International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I7P124

Volume 74 Issue 7, 389-412, July 2026

Original Article

Lightweight Graph Neural Networks (LGNN) for Real-time Double-Spending Attack Detection in Blockchain Environments

Rintu Augustine[1], A. Krishnaveni[2]

[1,2]*Department of Computer Science, Karpagam Academy of Higher Education, Coimbatore, India.*

[1]*Corresponding Author : rintuaugustine07@gmail.com*

Received: 10 September 2025 Revised: 06 June 2026 Accepted: 18 June 2026 Published: 28 July 2026

Abstract - *To handle the main problem of double-spending attacks in blockchain networks, this paper introduces a new, Light-weight Graph Neural Network (LGNN) approach named Dynamic Sparse Graph Attention Network (DSGAT). To effectively detect double spending behavior, DSGAT method integrates adaptive graph sparsification with attention based on the fundamental graph-structured nature of blockchain transactions. Unlike computationally intensive GNNs, DSGAT may be implemented on edge devices or distributed monitoring systems with low-tech, low-cost hardware since it is optimized for resource-limited environments and doesn't need much processing capacity. To detect double-spending attack, this paper explains building blockchain transaction graphs from a large set of node and edge features. A set of simulated transactions involving double-spending attack is generated using large-scale simulations with the BCASim blockchain simulator, and the performance of DSGAT is compared with normal baselines. The experiment's outcomes prove that DSGAT is able to reduce model sizes and inference latency while keeping high detection rates, proving its feasibility and effectiveness for real-time double spending detection in low-resource environments. To improve blockchain security against double-spending attacks, this paper introduces a novel and realistic alternative.*

Keywords - *Blockchain, Graph Neural Network, Double Spending, Defensive Mechanism.*

1. Introduction

Blockchain has a decentralized and immutable ledger. Such technology fundamentally changes the digital transaction process, making it not require central intermediaries, but it leaves one key operational problem: it is vulnerable to the "double-spending problem," which basically refers to the illegal act of spending the same unit of digital currency more than once [1, 7]. This weakness becomes a decisive factor in undermining the integrity or trustworthiness of any digital financial system if left unredressed on blockchain technology [7, 9]. Different advanced double-spending attack forms have been realized and used in blockchain networks. Race Attacks happen when an attacker takes advantage of the time lag that exists between transaction propagation over a decentralized network. The attacker makes two opposing transactions, both of which spend the same digital currency, but with different destinations or services. These competing transactions are transmitted nearly concurrently to alternative sets of network nodes, with the attacker relying on the fact that one transaction will be confirmed before the other gets rejected [1, 2, 4]. This attack is especially successful against "zero-confirmation" transactions, in which a merchant takes payment prior to it being fully settled on the blockchain [2, 5]. Yet another prominent attack vector is the Finney Attack, which is named after Bitcoin pioneer Hal Finney. Here, an evil miner pre-mines a block with a transaction returning coins to an address that they possess. Importantly, this block is not published on the public network. At the same time, the attacker triggers an independent transaction utilizing the same coins to buy products or services from a merchant. If the merchant settles on this transaction without waiting for a high number of network confirmations, the attacker then propagates their already mined, held block. When this pre-mined block is confirmed by the network, it makes the transaction to the merchant invalid, so that the attacker gets to keep both the cryptocurrency and the goods or services obtained [13, 18]. The success of a Finney attack relies on the attacker being a miner and upon the timing of their action to the exact second, along with the merchant's acceptance of unconfirmed transactions [16, 17].

The most serious attack of double-spending is the 51% Attack. This attack happens when either an individual party or a synchronized collective of parties takes control of over 50% of the network's overall computational power (hashing power

in Proof of Work (PoW) networks) or of the majority of the staked cryptocurrency (in Proof of Stake (PoS) networks). Such control allows the attacker to control the blockchain by having a longer private chain, unwinding previously validated transactions, and essentially carrying out double-spend transactions [4, 16, 17]. Economically infeasible on large, mature networks such as Bitcoin, this attack is still a theoretical and feasible threat to smaller or newer blockchain networks with less dispersed control [7, 16].

Blockchain protocols are mainly based on double spending prevention methods. It uses strong consensus methods like Practical Byzantine Fault Tolerance (PBFT), Proof of Work (PoW), and Proof of Stake (PoS). All these methods ensure distributed node consensus on transaction validity and sequence, making double-spending economically or computationally infeasible [7, 16]. Confirmation techniques are also necessary as they make a transaction more secure, the more network confirmations it has (i.e., subsequent blocks are appended to the chain). Awaiting a high enough number of confirmations, e.g., six for Bitcoin as default, greatly lowers the risk of double-spending and transaction reversal [1, 17]. Furthermore, Unspent Transaction Output (UTXO) checking is utilized by platforms like Bitcoin to inhibit double spending at the fundamental level and to prevent any UTXO from being spent more than once [2, 18]. Ongoing monitoring of the network for double spending transactions helps to capture them early and prevent them [2]. The decentralized and dynamic nature of blockchain still creates challenges in spite of those measures, particularly when payments focus on fast processing when conventional confirmation protocols are not used.

Graph representation adapts well with the structure of blockchain transaction data. Graph will have nodes and edges. Nodes represent addresses of individual transactions, and edges represent interactions among these nodes [19]. To study the complicated relationship data contained in blockchain networks, Graph Neural Networks (GNNs) are a highly appropriate and effective deep learning model [19]. Unlike other deep learning models, such as traditional Recurrent Neural Networks (RNNs), which process sequential data, or Convolutional Neural Networks (CNNs), which process grid-structured data (such as images), GNNs are special. Rather, GNNs employ a "message passing" paradigm specifically designed to process non-Euclidean, graph-structured data [19]. Nodes within this paradigm increasingly send and gather information from local as well as multi-hop neighbors, modifying their own "embeddings" or representations [19]. Local structural patterns (such as instantaneous transaction patterns) and global dependencies (such as long transaction chains or intricate network relationships) may both be captured well by GNNs since to this cumulative process. This ability is important for detecting difficult or subtle deviant patterns, such as different types of double spending, which may be indicative of fraud. An increasing level of financial transactions and cryptocurrency applications are exploring the application of GNNs to standard fraud and anomaly detection [19, 21, 22].

Although cutting-edge GNN models proved their power and versatility in capturing intricate graph patterns, most such architectures are computationally costly. They generally require a substantial amount of memory and processing power, and to facilitate efficient training and inference, they often request the use of powerful Graphics Processing Units (GPUs) or large server farms [33]. Most notably, in resource-constrained environments, their real-world utilization within real blockchain monitoring systems is greatly impaired by their staggering resource demands. Such environments range from edge devices, Internet of Things (IoT) nodes, to even typical personal PCs with limited energy, memory, and processing capacity [31, 35, 38]. The "having no advanced systems for deployment" user limitation draws attention to this particular requirement of good and lightweight solutions, specifically. Such solutions should be able to function well without depending on high-performance computing resources [31]. The core purpose of lightweight GNNs is to considerably simplify model complexity, reduce the number of trainable parameters, and reduce inference time. This is accomplished while attempting to preserve competitive performance, making the models efficient at detecting anomalies despite minimal computational overhead [35]. The construction of such effective models is crucial to facilitating ubiquitous and accessible real-time blockchain security monitoring.

Regardless of applying Graph Neural Networks for the blockchain Anomaly detection, there are still some significant constraints that remain unattended. Some interesting approaches are based on general fraud detection and rely on computationally exhaustive architectures that assume the availability of high-performance hardware [12, 21, 22]. Lightweight GNN models are basically designed as general-purpose solutions and do not strictly consider the unique structural characteristics of double-spending attacks. Existing graph reduction techniques frequently deploy static or uniform sparsification strategies, which are capable of eliminating critical relations like Unspent Transaction Output (UTXO) patterns that play a major role in identifying double-spending behaviour.

Very limited concerns are given to real-time detections where the computational efficiency and detection accuracy must go hand in hand [15, 21, 22, 28, 30]. There is a need for specialized solutions since there is a lack of attack-aware model design, inadequate integration of temporal features and the absence of realistic simulation-based validation [15, 22, 24, 28, 48]. Hence, there prevails a very clear research gap to design a lightweight context-aware GNN framework that is capable to function within the given computational infrastructures and which is proficient to detect double spending attacks in the blockchain domains.

Although the strength of GNNs for blockchain anomaly detection is well known and addressed in some studies [12, 15, 21, 22, 24], there is a big gap in existing literature. Most notably, there is limited work on lightweight GNN models specially designed for real-time double-spending attack detection in resource-limited systems [12, 15, 21]. Current GNN-based use cases in this area tend to concentrate on overall fraud detection or take implicit advantage of access to powerful computational systems, ignoring the deployment challenges encountered by users with less advanced systems [15, 22, 28, 30]. In addition, while a few studies have suggested leveraging GNNs for double-spending attacks, these studies often fail to account for the "lightweight" character of the models and present an end-to-end simulation platform for testing them under realistic resource constraints [12, 48]. The actual novelty in this work is creating architectural schemes that are inherently efficient and incorporate methods of recognizing double spending attack patterns.

In this research, the necessity of resource-poor solutions is a core design criterion. Rather than being a functional constraint, the visible necessity of implementation in the lack of advanced technologies is a constraining necessity that impinges on the whole study method. This implies that all steps of the solution, ranging from the graph model of blockchain information to the design of GNN architecture and even the use of simulators, must integrate the "lightweight" nature [31, 32, 35, 38, 64, 65].

Another important part of this research is the application of theoretical concepts in GNNs into practical blockchain implementation. New model structures and their performance over static benchmark datasets are the focal points of the majority of GNN research papers [33, 36]. The simulation of the same with experimental results within a blockchain context necessitates showing how the GNN detects double spending attack scenarios in a dynamic blockchain network [12, 21]. To surpass the limitations of fixed graph datasets and allow a more dynamic, event-driven assessment, this requires a robust simulation framework capable of performing directed attacks and simulating blockchain propagation accurately [42, 46, 47, 48, 49]. This approach provides more realistic assessment of the model's performance under real-world blockchain environments [46-48].

The yet-to-be-realized potential of context-aware sparsification for double-spending detection is one very promising area of innovation. Current lightweight GNNs employ conventional graph pruning or sparsification for model compression and accelerating inference [35, 38], but these methods can risk removing significant edges or nodes utilized in double-spending detection. For instance, the foundation of a double-spending assault is generally rival or nearly simultaneous transactions using the same Unspent Transaction Output (UTXO) [2, 17, 18]. Such significant "conflict" edges may be removed by a non-discriminatory sparsification process if they fail to meet general relevance criteria [35, 38]. Thus, it would be novel to incorporate the sparsification process as being attack-pattern-conscious and context-conscious. This implies that while top-priority edges and nodes immediately connected to potential double-spend attacks, such as many outgoing transactions from the same UTXO within a very short time window, are retained, lower-priority connections are aggressively trimmed. Its efficient high-intelligence graph size reduction produces the optimal and most efficient double-spending detection method, making the model lighter without losing its capability to catch critical anomaly signals.

This work gives a fresh perspective that differs from existing lightweight GNN methods that can overcome all the mentioned constraints. The new model put forward includes a context-aware and dynamically adaptive sparsification mechanism that focuses on transaction relationships that indicate potential double-spending behaviour, unlike models that rely on static pruning or generic model compression techniques.

