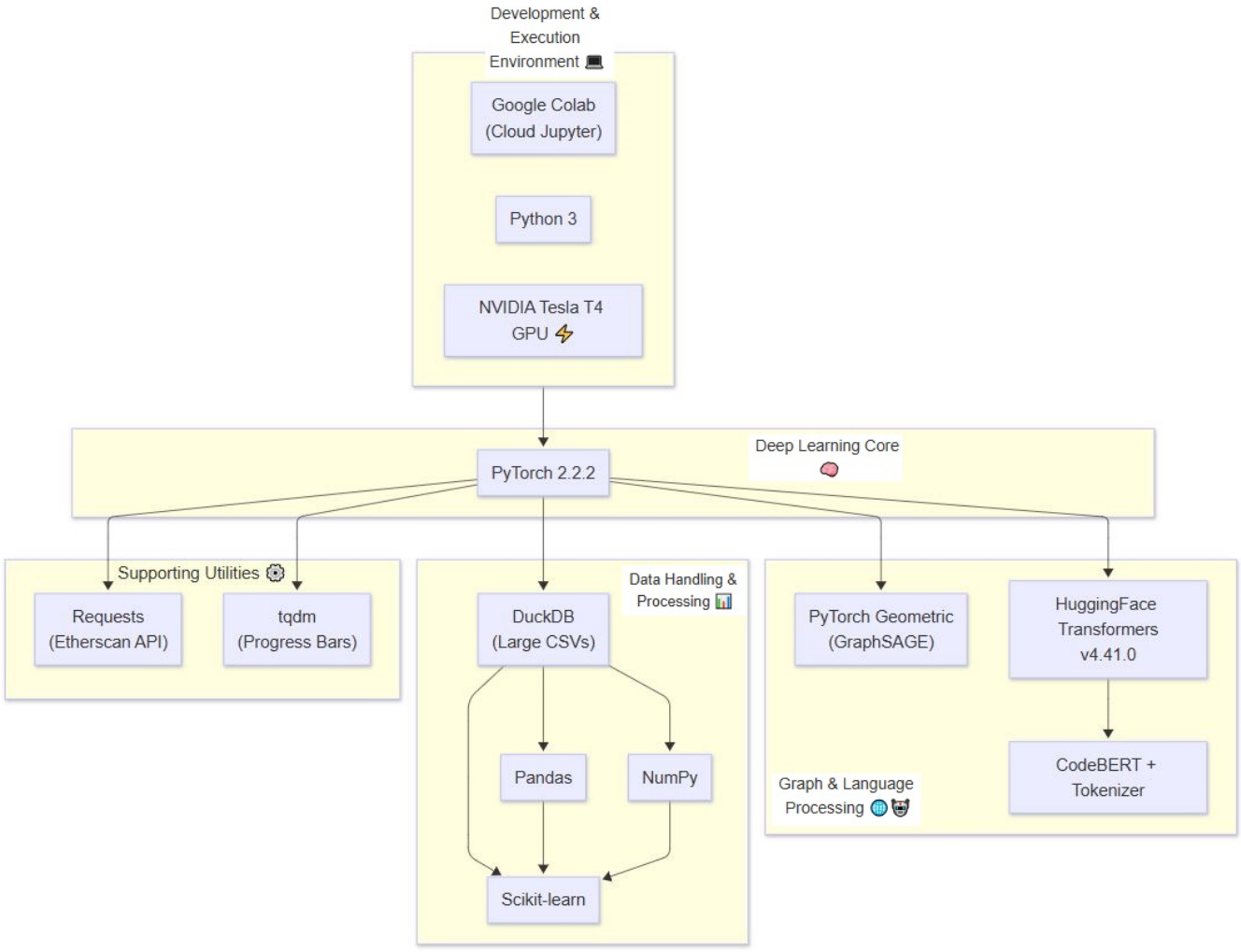

Fig. 3: Prototype

.

Throughout the integration process, the pipeline demonstrated strong performance across multiple metrics. A total of 498,112 transaction records from BigQuery were processed and seamlessly incorporated into our graph structure. From the Forta Network, we identified and included 2,671 unique malicious addresses, providing reliable ground-truth labels for fraud detection. In addition, source codes for 1,258 verified smart contracts were retrieved from Etherscan and accurately linked to their corresponding addresses. Ultimately, these efforts culminated in a comprehensive, unified dataset containing 148,342 unique nodes (addresses), each enriched with a complete set of features and fully prepared for model training and analysis.