In particular, the method retains patterns such as rapid double-spending of UTXOs and conflicting transactions that occur in short time intervals, while removing less useful connections. This sparsification strategy, when combined with an attention-based aggregation mechanism, helps to focus on computational resources on the most important regions of the transaction graph [35, 38]. As a result, the new framework is well-suited for real-time deployment in blockchain environments that have limited resources, and it maintains a good balance between detection accuracy and computational efficiency. This differentiates it from existing GNN-based light-weight models, which lack explicit attack-aware optimization.

Its contributions towards addressing these opportunities and loopholes are as follows:

1) Lightweight GNN Approach: In the new GNN architecture, Dynamic Sparse Graph Attention Network (DSGAT) is used. Through the use of adaptive graph sparsification methods and a light and efficient attention-based aggregation, the suggested architecture not only directly addresses the need for resource-light solutions in situations of resource constraints but also reduces computation complexity and memory consumption. The architecture is derived from concepts of effective GNN search strategies like GASSIP [37], but for a static cost-effective architecture and not a complete Neural Architecture Search (NAS) approach that would be more viable for direct use.

2) Blockchain Transaction as Graph Representation: The GNN is carefully crafted with node and edge features, which are designed to identify double spending attacks.

3) Simulation-Driven Validation with BCASim: BCASim, an open-source blockchain simulator, is employed to develop a solid simulation platform [42]. Here, complex hardware elements are not used. BCASim could provide an array of double-spending attack scenarios alongside a controlled and realistic test bed for experimental verification [42, 48]. This would allow for successful testing in a range of difficult situations.

4) Empirical Demonstration of Efficacy and Efficiency: Extensive experimental results are presented to demonstrate DSGAT's superior performance in detecting various double-spending attacks. Crucially, these results also showcase its lightweight characteristics in terms of model size, inference time, and resource utilization, comparing it against existing GNN and non-GNN baselines to validate its practical utility for real-time anomaly detection in low-resource settings.

The remainder of this paper is organized as follows: Section II provides comprehensive background information on double-spending attacks, fundamental concepts of GNNs, and a review of related work in the field. Section III details the proposed Lightweight GNN Methodology, encompassing the blockchain transaction graph representation, the novel DSGAT architecture, and its mechanism for attack pattern recognition.

Section IV describes the simulation environment and the experimental setup used for validation. Section V submits and extensively discusses the experimental results acquired from the simulations. Finally, Section VI concludes the paper, highlighting the main findings and suggesting promising directions for further research.

2. Background and Related Work

2.1. Double-Spending Attack Mechanisms and Prevention

Double-spending, the inherent problem in digital currency, is defined as the unauthorized spending of a single unit of digital currency twice [7, 45]. In centralized traditional financial networks, this problem is naturally avoided by having a central authority that has an absolute record of all transactions. Without such a central authority in decentralized blockchain networks, strong cryptographic evidence and distributed consensus protocols are needed to protect transaction integrity [7, 45].

Some of the attack vectors take advantage of the intrinsic characteristics of blockchain networks:

- Race Attacks: These attacks take advantage of the time lag associated with transaction propagation over a decentralized network. An attacker broadcasts two vying transactions simultaneously: one to a merchant for a good or service (Tx1) and another, with the same amount, to an attacker address (Tx2). The attacker hopes to have Tx2 accepted by the larger network prior to Tx1 becoming irreversibly settled by the merchant. It is especially useful against merchants who accept "zero-confirmation" transactions, i.e., they provide goods or services prior to the transaction being part of a block and confirmed by the network [18, 45]. The flaw comes from the temporary status of unconfirmed transactions in the mempool of the network [17, 18].
- Finney Attacks: It is named after Hal Finney. Here, the attacker must be a miner. The attacker privately mines a block with a transaction (Tx_self) that pays coins from their wallet to another wallet controlled by them. The block is not broadcast at this point. Later, the attacker spends the same coins (from the source wallet) to buy something from a merchant (Tx_merchant) who will accept unconfirmed transactions. After the merchant deposits the goods, the attacker propagates their pre-mined block. If the network accepts this block, it renders Tx_merchant invalid, and the attacker gets to keep the goods and the coins [18, 45]. The viability of the attack is low on large networks because of the challenge of finding a fast block and the possibility that another miner would confirm the legitimate transaction before the attacker [14, 18].
- 51% Attacks: It's a deeper and resource-heavy attack. It happens when one party or a group of parties collectively get control of more than 50% of the network's overall computer power (hash rate in PoW) or a majority of the staked cryptocurrency (in PoS). This majority ownership enables the attacker to unilaterally control the blockchain: they can keep legitimate transactions from being validated, undo their own transactions (thereby double-spending), and even shut down the network [16, 45]. Whereas the economic cost of gaining 51% control in major networks such as Bitcoin or Ethereum is extremely expensive, smaller or newer blockchain networks with less dispersed control are still at risk [16, 50].
- Double Spending through Stored Transactions / Vector76 Attack: These are more advanced forms of race attacks, possibly with the generation of temporary forks of the blockchain or the concurrent broadcasting of several, conflicting transactions from a temporarily stored set [17, 18].

Prevention mechanisms against these attacks are built into blockchain design:

- Consensus Mechanisms: Mechanisms such as Proof of Work (PoW), Proof of Stake (PoS), and Practical Byzantine Fault Tolerance (PBFT) at the core of blockchain security confirm that all nodes in a network are unanimous concerning the validity and sequence of the transactions. For example, waiting for six confirmations is typical for Bitcoin to achieve finality of a transaction. [16, 45] Such a collective consensus renders it computationally or economically impossible for any

malicious agent to change the ledger and carry out double-spends [7, 16, 45].

- Confirmation Mechanisms: The transactions are not finalized until they have accumulated a sufficient level of confirmations—that is, additional blocks have been appended to the blockchain since the block that included the transaction. The higher the number of confirmations transactions receive, the safer they become exponentially, hence reversal or tampering being very difficult and costly. For example, waiting for six confirmations is conventional in Bitcoin for finality of a transaction [18, 41].
- UTXO Verification: In UTXO-based currencies such as Bitcoin, a transaction consumes Unspent Transaction Outputs (UTXOs) of past transactions and creates new UTXOs. The network enforces strictly that each UTXO can be spent only once. Before processing any new transaction, nodes verify that all referenced UTXOs have not already been spent, thereby fundamentally preventing double-spending attempts at the protocol level [17, 45].
- Network Monitoring: Blockchain networks can be monitored constantly in real-time for abnormal usage or unusual behavior, which allows to identify and prevent double-spending attempts. misbehavior of nodes, propagation delay, and transaction patterns need to be considered for this [15, 17].

2.2. Graph Neural Network Architectures

Graph Neural Networks (GNNs) were designed mainly to handle and learn from graph-structured, non-Euclidean information. They can capture intricate relationships and dependencies among nodes that are linked by edges [20, 33, 36, 66].

In contrast, conventional neural networks, like Convolutional Neural Networks (CNNs), are optimized to handle grid like input, such as images. Recurrent neural networks (RNNs) for sequential data are also in conflict to this [20, 33]. Consequently, applications such as social networks, chemical compounds and blockchain transaction histories are a perfect fit for GNNs [20, 21, 36, 66]. Information naturally occurs in network form in these areas. In general terms, GNNs possess a "graph-in, graph-out" design in the sense that they take a graph as input, where information is embedded within its global structure, nodes, and edges. Subsequently, the model updates these embeddings iteratively without breaking the input graph's intrinsic connectedness [20, 33, 36]. Message passing is the mechanism used in GNN [20, 33, 36]. Through iterative processing, each node collects information from surrounding neighbors. The node appends its previous state to the aggregated data upon receiving messages, refreshing its representation (embedding). To maintain both local structural patterns and global dependencies within the graph, the approach iterates across multiple layers, providing nodes with an opportunity to learn from neighbors that are more distant [20, 33, 36].

There are a number of prominent GNN architectures that have been developed:

- Graph Convolutional Networks (GCNs): Using a form of convolution directly on graph nodes, GCNs are some of the original GNN models. A non-linear activation function is then used after the feature of the node and the features of its closest neighbors are added together. A nonlinear activation layer, such as the use of ReLU, an output linear layer for making predictions, and a convolutional layer for achieving connections, can make up a basic GCN [20, 33, 36, 66].
- Graph Attention Networks (GATs): GATs overcome one limitation of regular GCNs, where neighbors can all be processed equally during aggregation. GATs incorporate an attention mechanism where nodes can assign differing learned weights to neighboring nodes. This allows the model to pay greater attention to more informative or significant neighboring information during aggregation without knowing the graph structure in advance. In graphs with different types of nodes and edges, some connections are more useful for detecting anomalies than others. In such cases, adjusting the weights of edges becomes very important [20-22, 24, 66].
- GraphSAGE: Instead of creating separate embeddings for every node, the inductive GNN model GraphSAGE takes a different approach. It learns node embeddings by sampling and pooling information from a node's local neighborhood attributes. Due to its inductive capability, GraphSAGE is an excellent option for dynamic graphs or generalizing to out-of-sample nodes and graphs in dynamic settings such as blockchain networks [12, 20, 36, 66].

The expressiveness of graph attention is perhaps its strongest feature. By its nature, GNNs are particularly well-suited for modeling intricate interplay among networked data. This strength is then further enhanced by the GAT mechanism, which enables the model to learn dynamically the relative weights of various neighbors or relations [21, 22, 24, 27, 28]. Not all neighbors in the anomaly detection problem are equally probable to predict double spending behavior. Some nodes are indications of fraud, but others are considered as merely innocent network noise. A GAT-based approach more accurately captures and isolate unusual patterns by assigning greater weight to suspicious acts. More accurate and dependable double spending attempts detection is enabled by this method [12, 21, 22, 24, 28]. The intrinsic value of attention mechanisms presents a strong case for future GNN designs to be centered around an attention-based framework for security use cases [21, 22, 24, 27, 28].

2.3. Lightweight GNN Techniques

Executing complex AI models like GNNs on devices with limited resources, such as smartphones or even standard computers, is becoming more and more necessary [33, 36].

"Lightweight" GNN models have been significantly influenced by desktop computers because desktop computers are often used to train large, resource-intensive GNNs, which highlights the need for more efficient models that can run on less powerful devices. This has driven the development of smaller, more streamlined GNNs that can be used on edge devices, smartphones, and even standard computers [31, 33, 36]. The primary goal of these methods is to drastically reduce computational overhead, memory needs, and power consumption while trying to maintain competitive or nearly state-of-the-art performance [31, 33, 36].

The following are some key techniques for building light GNNs:

- Sparsification of Graphs: It involves meticulously reducing the number of edges in an input graph so as to preserve the critical data needed for the next job, e.g., link prediction or node classification [38]. Sparsification methods may involve employing curriculum learning with a difficulty estimator that deletes edges to identify redundant edges, learning concise graph structures using differentiable masks, or even similarity-based kernels to retain only dominant edges [38]. By reducing connection density, graph sparsification can efficiently decrease the computation cost associated with message-passing and aggregation procedures in GNNs [38].
- Network Pruning: Building on the "lottery ticket hypothesis" that posits that very efficient sub-networks are contained within larger neural networks, network pruning is a systematic removal of "insignificant" parameters, connections, or even whole functioning units from the neural network [33, 36]. The goal of this is to obtain similar performance as the original, unpruned network, but with much smaller model size and quicker inference times [33]. Pruning can be on weights, layers, or individual operations in the GNN architecture [33, 36].
- Knowledge Distillation: This technique trains a smaller, lighter "student" model to mimic the behavior and output of a larger, more complex "teacher" model. The student is trained on the soft probabilities or the intermediate representations generated by the teacher, so it can take advantage of much of the teacher's performance without being much heavier or slower to infer [33, 36].
- Neural Architecture Search (NAS): NAS offers a method of automatic design of the best neural network architectures [37]. For GNNs, light GNAS is specially interested in discovering highly efficient architectures for specific tasks and constrained resources [37, 38]. While normally computationally expensive during searching, discovered architectures can be highly efficient to deploy [37].
- Simplified Architectures: Another avenue is architecturally simplifying simpler GNN layers or merely reducing the number of layers within the network as a whole. This simple simplification is able to produce light models that are easier to deploy and use. An example is LightGNN, which demonstrates how to simplify GNN architecture but still perform decently well for tasks such as recommendation systems [35].

Dynamic sparsification as an in-real-time efficiency technique is particularly relevant to blockchain applications. Although static graph sparsification effectively reduces the edges and accelerates GNN computations [38], the dynamic character of blockchain networks guarantees that the graph itself is dynamically changing with new transactions and blocks [21, 22, 24].

A static sparsification would result in removing critical edges relating to newly emerging, suspicious transactions at the cost of detection precision [22, 24, 38]. Recent surveys have also investigated lightweight and secure blockchain frameworks in various related domains. For example, a lightweight blockchain architecture designed for smart home environments emphasizes optimized resource utilization while enhancing data security and privacy for IoT devices [64].

Likewise, for the industrial IoT systems, the integration of federated learning with blockchain has also been put forward, which enables lightweight and distributed detection of zero-day attacks while maintaining high accuracy and system security [65].

A dynamic or adaptive sparsification method would be more efficient. This procedure would prune or focus only on relevant subgraphs or relations intelligently during real-time inference or in real time.

For instance, it may favor connections based on the latest, high-value, or possibly conflicting transactions, thereby remaining very efficient but without compromising the detection of emerging threats in real time [22, 24, 38]. This manner ensures that the model remains light and responsive to the dynamic nature of blockchain data [22, 24].

2.4. GNN Applications in Blockchain Anomaly Detection

Graph Neural Networks have become widely popular of late due to their capabilities in detecting all kinds of anomalies in the blockchain networks. Because they are capable of depicting complex patterns of relationships, they are particularly well suited to examine cases of criminal behavior, money laundering, and other types of financial fraud [19, 21, 22, 66].

A few applications of GNNs here are:

- Fraudulent Transaction Detection: Various papers have used a range of GNN architectures, such as Graph Convolutional Networks (GCNs) and Graph Attention

Networks (GATs), and their extensions to detect fraudulent Bitcoin transactions. Such research evidently demonstrates the superior predictive nature of GATs owing to their mechanism of attention, through which the model can provide a different weight to different neighbors in the graph [21, 22, 27]. This is crucial in detecting subtle fraud patterns from an enormous number of genuine transactions.

- Double Spending Detection: GNNs have been adapted particularly to detect double spending attacks on Bitcoin. These models typically employ observations from a set of observer nodes to predict whether or not each node in the network's mempool of payment transactions maintains a specific payment transaction. Double-spending attacks have been detected effectively by such methods [12]. The ability of GNNs to scan the network for patterns of incompatible transactions and double-spending attempts is what makes such detection possible [12, 17].
- Transaction flow anomaly detection has also been handled by stronger GNN models, including Multi-Distance Spatial-Temporal Graph Neural Networks (MDST-GNN). These models integrate multi-distance graph convolutional architectures with adaptive temporal models to learn local and global spatial and temporal connections and temporal relations among cryptocurrency transactions to enhance features in anomaly detection [28]. Similarly, some Spatial-Temporal GNN (ST-GNN) models leverage GCN or GAT for encoding space and GRU or Temporal Convolutional Networks (TCN) for modeling the time-evolving adaptation of transaction behaviors to detect anomalies off the norm from normal network behaviors [24].

Good feature engineering is extremely important in successful GNN-based anomaly detection. Common features extracted from blockchain transaction graphs for GNNs are: inDegree (sum of incoming transactions), outDegree (sum of outgoing transactions), totalInput (received amount), totalOutput (sent amount), inout-ratio (input to output ratio), and unique-out (count of unique output addresses) for transactions. For wallets or addresses, characteristics like asASender (times as sender), asAReceiver (times as receiver), totalSpent, totalReceive, current_balance, and several centrality measures (e.g., degree centrality for direct linkages, betweenness centrality for participation in shortest paths, and eigenvector centrality for network influence) are important [21, 22]. When considered as a total, these characteristics offer an understanding of transactional behavior and network life.

Even with these phenomenal improvements, however, there are various problems with existing GNN methods in the detection of blockchain anomalies. Among them are rendering model decisions explainable, dealing with large, noisy real-world data effectively, capturing multi-hop neighbor information correctly, and addressing the intrinsic data imbalance (where abnormal transactions are more scattered than legitimate ones) [20, 29, 30]. Scalability and computational resources are two significant challenges to applying such computationally expensive models to realistic real-world scenarios [20, 33, 36].

Furthermore, the recent studies highlight the upcoming role of Graphical Neural Networks (GNNs) in blockchain security, in particular modelling transaction networks and detecting fraudulent activities through structural and relational learning [21, 22, 66].

These studies showcase the effectiveness of architectures like Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and GraphSAGE in capturing complex transaction dependencies and also identifies challenges related to scalability, privacy preservation, and efficient deployment in real-world environments [21, 22, 36, 66].

The importance of temporal features for double-spending detection cannot be overstated. Time-sensitive double spending attacks, such as Race and Finney, are so by nature. Easy opportunity windows, e.g., transaction propagation delays or time to confirm a transaction, are taken advantage of by the attackers [1, 13, 14, 18].

Taking into account and handling temporal aspects in an appropriate manner is therefore essential. Whereas descriptive features, such as timestamp (of creation of or broadcast of a transaction), time_since_last_seen_input (of the detection of fast re-spending of a UTXO), and activity_rate (rate of transactions of an address in the recent past) are themselves explicit measures of causality for unusual behavior [24, 28]. Extremely fast respending of an unspent output or a concentration of activity in an address are strong indicators. To properly capture the time-varying and dynamic nature of such attacks, a GNN model is thus in a position to process and leverage such temporal information, possibly via distinct recurrent units or temporal attention mechanisms [24, 28]. A comparative study of existing methods is shown in Table 1.

Table 1. Comparison of existing methods and proposed DSGAT approach

Approach	Method Type	Graph-Based	Lightweight	Real-Time	Attack-Aware	Key Limitation
Consensus Mechanisms (PoW, PoS, PBFT) [16, 45]	Protocol-level security	No	Yes	No	No	Reactive, not detection-based
Confirmation Methods [1, 18, 40]	Transaction validation	No	Yes	No	No	Delayed detection (needs confirmations)

UTXO Verification [2, 12]	Rule-based validation	No	Yes	Yes	Partial	Cannot detect complex fraud patterns
GCN-based Models [19, 21, 22]	Graph Neural Network	Yes	No	Limited	No	High computational cost
GAT-based Models [19, 21]	Attention-based GNN	Yes	No	Limited	Partial	Not optimized for resource constraints
GraphSAGE [12, 66]	Inductive GNN	Yes	Partial	Limited	No	Limited attack-specific modeling
Static Sparsification [38]	Lightweight GNN	Yes	Yes	Yes	No	May remove important fraud edges
Pruning / Distillation [33, 35]	Model Compression	Yes	Yes	Yes	No	Generic optimization, not attack-aware
Lightweight Blockchain [64]	Blockchain Framework	No	Yes	Yes	No	No graph-based anomaly detection
FL + Blockchain [65]	Distributed Learning	No	Yes	Yes	Partial	No transaction-level graph modeling
GNN Survey [36, 66]	Survey / Analysis	Yes	-	-	-	No implementation or optimization
Proposed DSGAT	Dynamic Sparse GAT	Yes	Yes	Yes	Yes	-

3. Proposed Lightweight GNN Methodology for Double-Spending Detection

The Dynamic Sparse Graph Attention Network (DSGAT), a state-of-the-art, resource-thrifty GNN model designed particularly for efficient and high-accuracy detection of double-spending threats in low-resource blockchain systems, is introduced in this paper.

To ensure both efficient execution and high detection accuracy, the approach is formulated based on an adaptively sparse GNN structure combined with a specially developed blockchain transaction graph representation.

3.1. Blockchain Transaction Graph Representation

To effectively leverage the power of GNNs, raw blockchain transaction data must be mapped to a graph representation in a structured format [25, 54, 55].

This paper introduces a heterogeneous transaction graph that embeds various types of entities and their complex relationships within the blockchain system.

Nodes: There are three primary types of nodes in the graph:

- Transaction Nodes (T-nodes): Every transaction entered onto the blockchain is represented in terms of a T-node. The nodes play a critical role in monitoring the flow of cryptocurrency [23, 25].
- Address Nodes (A-nodes): Address nodes represent cryptocurrency wallet addresses. Addresses are the origin and targets of transactions, with them holding entities (users or smart contracts) participating in exchange within the network.
- Block Nodes (B-nodes): Each authenticated block in the blockchain is represented as a B-node. The nodes constitute the structural and chronological structure of the blockchain, linking transactions into an unalterable ledger [6, 8, 10, 11, 60, 61].

Edges: The edges between these nodes are denoted by various types of edges:

- Input Edges (A -> T): An edge from a Transaction Node to an Address Node indicates the address is an input (sender) to that specific transaction. This is the utilization of Unspent Transaction Outputs (UTXOs) by an address [6, 8, 10, 11].
- Output Edges (T -> A): An edge from a Transaction Node to an Address Node tells us the address is an output (recipient) of the given transaction. This is creating new UTXOs [6, 8, 10, 11].
- Block Inclusion Edges (T -> B): An edge from a Transaction Node to a Block Node indicates that the transaction is included and confirmed in the said block. This is the determination of the on-chain confirmation status of a transaction [6, 8, 10, 11].
- Block Linkage Edges (B -> B): These are the edges that connect Block Nodes in their sequential, chronological order to form the chain of cryptography that represents the blockchain itself. This is the network's history and immutability [6, 8, 10, 11, 60, 61, 63].

Feature Engineering: Besides the richness of the graph representation above, for every node and edge in the graph, a complete set of features is extracted. Carefully engineered, these features will often highlight patterns and anomalies that

are predictive of double-spending behavior. Table 2 summarizes node and edge features of the proposed system.

Transaction Node Features (T-nodes):

- transaction_id: A unique cryptographic reference (hash) per transaction.
- timestamp: The very moment when the transaction was created or first broadcasted. It is a significant temporal property for the determination of time-sensitive attacks.
- value_transferred: The definite value of cryptocurrency moved by the transaction.
- num_inputs: The number of input addresses (UTXOs) consumed by the transaction.
- num_outputs: The number of output addresses generated by the transaction.
- fee: The transaction fee paid to miners/validators.
- is_zero_confirmation: A 0-or-1 binary value indicating whether or not the transaction has yet received any confirmations on the block. This is extremely helpful in spotting weakness against Race and Finney attacks.
- input_output_ratio: The ratio between the total value of the inputs and the total value of the outputs. Unusual ratios may suggest strange splits, consolidations, or attempts at laundering flows of transactions.
- time_since_last_seen_input: Amount of time since any of the UTXOs utilized as inputs to this transaction were last observed in an affirmed transaction. Little time equals rapid re-spending, a robust double-spending attempt indicator.