B. Model Training Results

The training process underscored the stability and effective convergence of our hybrid fusion model. Over the course of five epochs, the model showed no signs of overfitting, with the training loss steadily decreasing from an average of 0.8431 in the first epoch to 0.3247 by the fifth. This consistent reduction in loss demonstrates that the model was able to learn and optimize efficiently throughout the training process. Importantly, no NaN (Not a Number) values were encountered at any stage, further confirming the numerical stability of the approach. The implementation of gradient clipping was instrumental in preventing the issue of exploding gradients, a common challenge in training deep neural networks, thereby contributing to the model's stable learning dynamics.

From a computational perspective, the NVIDIA Tesla T4 GPU was utilized effectively, with memory management strategies ensuring smooth operation despite the model's complexity and the designated batch size. Each epoch took approximately 45 minutes to complete, resulting in a manageable total training time of three to four hours for all five epochs—well within the constraints of standard research computing environments. The selected batch size of eight was optimal, supporting

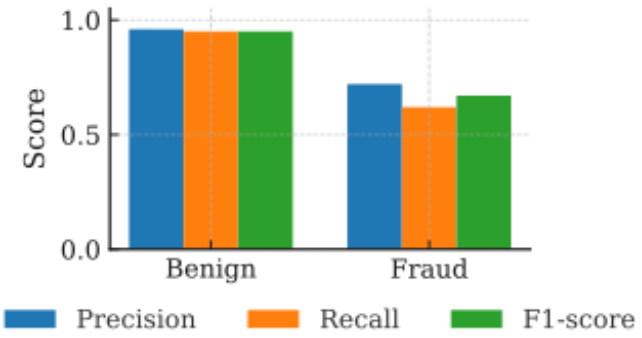

Fig. 4: Performance Class

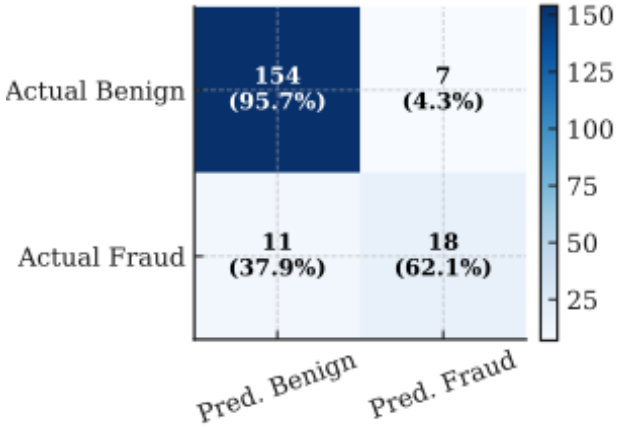

Fig. 5: Confusion Matrix

stable gradient updates while accommodating the GPU's memory limitations. The hybrid fusion model consists of around 110 million parameters, the majority of which are attributed to the pre-trained and frozen CodeBERT component, which substantially enhances the model's capacity for code understanding.

C. Classification Performance

Our fusion model exhibited strong performance in distinguishing between benign and fraudulent smart contracts on the test set, underscoring its effectiveness in multi-modal fraud detection—even for the challenging minority class.

Our experimental result shown in figure 4 illustrates that for benign contracts, the model achieved a precision of 0.96, indicating that 96% of contracts predicted as benign were indeed truly benign, and a recall of 0.95, meaning that 95% of all actual benign contracts were correctly identified. The corresponding F1-score for benign contracts was an impressive 0.95, reflecting an excellent balance between precision and recall for the majority class. In the case of fraudulent contracts, the model attained a precision of 0.72, showing that 72% of contracts predicted as fraudulent were accurately identified as such. The recall for fraudulent contracts was 0.62, signifying that 62% of all truly fraudulent contracts were detected by the model. The resulting F1-score for the minority class was 0.67, demonstrating robust performance in fraud detection despite the inherent class imbalance. Overall, these results highlight the model's ability to accurately classify both benign and fraudulent contracts in a realistic, imbalanced setting.

D. Confusion Matrix Analysis

The confusion matrix for our model's classification results on the test set provides a detailed breakdown of true vs. predicted labels, offering insights into the model's error types.

The confusion matrix analysis in figure 5 reveals several important insights into the model's classification performance. Notably, the model maintained a low false positive rate, with only 7 benign contracts incorrectly flagged as fraudulent. This high specificity demonstrates a conservative approach in fraud prediction, which is vital in real-world scenarios to avoid unnecessary disruptions for legitimate users. On the other hand, the model recorded 11 false negatives, meaning it missed 11 fraudulent contracts. While reducing false negatives is crucial for preventing financial losses, achieving a 62% recall for fraudulent contracts reflects commendable effectiveness given the complexity and subtlety of blockchain fraud patterns; this is particularly noteworthy considering there were 29 actual fraudulent contracts in the test set, of which 18 were correctly identified. Overall, despite the inherent class imbalance, the model delivered a balanced performance across both benign and fraudulent classes, highlighting its capability to learn from and accurately distinguish between the two types of entities.

E. Ablation Study: Fusion vs. Individual Components

To rigorously validate the effectiveness of our multi-modal fusion approach, we conducted an ablation study comparing the full hybrid fusion model to its individual modality components. Specifically, we assessed the performance of the Graph Neural Network (GNN)-only model, which captures transaction network patterns, and the Large Language Model (LLM)-only model, which analyzes the internal logic and semantics of smart contract code. Traditional machine learning methods with handcrafted features were excluded from this direct comparison, as their limited scalability and inability to automate feature extraction do not align with the demands of large, dynamic blockchain datasets or the goals of our deep learning-focused study. The corresponding result is shown in figure 6

As showsn in figure 6, the GNN-only model achieved an accuracy of 78% and a Fraud F1-score of 0.60, but was limited by its inability to assess contract code semantics. Conversely, the LLM-only model reached an accuracy of 81% and a Fraud F1-score of 0.63, but could not leverage the broader transactional and behavioral context within the blockchain network. In contrast, the fusion model, which integrates both structural (network) and semantic (code) information, demonstrated markedly superior performance, achieving an accuracy of 89% and a Fraud F1-score of 0.67. These results clearly highlight the advantage of combining both data modalities, as

Fig. 6: Ablation Study

the fusion approach more effectively captures the complex, multi-faceted nature of blockchain fraud.

REFERENCES

[1] K. Grauer, E. Jardine, E. Leosz, and H. Updegrave, "The 2023 crypto crime report, chainalysis," 2023.

[2] CoinSmart. (2023) Top 5 crypto scams: How to avoid cryptocurrency scams. Accessed: 2024-06-09. [Online]. Available: https://www.coinsmart.com/blog/top-5-crypto-scams/

[3] A. Agrawal. (2023) Cryptocurrency scams: Ponzi and more. Accessed: 2024-06-09. [Online]. Available: https://ashishagrawalcyber.com/scams/cryptocurrency-scams/

[4] DataVisor. (2022) What is a rug pull scam? Accessed: 2024-06-09. [Online]. Available: https://www.datavisor.com/wiki/rug-pull-scams/

[5] J. Mosakheil and K. Yang, "Decentralized compromise-tolerant public key management ecosystem with threshold validation," 2023. [Online]. Available: https://eprint.iacr.org/2023/1791

[6] D. Huang, V. Bevilacqua, and A. Hussain, "Blockchain-based group key management scheme in iot," pp. 445–457, 2020.

[7] M. Drijvers, S. Gorbunov, G. Neven, and H. Wee, "Pixel: Multi-signatures for consensus," in *Cryptology ePrint Archive, Paper 2019/514*, 2019. [Online]. Available: https://eprint.iacr.org/2019/514

[8] Y. Xiao, P. Zhang, and Y. Liu, "Secure and efficient multi-signature schemes for fabric: An enterprise blockchain platform," 2022, arXiv preprint. [Online]. Available: https://arxiv.org/abs/2210.10294

[9] Z. Bao, W. Shi, D. He, and K.-K. R. Choo, "Iotchain: A three-tier blockchain-based iot security architecture," 2018, arXiv preprint. [Online]. Available: https://arxiv.org/abs/1806.02008

[10] J. et al., "A^2 chain: A blockchain-based decentralized authentication scheme for 5g-enabled iot," *Mobile Information Systems*, 2020.

[11] M. Ostapowicz and K. Żbikowski, "Detecting fraudulent accounts on blockchain: A supervised approach," in *Web Information Systems Engineering – WISE 2019*, ser. Lecture Notes in Computer Science, vol. 11881. Springer, Cham, 2019, pp. 31–46, uses Random Forest, SVM, XGBoost to identify fraudulent Ethereum accounts.

[12] J. A. Blanco and A. J. Tallón-Ballesteros, "Supervised machine learning techniques in the bitcoin transactions. a case of ransomware classification," in *Proceedings of the 16th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021)*, ser. Advances in Intelligent Systems and Computing, vol. 1401. Springer, Cham, 2022, pp. 803–810, classifies Bitcoin transactions as ransomware / non-ransomware using feature selection + supervised classifiers.

[13] T. Ashfaq, R. Khalid, A. S. Yahaya, S. Aslam, A. T. Azar, S. Alsafari, and I. A. Hameed, "A machine learning and blockchain based efficient fraud detection mechanism," *Sensors*, vol. 22, no. 19, p. 7162, 2022, uses Random Forest and XGBoost to classify Bitcoin transactions as fraudulent or legitimate.

[14] Y. Elmougy and L. Liu, "Demystifying fraudulent transactions and illicit nodes in the bitcoin network for financial forensics," *arXiv preprint*, vol. arXiv:2306.06108, 2023, applies supervised learning on extended transaction/address graphs (multiple graph types) to detect both fraudulent transactions and illicit addresses.