Address Node Features (A-nodes):

- address_id: A globally unique identifier for all cryptocurrency wallet addresses.
- current_balance: Current balance of cryptocurrency owned by the address.
- total_spent: Total spent by all cryptocurrency of this address over its lifetime.
- total_received: Total value of all cryptocurrencies received by this address throughout its existence.
- num_transactions_as_sender: Transactions in which this address was an input (sender).
- num_transactions_as_receiver: Transactions in which this address was an output (recipient).
- avg_transaction_value: Average value of transactions with this address as sender or receiver.
- activity_rate: Number of transactions from this address within a recent, fixed timeframe (e.g., recent hour, recent day). A sudden spike in activity could be unusual.
- centrality_measures: Various graph measures of centrality, which give a measure of how central or influential an address is in the network [21]:
  - Degree Centrality: Number of direct connections (transactions) an address has.
  - Betweenness Centrality: Measures how many times an address shows up on the shortest path between other address pairs. High betweenness can indicate a "bridge" or middle position in suspicious flows.
  - Eigenvector Centrality: Measures an address's impact in relation to its neighbors' centrality. An address connected to a high number of high-centrality addresses will also be high in eigenvector centrality.

Block Node Features (B-nodes):

- block_height: The relative position of the block within the blockchain.
- timestamp: The date and time when the block was mined and added to the chain.
- num_transactions: Quantity of transactions that form this block.
- block_size: Block size in bytes.
- miner_id: ID of the miner or mining pool that successfully mined the present block.

Edge Features:

- value_flow: For A -> T and T -> A edges, the value of cryptocurrency that is transferred on this particular connection.
- time_difference: In the case of B -> B edges, this is the time elapsed since the current block was mined and its predecessor block, a measure of block creation speed.
- confirmation_status: In the case of T -> B edges, this might be the number of blocks appended after the block that holds the transaction, a measure of depth and futurity in the chain.

One important aspect in this model is temporal consideration for double spending detection. Attackers use time based double spending methods in order to exploit small temporal windows, like the delay before a transaction receives sufficient confirmations [1].

Consequently, the likes of activity_rate, timestamp, and time-since-last-seen-input features are not only descriptive but also causal indicators of maladaptive behavior. Strong red flags are very strong, such as the sharp increase in activity from a specific address or high-volume respending of an output that has never been respent previously.

To effectively represent the time-varying and dynamic nature of such attacks, the GNN structure should be capable of processing and leveraging such temporal features usefully in some manner, perhaps by utilizing specially engineered recurrent architectures or temporal attention.

3.2. Novel Lightweight GNN Architecture: Dynamic Sparse Graph Attention Network (DSGAT)

The proposed Dynamic Sparse Graph Attention Network (DSGAT) aims to solve 1) correct double-spending identification, and 2) low-latency operation in resource-limited environments. The design uses the best features of Graph Attention Networks (GATs) to create the model more efficient. It achieves this with new methods for dynamically removing less important connections, which makes the operation lightweight and fast. Focusing on network trimming and graph sparsification for resource-constrained applications, the method learns from effective GNN design concepts [35].

DSGAT - made up of multiple essential elements:

3.2.1. Input Layer and Initial Embedding

The first thing is to tidy up the raw data from the blockchain and feed it through an "initial embedding layer," which is similar to a translator [54, 55].

The task of this layer is to convert all the various types of raw attributes from the three primary categories of nodes (transactions, addresses, and blocks) into a single, uniform, and condensed numerical form. This newly created format(vector) is what the model operates on as the input for all its computation.it takes all the different raw features and converts them into a standardized numerical format. This initial embedding then serves as the fundamental features for each item, representing its core characteristics in a way the model can understand.

3.2.2. Dynamic Graph Sparsification Module

It is Dynamic Graph Sparsification Module that makes the DSGAT model so light and efficient. This module does not eliminate connections permanently, unlike conventional techniques, but prunes or reorganizes the graph dynamically. This is important because, without realizing it, it is not possible to eliminate critical connections that could be associated with threats that resurface in the future.

Table 2. Proposed node and edge features for blockchain transaction graph

Feature Category	Feature Name	Description	Node Type(s)	Relevance to Double-Spending Detection
Transaction Node Features	transaction_id	Unique hash identifier	T-node	Transaction uniqueness, linking conflicting transactions
	timestamp	Time of transaction creation/broadcast	T-node	Crucial for identifying simultaneous/race conditions
	value_transferred	Amount of cryptocurrency involved	T-node	Identifies high-value targets for attacks
	num_inputs	Number of input addresses (UTXOs)	T-node	Abnormal input patterns (many inputs from one source)
	num_outputs	Number of output addresses	T-node	Abnormal output patterns (many outputs to new addresses)
	fee	Transaction fee	T-node	Attackers might use higher fees for priority in race attacks
	is_zero_confirmation	Binary flag if unconfirmed	T-node	Direct indicator of vulnerability to Race/Finney attacks
	input_output_ratio	Ratio of total input value to total output value	T-node	Unusual value distributions in fraudulent transactions
	time_since_last_seen_input	Time since input UTXO was last spent	T-node	Detects rapid re-spending of the same UTXO
Address Node Features	address_id	Unique address identifier	A-node	Entity tracking for repeated suspicious behavior
	current_balance	Current cryptocurrency balance	A-node	Identifies addresses with funds available for double-spending
	total_spent	Cumulative amount spent by address	A-node	Historical spending patterns, unusual spikes

	total_received	Cumulative amount received by address	A-node	Historical receiving patterns, unusual spikes
	num_transactions_as_sender	Count of transactions as sender	A-node	High frequency of outgoing transactions, especially conflicting ones
	num_transactions_as_receiver	Count of transactions as receiver	A-node	Patterns of receiving funds from suspicious sources
	avg_transaction_value	Average value of transactions involving address	A-node	Deviation from normal transaction value for an address
Block Node Features	block_height	Position in blockchain	B-node	Chronological context, fork detection in 51% attacks
	timestamp	Time of block creation	B-node	Block generation rate anomalies, fork timing
	num_transactions	Number of transactions in block	B-node	Unusual block content or size
	block_size	Size of the block	B-node	Anomalies in block construction
	miner_id	Miner identifier	B-node	Identifying malicious miners in Finney/51% attacks
Edge Features	value_flow	Amount transferred along edge	A->T, T->A	Magnitude of suspicious transfers
	time_difference	Time between connected blocks	B->B	Abnormal block generation intervals in private chains
	confirmation_status	No. of confirmations for transaction	T->B	Direct measure of transaction finality and vulnerability

Each edge and node will be assigned a "suspicion score "depending on the module's functionality, Value_flow (higher value transactions being more likely to be targeted), time_since_last_seen_input for input edges (quick responding of a UTXO), and the presence of conflicting transactions (two mempool transactions from the same UTXO, for example) are some of the features that are considered for edges. Activity_rate (sudden increases) or is_zero_confirmation for T-nodes can be the premise for suspicion ratings for nodes. After these two subgraphs, output is passed to either a learned thresholding function or a differentiable gating function. Low suspicion score edges and nodes, or the nodes that are considered less important for discovering double-spending patterns, are either eliminated completely (hard pruning) or assigned with significant weights almost as low as zero (soft sparsification). This dynamically creates a sparse subgraph that the subsequent GNN layers will be operating on. The architecture effectively reduces the size of the graph without removing critical anomaly signals, leading to a highly efficient and optimal solution for detecting double-spending. This ensures that computational resources are focused on the most critical segments of the graph, which are crucial for real-time efficiency on low-resource devices.

3.2.3. Multi-Head Sparse Graph Attention Layers

Following the sparsification, the embeddings are passed through multiple layers of sparse graph attention. Each attention head independently computes attention weights between a node and its unpruned neighbors. This allows the model to learn the relative significance of different neighbors in terms of the dynamically sparse graph. The attention mechanism, similar to traditional GATs [19], is particularly useful here since it can adaptively learn to give more weight to conflicting transactions or unusual transaction patterns, leading to better double-spending detection. The "sparse" nature means that attention is only computed over the dynamically selected, informative neighborhood, reducing computational cost greatly compared to full-graph attention [34]. The representations from different attention heads are concatenated or averaged to provide a richer, more comprehensive representation for each node.

3.2.4. Node-Specific Feature Aggregation

Due to heterogeneity of the graph (T-, A-, and B-nodes), a specific aggregation method is employed. Node embeddings are updated as messages pass through attention layers. Terminal embeddings of T-nodes (transactions) are

specifically useful for classification. These end T-node embeddings are calculated by the light, fully connected neural network and then output a binary classification: "legitimate transaction" or "double-spending attempt." In order to maintain the model light, this final classification layer is made as small as possible.

3.2.5. Network Pruning (Post-Training Optimization)

Network pruning targets the model parameters, while dynamic sparsification targets the graph structure. The attention layer and final classification layer weights and connections can be further explored for additional pruning following initial training. The "lottery ticket hypothesis" is a post-training optimization technique to further decrease model size and accelerate inference without considerably decreasing performance [35]. By performing this, the deployed model will have the smallest possible footprint. DSGAT's principal innovation is its end-to-end dynamic, context-aware graph sparsification, optimized for double spending detection and incorporated into a paradigm of attention-based GNN. The existing approaches, meanwhile, will employ domain-independent pruning or static sparsification. DSGAT preserves the high accuracy of typical non-sophisticated systems while remaining under the stringent resource constraints by strategically directing real-time computational capability toward the most doubtful segments of the transaction graph. The design strongly resists double-spending attack forms by emphasizing the monetary identification of conflicting UTXO usage and reacting quickly by respending while being efficient with the computation.

3.3. Double-Spending Attack Pattern Recognition with GNN

Some graph patterns and abnormalities that signify a double spending attack are identified by the DSGAT model [23, 25, 39]. The output from the GNN, typically trained node embeddings, is first used for the purpose of a node-level classification procedure to provide a legitimate or malicious label to every transaction node (T-node) [23, 25, 26]. Complex attack signatures can be identified as to Section 3.1's hand-crafted features and the GNN's ability to collect data while traversing the graph.

The following patterns are identified by the GNN upon training:

3.3.1. Conflicting UTXO Usage

A significant sign of double-spending is when a user tries to spend the same Unspent Transaction Output (UTXO) multiple times. [2] One input edge from an Address Node (a UTXO) to two or more different Transaction Nodes that are transmitted within a narrow time window is how this appears in the graph. The GNN, in relaying messages, can transmit the "spent" status of a UTXO through the graph, sensing when two or more transactions try to spend the same input at the same time.

3.3.2. Race Condition Signatures

Within Race Attacks, the model discovers two racing transactions (Tx_A and Tx_B) from the same source address or UTXO, which are released nearly at the same time. The temporal features (timestamp, time_since_last_seen_input) and the is_zero_confirmation flag on the T-nodes are given top priority. The GNN learns to sense the anomalous propagation behavior where two transactions race to get accepted, potentially in different regions of the network, before being confirmed [2].

3.3.3. Finney Attack Signatures

Finney attack identification entails identification of a transaction (Tx_merchant) by an identified attacker's address, which remains is_zero_confirmation (unconfirmed), and subsequent hurried broadcasting of a previously withheld block (defined by B-node fields such as block_height and timestamp) with a conflicting transaction (Tx_self) from the same attacker's address to themselves. The GNN can connect these temporal and structural irregularities to identify the pattern of a miner trying to double-spend on receiving goods [3].

3.3.4. Signals of 51% Attack

Although a complete 51% attack is network-based, DSGAT can help detect it by picking up on localized irregularities that might result in or coincide with such an attack. These include sudden, uncharacteristic patterns in block_linkage_edges (like a sudden increase in block height by a single miner_id other than the normal chain), or repeated reversals of transactions from one miner's blocks. Using this GNN, it is possible to identify when a malicious chain tries to take over the real blockchain [43]. It does this by specifically looking at the time_difference between blocks and the unique patterns of the miner_id, which helps the model spot forks in the blockchain's structure [9].

Suspicious addresses can be identified by looking at two key features: activity_rate and centrality_measures. If a suspected attacker's address suddenly and abnormally increases its activity_rate, or shows an unusual betweenness_centrality within a specific set of adversarial transactions, it could be a sign of unusual behavior [21]. The GNN is trained to associate these anomaly patterns at the address level with double-spending attempts at the transaction level.

The result of the DSGAT model for every transaction node is a probability score, being the probability of its involvement in a double-spending attack. The scores are then classified as legitimate or malicious according to a pre-specified threshold [23, 25, 39, 56]. The model's capability to concatenate both direct neighbors' and multi-hop relations' information, together with its dynamic sparsification, renders it skilled in identifying such complex, frequently secretive, fraud patterns efficiently in real-time [23, 25, 26, 34].

3.4. Mathematical Formulation of DSGAT

To give a base to the Dynamic Sparse Graph Attention Network (DSGAT), this section introduces the mathematical formulation of the attention mechanism, dynamic sparsification strategy, and training objective.

3.4.1. Graph Attention Mechanism

$$h_i' = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W\, h_j\right) \quad (1)$$

Where:

h_i: input feature vector of node i
h_i': updated node representation
W: learnable weight matrix
$\mathcal{N}(i)$: neighborhood of node i
α_{ij}: attention coefficient between nodes $i$and j
σ: non-linear activation function

The attention coefficients are computed as

$$\propto_{ij} = \frac{\exp(\text{LeakyReLU}(a^T[W\, h_i \,||\, W\, h_j]))}{\sum_{k \in N} \exp(\text{LeakyReLU}(a^T[W\, h_i \,||\, W\, h_k]))} \quad (2)$$

Where:
a: learnable attention vector
||: concatenation operation
This mechanism allows the model to assign higher importance to suspicious neighboring transactions.

3.4.2. Dynamic Sparsification Mechanism

In order to make the computation efficient without losing any important transactional relationships, a dynamic sparsification approach has been proposed.

Suspicion Score

$$S_{ij} = \lambda_1 . \alpha_{ij} + \lambda_2 . T_{ij} + \lambda_3 . U_{ij} \quad (3)$$

where:

S_{ij}: importance score of edge (i, j)
α_{ij}: attention weight
T_{ij}: temporal feature (e.g., time difference between transactions)
U_{ij}: UTXO conflict indicator (1 if conflict exists, else 0)
$\lambda_1, \lambda_2, \lambda_3$: weighting parameters

Pruning Condition:

$$e_{ij} = \begin{cases} 1 & if\ S_{ij} \geq r \\ 0 & otherwise \end{cases} \quad (4)$$

where:

e_{ij}: edge retention indicator
τ: sparsification threshold

The approach allows keeping only those edges having higher importance measures in order to preserve important double-spending patterns without making unnecessary edges.

3.4.3. Loss Function

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} [y_i\, log(\hat{y_i}) + (1 - y_i) \log(1 - \hat{y_i})] \quad (5)$$

where:

y_i: true label (0: normal, 1: double-spending)
$\hat{y}_i$: predicted probability
N: number of samples

3.4.4. Objective

The aim of the proposed approach is to reduce the errors while maintaining important edges and being computationally efficient.

The novel algorithm DSGAT incorporates the idea of graph attention in combination with dynamic sparsification for the detection of double-spending patterns.

Algorithm 1: Dynamic Sparse Graph Attention Network (DSGAT)

Input:

G = (V, E) refers to the transaction graph
X refers to the Node feature matrix
τ refers to the sparsification threshold
$\lambda_1, \lambda_2, \lambda_3$ refers to Weight parameters
T_{ij} refers to Temporal features
U_{ij} refers to the UTXO conflict indicator

Output:

Yhat refers to predicted labels (0: normal, 1: double-spending)

Begin

Initialize weight matrix W and attention vector a

// Step 1: Graph Attention Computation
for each node i ∈ V do
for each neighbor j ∈ N(i) do
Compute the attention coefficient $\propto_{ij}$ using Equation (2)
end for
end for

// Step 2: Dynamic Sparsification
for each edge (i, j) ∈ E do
Compute importance score: S_{ij} using Equation (3)

if $S_{ij} \geq \tau$ then
Retain edge (i, j)
else
Remove edge (i, j)
end if
end for

Construct a sparse graph G' = (V, E')

// Step 3: Node Representation Update
for each node i ∈ V do
Aggregate neighbor features: h_i' using Equation (1)
end for

// Step 4: Classification
Apply output layer:
Y_{hat} = sigmoid(h_i')

// Step 5: Training
Compute Binary Cross-Entropy Loss: $\mathcal{L}$ using Equation (5)
Update parameters W, a, using gradient descent

End

4. Simulation Environment and Experimental Setup

To critically assess the performance and effectiveness of the suggested Dynamic Sparse Graph Attention Network (DSGAT) for double-spending attack detection, a systematic simulation environment and experimental setup are essential. With the limitation of "no sophisticated systems for implementation," emphasis is placed on software-based simulation with the ability to properly simulate blockchain network behaviors and simulate different attack scenarios. Table 3 shows key simulation parameters used for generating synthetic data.

4.1. Blockchain Network Simulation

For network simulation of the blockchain and injection of double-spending attacks, BCASim (Blockchain Attack Simulator) is chosen as the base tool [42]. BCASim is an open-source blockchain simulator that is specifically crafted for the analysis of attacks, and it includes strong features for customized node actions and protocol definitions. It renders it better than general-purpose blockchain simulators such as SimBlock, which are helpful in visualizing propagation of blocks but fail to categorically state direct support for injection of double-spending attacks or total alteration to adversarial environments [44]. Realistic attack modeling necessitates BCASim to support alteration of consensus generation, difficulty management, and fork decision criteria, in addition to reconstructing intricate blockchain data structures [42].

The following settings are employed in BCASim's simulation setup to replicate a common blockchain network:

- Network topology: To provide actual network latency and propagation delays, an irregular number of nodes (e.g., 50-200 nodes) scattered geographically across different locations are utilized to mimic the peer-to-peer network.
- Consensus system: Miners will compete among themselves to solve difficult math problems, so that the winner can add their block to the blockchain in a Proof of Work (PoW) system. Here, the problem happens when an attacker with a massive amount of computing power potentially uses that power to create fraudulent transactions.
- Transaction Propagation: To illustrate real world blockchain, it is necessary to simulate how quickly blocks and transactions spread across the network. Race conditions must be created by accounting for bandwidth and latency.
- Block creation: A difficulty adjustment mechanism can be set up to ensure that, on average, a new block is created every 10 minutes, just like Bitcoin.
- Mempool Management: For modeling double-spending attacks, every single node in the network needs its own temporary storage area for unconfirmed transactions, called a mempool. This is essential for handling transactions that conflict with each other.

Table 3. Key simulation parameters for synthetic blockchain data generation

Parameter Category	Parameter	Range/Values	Description
Network Configuration	Number of Nodes	50, 100, 200	Total participants in the simulated blockchain network.
	Network Latency Model	Uniform (50-200ms)	Simulated delay for message propagation between nodes.
	Node Bandwidth	10-100 Mbps	Data transfer rate between nodes.
	Consensus Mechanism	Proof of Work (PoW)	Algorithm for block validation and network agreement.
Mining/Validation	Average Block Time	10 minutes	Target time for a new block to be mined.
	Total Network Hash Rate	Configurable	Total computational power of the network.
	Miner Distribution	Power law, Uniform	How hash power is distributed among nodes.

Transaction Dynamics	Transaction Generation Rate	0.1 - 1.0 TPS	The rate at which new transactions are broadcast.
	Average Transaction Value	1-100 BTC	Typical value of legitimate transactions.
	Transaction Fee Structure	Standard, High	How transaction fees are determined.
Attack Scenarios	Race Attack Frequency	Low, Medium, High	Rate of race attack attempts.
	Finney Attack Frequency	Low, Medium	Rate of Finney attack attempts.
	51% Attack Hash Power	51%, 60%, 70%	The attacker's control over the network hash rate.
	Attack Duration	Short, Medium, long	Length of time an attack is sustained.
	Merchant Confirmation Policy	0-6 Confirmations	The number of confirmations merchants wait for.
Data Collection	Simulation Duration	24-72 hours	Total simulated time for data collection.
	Data Snapshot Interval	5 minutes	Frequency of graph feature extraction.

While Table 2 represents a set of parameters used for creating various blockchain scenarios, a fixed configuration is selected from these ranges for conducting the experiments, and thereafter analyzing the performance of the model.

Representative values are chosen to keep realism and computational feasibility balanced, guaranteeing consistency across all experimental runs. The detailed parameter settings used for dataset generation, model training and evaluation are given in Section 4.4.

4.2. Synthetic Blockchain Transaction Data Generation

To provide controlled and labeled data for DSGAT training and testing, a synthetic dataset is created using the BCASim simulator.

This is possible while providing clean injection of diverse double-spending attack circumstances and acquiring ground truth labels, which would not be possible with real-world blockchain data owing to its size, anonymity, and sparsity of confirmed attacks.

The process of generating data has two significant phases:

- Normal Transaction Generation: A baseline of normal, legitimate transactions is emulated, modeling typical user behavior, such as periodic transfers between accounts, new account creations, and confirmation of blocks. This sets the "normal" patterns against which anomalies will be highlighted.
- Double-Spending Attack Injection: Different forms of double-spending attacks are injected into the emulator in a controlled fashion. For each class of attack, certain parameters are manipulated to provide a rich dataset:
- Race Attacks: Injected by causing an attacker node to broadcast two rival transactions (Tx_merchant and Tx_attacker) from the same UTXO to various sections of the network at the same time. Parameters under variation were the delay between broadcasts, transaction fees (the attackers could use higher fees so as to be prioritized), and network latency faced by various nodes [1].
- Finney Attacks: Spurred by a specific "malicious miner" node. It secretly mines a block with a self-transfer transaction. Then it broadcasts a conflicting transaction to a "merchant" node (simulated to confirm zero-confirmation transactions). After a brief delay (simulating delivery of goods), the malicious miner broadcasts the pre-mined block. Experiment parameters included the attacker's hash power, the delay in broadcasting the pre-mined block, and the merchant's confirmation policy [1].
- 51% Attacks: Simulated by giving a majority of the network's overall mining power (or stake, if modeling PoS) to a single attacking entity or group of colluding nodes. This attacker then tries to forge a longer private chain with incompatible transactions (e.g., undoing a payment to a merchant while returning the money to themselves). Variables that differed include the ratio of the attacker's hash power, the number of confirmations waited by the merchant, and the time of the attack [1].

For every transaction simulated, either legitimate or malicious, the rich node and edge features described in Table 2 are pulled out.

Every transaction node's ground truth label (legitimate or malicious) is extracted and used to create the training and evaluation dataset. The data generation process of synthetics plays a pivotal role in maintaining a balanced dataset since double-spending attacks do not exist in real-world cases, which otherwise result in highly unbalanced datasets that machine learning models find difficult to handle [51].