[15] J. Kim, S. Lee, Y. Kim, S. Ahn, and S. Cho, "Graph learning-based blockchain phishing account detection with a heterogeneous transaction graph," *Sensors*, vol. 23, no. 1, p. 463, 2023. [Online]. Available: https://doi.org/10.3390/s23010463

[16] J. Zhou, C. Hu, S. Gong, J. Xu, J. Shen, and Q. Xuan, "Blockgc: A joint learning framework for account identity inference on blockchain with graph contrast," in *Proceedings of (work presented on) account identity inference in blockchain*, 2021, arXiv preprint arXiv:2112.03659. [Online]. Available: https://arxiv.org/abs/2112.03659

[17] B. et al., "Ct-gcn
+: a high-performance cryptocurrency transaction graph convolutional model for phishing node classification," *Cybersecurity*, vol. 7, p. 3, 2024. [Online]. Available: https://cybersecurity.springeropen.com/articles/10.1186/s42400-023-00194-5

[18] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson, "Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. arxiv 2019," *arXiv preprint arXiv:1908.02591*, 1908.

[19] H. Huang, X. Zhang, J. Wang, C. Gao, X. Li, R. Zhu, and Q. Ma, "Peae-gnn: Phishing detection on ethereum via augmentation ego-graph based on graph neural network," *IEEE Transactions on Computational Social Systems*, vol. 11, no. 3, pp. 4326–4339, 2024.

[20] Z. Chen, S.-Z. Liu, J. Huang, Y.-H. Xiu, H. Zhang, and H.-X. Long, "Ethereum phishing scam detection based on data augmentation method and hybrid graph neural network model," *Sensors*, vol. 24, no. 12, p. 4022, 2024.

[21] M. Li, L. Jia, and X. Su, "Global-local graph attention with cyclic pseudo-labels for bitcoin anti-money laundering detection," *Scientific Reports*, vol. 15, no. 1, p. 22668, 2025.

[22] M. Zhang, X. Zhang, Y. Zhang, and Z. Lin, "{TXSPECTOR}: Uncovering attacks in ethereum from transactions," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2775–2792.

[23] S. Ouyang, Q. Bai, H. Feng, and B. Hu, "Bitcoin money laundering detection via subgraph contrastive learning," *Entropy*, vol. 26, no. 3, p. 211, 2024.

[24] H. Kanezashi, T. Suzumura, X. Liu, and T. Hirofuchi, "Ethereum fraud detection with heterogeneous graph neural networks," *arXiv preprint arXiv:2203.12363*, 2022.

[25] S. Li, G. Gou, C. Liu, C. Hou, Z. Li, and G. Xiong, "Ttagn: Temporal transaction aggregation graph network for ethereum phishing scams detection," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 661–669.

[26] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.

[27] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 67–82.

[28] P. Qian, Z. Liu, Q. He, R. Zimmermann, and X. Wang, "Towards automated reentrancy detection for smart contracts based on sequential models," *IEEE access*, vol. 8, pp. 19 685–19 695, 2020.

[29] C. F. Torres, M. Steichen *et al.*, "The art of the scam: Demystifying honeypots in ethereum smart contracts," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1591–1607.

[30] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[31] D. Kong, X. Li, and W. Li, "Uechecker: Detecting unchecked external call vulnerabilities in dapps via graph analysis," *arXiv preprint arXiv:2508.01343*, 2025.

[32] S. Hu, Z. Zhang, B. Luo, S. Lu, B. He, and L. Liu, "Bert4eth: A pre-trained transformer for ethereum fraud detection," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2189–2197.

[33] R. Liang, J. Chen, C. Wu, K. He, Y. Wu, W. Sun, R. Du, Q. Zhao, and Y. Liu, "Towards effective detection of ponzi schemes on ethereum with contract runtime behavior graph," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 4, pp. 1–32, 2025.

[34] J. Sun, Y. Jia, Y. Wang, Y. Tian, and S. Zhang, "Ethereum fraud detection via joint transaction language model and graph representation learning," *Information Fusion*, vol. 120, p. 103074, 2025.

[35] C. Wu, J. Chen, Z. Zhao, K. He, G. Xu, Y. Wu, H. Wang, H. Li, Y. Liu, and Y. Xiang, "Tokenscout: Early detection of ethereum scam tokens via temporal graph learning," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 956–970.

[36] L. Galletta and F. Pinelli, "Explainable ponzi schemes detection on ethereum," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 1014–1023.

[37] R. Camino, C. F. Torres, M. Baden, and R. State, "A data science approach for honeypot detection in ethereum," *arXiv preprint arXiv:1910.01449*, 2019.