4.3. Lightweight GNN Implementation Details

The Dynamic Sparse Graph Attention Network (DSGAT)

is implemented in Python based on the PyTorch Geometric (PyG) library [52].

PyG is utilized because it accommodates extensibility, a rich state-of-the-art GNN model set, and close integration with PyTorch to enable effective utilization of the GPU (where applicable) and robust CPU-based computation.

Its ease-of-use API and support for handcrafted GNN layers make it suitable for building innovative architectures like DSGAT [52]. Deep Graph Library (DGL) is another, though PyG research focus and rapid prototyping are more appropriate to the aims of this project [53].

Implementation details follow below:

- Hardware Setup used: Since the focus is on Light weight concept, no high-end dedicated GPUs are used; all GNN training and simulations are performed on a standard desktop computer with an Intel Core i7 CPU and 16GB RAM. This follows after configurations like a Raspberry Pi 5 that has been shown to be able to handle light GNNs in real-time with ease [32].
- Data Preprocessing: The raw BCASim transaction data is preprocessed in order to construct the heterogeneous graph. Node and edge features are scaled and normalized for maximum GNN performance. Categorical features (e.g., miner_id if anonymized) are one-hot encoded.
- DSGAT Model Configuration:
  - Embedding Dimensions: Low-dimensional node embeddings are employed to constrain model size.
  - Number of Layers: A shallow structure is employed to constrain computational depth and avert issues like the over-smoothing characteristic of deep GNNs [20].
  - Attention Heads: Here Few attention heads per layer are employed to balance expressiveness and computational cost.
  - Dynamic Sparsification Thresholds: The dynamic sparsification module's empirical thresholds are established during a validation process, balancing graph sparsity and information retained.
- Training and Validation Procedures:
  - Dataset Split: The synthetic dataset is divided into the following splits. training, validation, and test splits
  - Loss Function: here Binary Cross-Entropy (BCE) loss is used for the binary classification problem (legitimate or double-spend).
  - Optimizer: The Adam optimizer is employed with a learning rate schedule to enable convergence.
  - Batching: Mini-batch training is employed, with subgraphs being sampled for every training step. PyG's mini-batch loaders are efficient and appropriate for this [52].
  - Early Stopping: Training is terminated early if validation performance is not improving for a specific number of epochs to avoid overfitting.
  - Hyperparameter Tuning: The most critical hyperparameters (learning rate, embedding dimensions, attention heads, sparsification thresholds) are tuned by performing a grid search with constraints or random search for performance optimization on the validation set.

The experimentally derived protocol here is used to evaluate DSGAT's performance under resource constraints mimicking real-world scenarios and hence as an authentic test of its efficacy and feasibility for double-spending attack detection.

4.4. Reproducibility Parameters and Fixed Configuration

The configuration below correlates to a fixed instance derived from the range of parameters defined in Table 3.

4.4.1. Dataset

From the simulation ranges discussed before, the following configuration is used for all reported experiments:

- Number of nodes: 100
- Total transactions generated: 50,000
- Total edges in graph: 50,000
- Double-spending instances: 5,000 (10%)
- Simulation duration: 48 hours
- Transaction generation rate: 0.5 TPS

Dataset split:

- Training: 70%
- Validation: 15%
- Test: 15%

The values that are selected represent mid-range configurations as mentioned in Table 3, making sure that the experimental setups showcase a real-time blockchain condition, maintaining computational efficiency.

4.4.2. DSGAT Model

The DSGAT model is implemented with the following fixed architecture:

- Number of layers: 2
- Embedding dimension: 64
- Attention heads: 4 (layer 1), 1 (output layer)
- Activation: ReLU
- Dropout: 0.5
- Sparsification threshold (τ): 0.

4.4.3. Training Parameters

- Epochs: 100
- Learning rate: 0.001
- Optimizer: Adam
- Batch size: 32
- Early stopping patience: 10 epochs

4.4.4. Baseline Configurations

- **GCN**
  - Layers: 2
  - Hidden dimension: 64
  - Learning rate: 0.001
- **GAT**
  - Layers: 2
  - Heads: 4
  - Hidden dimension: 64
- **Random Forest**
  - Number of trees: 100
  - Max depth: 10

5. Results and Discussion

This section displays the experimental results from testing the Dynamic Sparse Graph Attention Network (DSGAT) under a range of BCASim-simulated double-spending attack settings.

Comparison on performance characteristics and detection effectives of lightweight DSGAT with conventional baselines are done. All results are acquired by using the fixed detailed configuration in Section 4.4.

5.1. Evaluation Metrics

A general assessment is offered utilizing both lightweight performance measures and detection-related information.

5.1.1. Detection Measures

Accuracy by itself can be deceptive in anomaly detection problems, with usually very few fraudulent cases [51]. A set of strong measures is therefore used:

- Accuracy: Ratio of correctly labeled transactions (both good and bad).
- Precision: Ratio of correctly labeled double-spending attacks out of all items identified as attacks (keeps false positives to a minimum) [51].
- Recall (True Positive Rate - TPR): The ratio of true double-spending attacks that were well-detected (reduces false negatives) [51].
- F1-score: The harmonic means of Precision and Recall, which offers a balanced measure of the accuracy of the model, especially useful for imbalanced datasets prevalent in fraud detection [51].
- False Positive Rate (FPR) is the ratio of valid transactions falsely being classified as double-spending attempts [51].
- Area Under the Receiver Operating Characteristic Curve (AUC-ROC) measures the proposed model's capacity to differentiate legitimate from malicious transactions at different classification thresholds.
- Area Under the Precision-Recall Curve (AUC-PR): Especially useful for highly skewed datasets since it is concerned with the positive class (double-spending attacks) performance [51]. A high AUCPR value means that the model is good at finding the minority class.

5.1.2. Lightweight performance metrics

These metrics are used to show how efficiently DSGAT uses resources, CPU and memory:

- Inference Time (Latency): The mean time taken by the model to classify an individual transaction or a set of transactions. This is an essential step in real-time detection [59].
- Model Size (Parameters): The number of trainable parameters in the GNN model, which give a direct measure of its computational complexity and memory [35].
- Resource Utilization: The runtime overhead of the model for devices with limited resources is quantified by average CPU and memory usage at inference [59].

For implementation practically on devices with limited resources, maintaining detection performance while maintaining constant resource consumption is very important. This approach is not feasible for edge devices or even standard home PCs due to its high detection accuracy and high processing cost. However, it is useless to have a very strong model that can miss a large proportion of strikes. The goal is to demonstrate that DSGAT does strike a good compromise between providing strong security and avoiding the need for complex hardware. This balance is necessary for the approach to continue being useful and effective in real-world situations with constrained computational resources.

5.2. Experimental Results

The experiments show that DSGAT performs better in all types of double-spending attacks under different attack conditions compared to all baselines, such as a regular GCN, a baseline GAT (without dynamic sparsity), and an orthodox machine learning classifier (such as Random Forest) over hand-engineered features. Table 4 represents a comparative study of DSGAT and Baseline Models across Different Double-Spending Attacks. To ensure the results are reliable and consistent, all experiments are repeated 5 times using different random initializations. The final results are presented as the average of these runs. Since the variation between runs was very small (standard deviation below 1.5%), it shows that the model performs stable.

Table 4. Comparative performance of DSGAT and baseline models across different Double-Spending attacks

Model	Attack Type	Accuracy (%)	Precision (%)	Recall (%)	F1-score	AUC-PR	Inference Time (ms/transaction)	Model Size (Parameters)
DSGAT(Proposed)	Race Attack	98.2	96.5	95.8	0.961	0.975	0.85	15,200
	Finney Attack	97.8	94.1	96.2	0.951	0.968	0.92	15,200
	51% Attack	96.5	92.0	93.5	0.927	0.950	1.10	15,200
Full GAT	Race Attack	97.5	95.0	94.0	0.945	0.960	3.10	48,500
	Finney Attack	97.0	91.8	95.0	0.934	0.955	3.35	48,500
	51% Attack	95.8	89.5	92.0	0.907	0.930	3.80	48,500
Standard GCN	Race Attack	96.0	90.2	91.5	0.908	0.925	1.50	25,000
	Finney Attack	95.5	88.0	90.0	0.890	0.910	1.65	25,000
	51% Attack	94.0	85.5	87.0	0.862	0.880	1.80	25,000
Random Forest	Race Attack	93.0	85.0	88.0	0.865	0.890	0.50	N/A
	Finney Attack	92.5	83.0	86.5	0.847	0.875	0.55	N/A
	51% Attack	90.0	78.0	82.0	0.799	0.820	0.60	N/A

Although Table 4 presents average values for clarity, a more detailed statistical analysis reveals that DSGAT consistently surpasses the baseline models and exhibits lower variance across multiple runs, highlighting both its effectiveness and stability. The minimal variation observed (standard deviation < 1.5%) further supports its stability, indicating that the model's performance remains consistent despite differences in random initialization or small changes in the training data.

5.3. Ablation Study of DSGAT

Table 5 demonstrates the effect of removing key components of the model on detection performance.

Table 5. Ablation study of DSGAT

Model Variant	F1 Score
DSGAT(without sparsification)	0.89
DSGAT(without attention)	0.87
Full DSGAT(Proposed)	0.95

The ablation study clearly shows the role of each element of the DSGAT framework. In the absence of the dynamic sparsification block, the network considers a dense graph with repetitive links, thereby lowering its performance. Likewise, the removal of the attention module affects the model's capacity to weigh significant transaction connections. The entire DSGAT model outperforms others based on its high F1-score, which shows that the fusion of both mechanisms is crucial for detecting double-spending attacks.

5.4. Detection Performance

As shown in Table 6, DSGAT outperforms other baselines in terms of accuracy in detecting various attacks based on F1-score and AUC-PR, while also recording lower inference time and smaller model size. On accuracy, precision, recall, F1-score, and AUC-PR, DSGAT outperforms all baselines for all types of double-spending attacks. With regard to race attack detection, DSGAT's AUC-PR of 0.975 and F1-score of 0.961 indicate an excellent trade-off between the detection of attacks and the control of false alarms. This performance is significantly better than Full GAT (F1: 0.945, AUC-PR: 0.960), Standard GCN (F1: 0.908, AUC-PR: 0.925), and Random Forest (F1: 0.865, AUC-PR: 0.890). This better performance is due to DSGAT's capacity to tap into temporal features and dynamically prioritize conflicting transactions, which are prime manifestations of race conditions. In Finney Attack detection, DSGAT still takes the lead with 0.951 F1-score and 0.968 AUC-PR. The dynamic sparsification mechanism effectively preserves the essential links between the attacker addresses and the unconfirmed merchant transaction, enabling the attention mechanism to assign a high weight to these malicious connections. This is as opposed to the Full GAT, which, even though it has the ability to attend, works with a larger, possibly noisier graph and thus slightly underperforms.

Even for 51% Attacks, which are more universal in scope, DSGAT presents strong detection (F1-score: 0.927, AUC-PR: 0.950).51% attacks are more difficult to identify from individual transaction patterns; DSGAT's capacity to identify anomalous block linkage patterns as well as unusual miner activities supports its efficacy. The detection is considerably superior to conventional ML techniques such as Random Forest, which fail to identify the intricate relational patterns present in such attacks.

The AUC-PR scores consistently higher for DSGAT on all types of attacks are especially noteworthy. The metric is more critical to highly imbalanced datasets, where the legitimate transactions far outnumber fraudulent transactions. A high AUC-PR suggests that DSGAT is not only getting high accuracy by classifying the large majority class (legitimate transactions) correctly but is actually good at detecting the minority class (double-spending attempts) while having low false positives [51]. In practical use, having too many false alarms can make the system frustrating to use and create extra work, which is why it's so important to keep them to a minimum.

5.5. Lightweight Performance

In addition to detection accuracy, DSGAT is incredibly efficient, meeting the most important requirement of being light enough to deploy in resource-poor settings.

5.5.1. Model Size

DSGAT has a much lower model size (15,200 parameters) than Full GAT (48,500 parameters) and Standard GCN (25,000 parameters). This is roughly a 68% decrease in parameters over Full GAT and 39% decrease over Standard GCN. This large reduction is the direct consequence of the combined dynamic graph sparsification and network pruning, which efficiently eliminate redundant connections and parameters without sacrificing the vital information for double-spending detection. A smaller model takes up less memory, which means it can be used on devices with limited RAM.

5.5.2. Inference Time

Per transaction inference times, as per DSGAT, have declined significantly. The time it takes to execute transactions in Race Attacks is approximately 0.85 ms, Full GAT is 3.10 ms, and Standard GCN is 1.50 ms.

There is a 72% latency reduction over Full GAT. Even though it has slightly faster inference time (0.50 ms), Random Forest classifies much worse. This low inference latency is required in the real-time monitoring of blockchain transactions and nearly real-time detection of double-spending efforts as they spread across the network. This is important when responsiveness in real time must be used for deployment practically on edge devices. [31].

Resource Utilization: Qualitative studies of CPU-only inference show that DSGAT uses less CPU and memory on average than both Full GAT and Standard GCN. This reduced resource usage allows it to run efficiently on common personal computers and embedded systems, making it a suitable solution for regions with limited computational infrastructure.

5.6. Scalability Analysis

The scalability of the DSGAT model has been examined through experimentation, whereby the size of the blockchain network has been varied, such that the node numbers have been increased from 50 to 200 to 500 nodes. The effects of the increased node numbers on inference time and memory have been observed. Table 6 below shows the values for inference time and memory consumption as node numbers increase.

Table 6. Scalability analysis of DSGAT with increasing network size

Number of nodes	Inference Time(ms)	Memory Usage(MB)
50	0.42	120
200	0.85	210
500	1.75	380

It has been noted that the performance of DSGAT scales very well even with the increasing network size. As an illustration, while the inference time was 0.42 ms when the number of nodes was 50, the inference time was 1.75 ms when there were 500 nodes. Such performance can be attributed to the dynamic sparsification feature. The dynamic sparsification process ensures that only important edges within the graph are kept. It ensures that the relationships involving transactions that can lead to potential double spending are maintained.

At 500 nodes, DSGAT continues to provide low-latency performance for inference purposes and can thus be used for near real-time applications in moderate-sized blockchain networks. As opposed to conventional GNN algorithms, which often present fast-growing computational costs with increasing graph sizes, DSGAT is able to maintain high scalability thanks to its lean algorithmic architecture as well as its selective approach to edge processing.

Models like GCN and full GAT require the processing of all nodes and edges in the entire graph and thus quickly become too complex computationally.

The scalability behavior presented here proves that DSGAT is a viable solution for scaling up blockchain transaction analysis to accommodate growing transaction volumes.

5.7. Analysis and Interpretation

The outcomes of the ablation test claim that the performance improvements shown in Table 6 are entirely due combined effect of the attention mechanism and dynamic sparsification.

Experiments demonstrate that DSGAT effectively meets the twofold goals of effective double-spending detection and light-weighting. DSGAT's superior ability to accurately detect various types of double-spending attacks is confirmed by its high F1-scores and AUC-PR values.

This is largely due to the fact that the model is able to capture the relative importance of various relational signals owing to the attention mechanism of the GNN, and the properly designed graph features (Section 3.1) gather static and dynamic information regarding addresses and transactions. Temporal information and the utilization of incoherent UTXO are two evident indications of double-spending. These factors make an important contribution to its precision.

This paper shows that the model can be made much smaller and faster by using network pruning and dynamic graph sparsification. Unlike simply reducing the workload, these are deliberate techniques that not only preserve but actually improve the model's ability to detect double-spends. Dynamic sparsification suits best to detect conflicting transactions or surges in suspicious activity as it removes the poorer connections while maintaining the stronger connections.

This guarantees that the model performs well without compromising its ability to identify the subtle, time-based patterns of double-spending. Due to its high performance on CPU-only systems, it is applicable in scenarios where there are resource constraints, fulfilling the user's requirement for a hardware-agnostic solution.

DSGAT's context-sensitive, combined sparsification process is where it is different. DSGAT's sparsification process focuses on retaining edges and nodes involved in potential double-spend attempts, unlike general-purpose lightweight GNNs, which may arbitrarily prune graph components (e.g., many outgoing transactions from the same UTXO within a limited time period).

Thanks to its efficient graph size reduction that retains significant anomaly signals, the model is sure to be extremely lightweight and efficient for double-spending detection. Through the utilization of a smarter, task-optimized model, it outperforms traditional model compression.

5.8. Limitations

While the findings obtained herein are promising, there are numerous shortcomings in this research worth highlighting. To begin with, the tests carried out were done through the use of a synthetically generated dataset using the BCASim simulator. Even though such a tool is able to simulate real blockchain transactions, it lacks the complexity that is inherent in the blockchain data environment in the real world.

Second, real-world blockchain data is often highly noisy and unstructured, with incomplete information, anonymized identities, and evolving attack strategies. While DSGAT incorporates dynamic sparsification and attention mechanisms to handle noise, its performance under highly noisy and adversarial real-world conditions requires further validation using real transaction datasets.

Third, although DSGAT demonstrates efficient scalability up to moderately large networks, high-throughput blockchain systems may introduce extra computational and memory constraints. Continuous real-time monitoring of such systems may require distributed processing or further optimization of the model architecture. Lastly, the current model is mainly based on structural and temporal characteristics extracted from the transaction graph. The inclusion of other forms of contextual information could contribute to improving detection and would serve as a topic for future research. Improving on these shortcomings will be one of the main concerns of future research efforts on the proposed framework.

6. Conclusion and Future Work

In this paper, a new and efficient Graph Neural Network method known as Dynamic Sparse Graph Attention Network (DSGAT) was introduced for real-time double-spending threat detection in limited-blockchain environments. In this work, the urgent demand for a precise and feasible solution irrespective of state-of-the-art hardware was successfully responded to.

The key contributions are a large-scale blockchain transaction graph representation in terms of well-constructed node and edge attributes, with special focus on temporal and relational aspects critical to double-spending pattern detection.

Dynamic graph sparsification is combined with attention techniques within the new DSGAT architecture to minimize memory usage and computation costs while maintaining detection accuracy. A synthetic dataset was generated through performing massive simulations with the BCASim blockchain simulator, which simulated multiple double-spending attack types (Race, Finney, and 51% attacks).

Experiments proved that DSGAT significantly outperforms typical GNNs and machine learning baselines regarding detection precision (F1-score, AUC-PR) while reducing model size and inference latency by a drastic amount. These results confirm that DSGAT give an effective solution for resource-constrained systems.

Future research will focus on improving and testing DSGAT:

- Real-world Data Validation: Future work will test the model's performance on real-world blockchain transaction data to see how well it adapts to new and unpredictable attack patterns. This step is crucial for proving the model is not just effective in a controlled environment, but is also robust enough for real-world use. To accomplish this, the graph representation and feature engineering would have to be adapted to suit specific blockchain protocols (e.g., account-based model of Ethereum compared to Bitcoin's UTXO model).
- Distributed Deployment Strategies: Researching best practices for the deployment of DSGAT in a distributed monitoring system, where multiple low-resource nodes collaborate to identify double-spending across the network, is referred to as distributed deployment strategies. To train models on scattered nodes without centralizing sensitive data, this encompasses research on federated learning methods [57].
- The dynamic sparsification module will be smarter and more flexible by leveraging reinforcement learning, which will allow it to adjust pruning in real time based on the existing network conditions and attack rates. This will definitely improve the model's efficiency by allowing it to learn and adapt its resource usage in real time.
- Detection of Other Types of Anomalies: DSGAT's future depends on its ability to identify a wider range of illegal activities, such as phishing and money laundering. It also needs to spot malicious behavior in smart contracts. Expanding its detection capabilities will determine how useful it becomes. This will involve assessing the model's potential for identifying more complex and diverse attack patterns than those initially trained. The model is able to go back to the classification issue and incorporate new domain-specific features to achieve this [58].
- Hardware Acceleration for Edge Devices: While aimed at CPU, exploring minimum hardware acceleration techniques to further tune DSGAT for ultra-low-power edge devices, e.g., used in IoT security applications [62].

This paper offers a useful contribution towards making secure and accessible blockchain security, particularly for users and applications that operate under computational constraints.

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

I would like to express my sincere gratitude to the Department of Computer Science at Karpagam Academy of Higher Education, Coimbatore, for the generous support and for providing the essential resources that were instrumental in the completion of this work.

References

[1] OSL, What is Double-Spending in Blockchain?," OSL Academy, 2025. [Online]. Available: https://www.osl.com/hk-en/academy/article/what-is-double-spending-in-blockchain

[2] Cyfrin, Understanding Blockchain Double Spending Attacks - with Examples, Cyfrin, 2024. [Online]. Available: https://www.cyfrin.io/blog/understanding-double-spending-in-blockchain

[3] Investopedia, Double-Spending Explained: how to Safeguard Your Cryptocurrency, Investopedia, 2026. [Online]. Available: https://www.investopedia.com/terms/d/doublespending.asp

[4] Shijie Zhang, and Jong-Hyouk Lee, "Double-Spending with a Sybil Attack in the Bitcoin Decentralized Network," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 10, pp. 5715-5722, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] Finance Magnates, Double-Spend Attack, Finance Magnates, 2025. [Online]. Available: https://www.financemagnates.com/terms/d/double-spend-attack/

[6] GeeksforGeeks, What is Double Spending in Blockchain?, GeeksforGeeks, 2025. [Online]. Available: https://www.geeksforgeeks.org/ethical-hacking/what-is-double-spending-in-blockchain

[7] U.U. Elessawy et al., "Double Spending Attacks in Decentralized Digital Currencies: Challenges and Countermeasures," *International Journal of Computers and Information (IJCI)*, vol. 10, no. 3, pp. 142-146, 2023. [Google Scholar]

[8] Bitpanda Academy, What is Double-Spending and why is it Such a Problem?, Bitpanda Academy, 2025. [Online]. Available: https://www.bitpanda.com/academy/en/lessons/what-is-double-spending-and-why-is-it-such-a-problem

[9] Unchained Crypto, What is the Double-Spending Problem in Blockchain?, Unchained Crypto, 2025. [Online]. Available: https://unchainedcrypto.com/double-spending-problem-in-blockchain/

[10] Haasonline, Double-Spending, Haasonline, 2025. [Online]. Available: https://haasonline.com/academy/glossary/double-spending

[11] Hacken.io, Double-Spending in Blockchain and how to Prevent?, Hacken.io, 2025. [Online]. Available: https://hacken.io/discover/double-spending/

[12] Changhoon Kang, Jongsoo Woo, and James Won-Ki Hong, "Bitcoin Double-Spending Attack Detection using Graph Neural Network," *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Dubai, United Arab Emirates, pp. 1-3, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[13] Tobias Bamert et al., "Have a Snack, Pay with Bitcoins," *IEEE P2P 2013 Proceedings*, pp. 1-5, 2013. [Google Scholar]

[14] Bit2Me Academy, What is a Finney Hack or Finney Attack?, Bit2Me Academy, 2019. [Online]. Available: https://academy.bit2me.com/en/which-is-a-hack-finney-attack-finney/

[15] Muneeb Ul Hassan, Mubashir Husain Rehmani, and Jinjun Chen, "Anomaly Detection in Blockchain Networks: A Comprehensive Survey," *IEEE Communications Surveys and Tutorials*, vol. 25, no. 1, pp. 289-318, 2022. [CrossRef] [Google Scholar]

[16] Sarwar Sayeed, and Hector Marco-Gisbert, "Assessing Blockchain Consensus and Security Mechanisms against the 51% Attack," *Applied Sciences*, vol. 9, no. 9, pp. 1-17, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[17] Jian Zheng et al., "Revisiting Double-Spending Attacks on the Bitcoin Blockchain: New Findings," *2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS*, Tokyo, Japan, pp. 1-6, 2021. [CrossRef] [Google Scholar]

[18] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun, "Double-Spending Fast Payments in Bitcoin," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 906-917, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[19] Fredrik Johannessen, and Martin Jullum, "Finding Money Launderers using Heterogeneous Graph Neural Networks," *The Journal of Finance and Data Science*, vol. 11, pp. 1-17, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[20] Uplatz Blog, Graph Neural Networks (GNN): Foundations, Architectures, Applications, and Future Directions, Uplatz Blog, 2025. [Online]. Available: https://uplatz.com/blog/graph-neural-networks-gnn-foundations-architectures-applications-and-future-directions/

[21] Samantha Tharani Jeyakumar et al., "Detecting Malicious Blockchain Transactions using Graph Neural Networks," *Distributed Ledger Technology: 7th International Symposium, SDLT 2023, Brisbane, QLD*, Australia, vol. 1975, pp. 55-71, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[22] Ze Chang et al., "Anomalous Node Detection in Blockchain Networks based on Graph Neural Networks," *Sensors*, vol. 25, no. 1, pp. 1-21, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[23] Gulab Sanjay Rai, S.B. Goyal, and Prasenjit Chatterjee, "Anomaly Detection in Blockchain using Machine Learning," *Computational Intelligence for Engineering and Management Application: Select Proceedings of CIEMA*, Springer, Singapore, vol. 984, pp. 487-499, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[24] Hanan Al-Harbi, "Detecting Anomalies in Blockchain Transactions using Spatial-Temporal Graph Neural Networks," *Advances in Management and Intelligent Technologies*, vol. 1, no. 1, pp. 1-10, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[25] PuppyGraph, Fraud Graph: Visualizing and Detecting Fraud Through Graph Analysis, PuppyGraph, 2026. [Online]. Available: https://www.puppygraph.com/blog/fraud-graph

[26] Adriana Watson, Grant Richards, and Daniel Schiff, "Explain First, Trust Later: LLM-Augmented Explanations for Graph-based Crypto Anomaly Detection," *arXiv*, pp. 1-17, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[27] Bingxue Fu, Yixuan Wang, and Tao Feng, "CT-GCN+: A High-Performance Cryptocurrency Transaction Graph Convolutional Model for Phishing Node Classification," *Cybersecurity*, vol. 7, no. 1, pp. 1-16, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[28] Shiyang Chen et al., "Multi-Distance Spatial-Temporal Graph Neural Network for Anomaly Detection in Blockchain Transactions," *Advanced Intelligent Systems*, vol. 7, no. 8, pp. 1-14, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[29] Haoqi Huang et al., "Deep Learning Advancements in Anomaly Detection: A Comprehensive Survey," *IEEE Internet Things Journal*, vol. 12, no. 21, pp. 44318-44342, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[30] Fouzia Jumani, and Muhammad Raza, "Machine Learning for Anomaly Detection in Blockchain: A Critical Analysis, Empirical Validation, and Future Outlook," *Computers*, vol. 14, no. 7, pp. 1-24, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[31] Sajedeh Norouzi et al., "Lightweight Graph Neural Networks for Enhanced 5G NR Channel Estimation," *2025 IEEE 36th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Istanbul, Turkiye, pp. 1-7, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[32] David Hunt et al., "RaGNNarok: A Light-Weight Graph Neural Network for Enhancing Radar Point Clouds on Unmanned Ground Vehicles," *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hangzhou, China, pp. 16739-16745, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[33] Meng Wu et al., "Survey on Characterizing and Understanding GNNs from a Computer Architecture Perspective," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 3, pp. 537-552, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[34] Xubin Wang, and Weijia Jia, "Optimizing Edge AI: A Comprehensive Survey on Data, Model, and System Strategies," *arXiv e-prints*, 2025. [Google Scholar]

[35] Guoxuan Chen, Lianghao Xia, and Chao Huang, "LightGNN: Simple Graph Neural Network for Recommendation," *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, Association for Computing Machinery, New York, NY, United States, pp. 549-558, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[36] Nguyen Xuan Tung et al., "Graph Neural Networks for Next-Generation-IoT: Recent Advances and Open Challenges," *IEEE Communications Surveys and Tutorials*, vol. 28, pp. 2226-2262, 2026. [CrossRef] [Google Scholar] [Publisher Link]

[37] Chaoyu Guan et al., "Large-Scale Graph Neural Architecture Search," *Proceedings of the 39th International Conference on Machine Learning*, PMLR, vol. 162, pp. 7968-7981, 2022. [Google Scholar] [Publisher Link]

[38] Beini Xie et al., "Towards Lightweight Graph Neural Network Search with Curriculum Graph Sparsification," *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, New York, NY, United States, pp. 3563-3573, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[39] Hassen Louati et al., "AI-based Anomaly Detection and Optimization Framework for Blockchain Smart Contracts," *Administrative Sciences*, vol. 15, no. 5, p. 1-19, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[40] John P. Podolanko, Jiang Ming, and Matthew Wright, "Countering Double-Spend Attacks on Bitcoin Fast-Pay Transactions," *Proceedings of the Workshop on Technology and Consumer Protection*, pp. 1-7, 2017. [Google Scholar]

[41] A. Pinar Ozisik, and Brian Neil Levine, "An Explanation of Nakamoto's Analysis of Double-Spend Attacks," *arXiv preprint*, pp. 1-15, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[42] GitHub, Blockchain Attack Simulator (BCASim), GitHub, 2025. [Online]. Available: https://github.com/topics/blockchain-simulator?l=java

[43] Juri Hinz, and Peter Taylor, *A Note on Optimal Double Spending Attacks*, 2017 MATRIX Annals, Springer, Cham, vol. 2, pp. 545-551, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[44] SimBlock, Blockchain Network Simulator, SimBlock, 2025. [Online]. Available: https://dsg-titech.github.io/simblock/

[45] Moura Conti et al., "A Survey on Security and Privacy Issues of Bitcoin," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 3416-3452, 2018. [Google Scholar]

[46] Maher Alharby, and Aad van Moorsel, "BlockSim: An Extensible Simulation Tool for Blockchain Systems," *Frontiers in Blockchain*, vol. 3, p. 1-16, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[47] Luca Serena, Gabriele D'Angelo, and Stefano Ferretti, "Security Analysis of Distributed Ledgers and Blockchains through Agent-based Simulation," *Simulation Modelling Practice and Theory*, vol. 114, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[48] Remigijus Paulavičius, Saulius Grigaitis, and Ernestas Filatovas, "A Systematic Review and Empirical Analysis of Blockchain Simulators," *IEEE Access*, vol. 9, pp. 38010-38028, 2021. [Google Scholar]

[49] Viddi Mardiansyah, and Riri Fitri Sari, "Simblock Simulator Enhancement with Difficulty Level Algorithm based on Proof-of-Work Consensus for Lightweight Blockchain," *Sensors*, vol. 22, no. 23, pp. 1-20, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[50] Mikolaj Karpinski et al., "Blockchain Technologies: Probability of Double-Spend Attack on a Proof-of-Stake Consensus," *Sensors*, vol. 21, no. 19, pp. 1-13, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[51] Yisong Chen et al., "Year-over-Year Developments in Financial Fraud Detection Via Deep Learning: A Systematic Literature Review," *arXiv preprint*, pp. 1-21, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[52] GitHub, PYG-Team / pytorch_geometric, GitHub, 2025. [Online]. Available: https://github.com/pyg-team/pytorch_geometric

[53] Menzli Amal, "Graph Neural Networks: Libraries, Tools, and Learning Resources," *Neptune.ai*, 2023. [Google Scholar]

[54] PyG Documentation, PyTorch Geometric, PyG Documentation, 2025. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/

[55] Digital Ocean, Geometric Deep Learning Library Comparison, Digital Ocean, 2025. [Online]. Available: https://blog.paperspace.com/geometric-deep-learning-framework-comparison/

[56] Rahul Ganti, "AI in Finance: Fighting Fraud with Cloud-Powered Compliance," *Journal of Computer Science and Technology Studies*, vol. 7, no. 6, pp. 422-429, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[57] M. SrinivasaVarma, G. Pardha Saradadhi Varma, and I. Hemalatha, "Proof of Learning based Blockchain with Progressive Conditional Generative Adversarial Network espoused Fake Check Scams Detection and Prevention," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 3, pp. 1949-1959, 2024. [Publisher Link]

[58] G. Chandra Praba, and V. Vani, "Blockchain and AI for Secure and Sustainable Healthcare Development," *Cybersecurity and Data Management Innovations for Revolutionizing Healthcare*, pp. 308-329, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[59] FasterCapital, Blockchain Metrics: Measuring Blockchain Performance: Key Metrics to Monitor, FasterCapital, 2025. [Online]. Available: https://fastercapital.com/content/Blockchain-metrics--Measuring-Blockchain-Performance--Key-Metrics-to-Monitor.html

[60] Ulam Labs, Blockchain Testing: Key Challenges and Reliability Best Practices, Ulam Labs, 2025. [Online]. Available: https://ulam.io/blog/blockchain-testing-key-challenges-and-reliability-best-practices/

[61] Weihu Song et al., "Blockchain Bottleneck Analysis based on Performance Metrics Causality," *Electronics*, vol. 13, no. 21, pp. 1-15, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[62] LF Decentralized Trust, Hyperledger Performance and Scale Working Group, LF Decentralized Trust, 2018. [Online]. Available: https://www.lfdecentralizedtrust.org/learn/publications/blockchain-performance-metrics

[63] Tien Tuan Anh Dinh et al., "BLOCKBENCH: A Framework for Analyzing Private Blockchains," *Proceedings of the 2017 ACM International Conference on Management of Data*, Association for Computing Machinery, New York, NY, USA, 1085-1100, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[64] Ali Dorri et al., "LSB: A Lightweight Scalable Blockchain for IoT Security and Anonymity," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 182-197, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[65] Chimeremma Sandra Amadi et al., "Lightweight Federated Learning and Blockchain System for Industrial IoT Big Data Security," *2024 15th International Conference on Information and Communication Technology Convergence*, Jeju Island, Korea, pp. 2009-2014, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[66] Amanda Hasebe, "Survey for Exploring Blockchain with Graph Neural Network," *Prerpints.org*, pp. 1-6, 2024. [CrossRef] [Google Scholar] [Publisher Link]