Scholar Publishing Group
Socio-Economic Statistics Research
https://doi.org/10.38007/SESR.2026.070201
ISSN 2790-2722 Vol. 7, Issue 2: 1-9

Vulnerability Detection and Formal Verification Methods of Blockchain Smart Contracts in Supply Chain Finance

Jiayi Sun

Zhifang Design Co., Ltd, 430000, Hubei, China

Keywords: Supply chain finance; smart contracts; vulnerability detection; graph neural networks; formal verification

Abstract: Supply chain finance smart contracts incorporate rules for order confirmation, financing disbursement, performance verification, and fund settlement into on-chain code. This approach shortens the time for information confirmation and fund processing among multiple parties, but also makes fund security more dependent on the correctness of the code logic. Rule-based static testing struggles to identify cross-function state errors, while individual model checks are susceptible to state space expansion. To address these issues, this study proposes a collaborative method combining graph neural network initial screening and formal verification. Taking order financing as an example, the study constructs a state machine for "order registration—confirmation—credit granting—disbursement—performance—settlement," and extracts core constraints such as fund conservation, role-based permissions, and duplicate withdrawals. The system integrates abstract syntax trees, control flow, data dependencies, and call relationships at the code layer to locate high-risk functions; the system then selects computation tree logic reductions based on the code structure pattern and uses NuSMV to verify key fund paths. The study also elucidates the screening rules for publicly available data sources and real-world deployed contracts, as well as a layered comparison scheme, providing a reproducible research path for pre-deployment auditing of supply chain finance smart contracts.

1. Introduction

Supply chain finance provides financial support to upstream and downstream enterprises based on orders, accounts receivable, logistics vouchers and core enterprise credit. Blockchain technology provides a verifiable business foundation for multi-entity collaboration in accounts receivable financing through shared transaction records and automatic triggering mechanisms [1]. Existing research indicates that information visibility, risk sharing and platform rules will jointly affect the operational efficiency of supply chain finance [2]. Such contracts manage the amount of funds, term conditions and role permissions at the same time. If the contract is locked only after being called externally, or allows unauthorized entities to modify credit parameters, a single code defect may lead to duplicate withdrawals, duplicate financing or abnormal liquidation. Full-chain analysis of deployed Ethereum contracts shows that increasing the number of contract deployments does not

automatically reduce the risk of vulnerabilities. Therefore, financial contracts still need to undergo targeted security checks before going live [3].

Existing detection tools mainly employ rule matching, static analysis, symbolic execution, and learning-based detection. Relevant reviews indicate that different methods have different focuses in terms of coverage, false alarm control, and complex state reasoning capabilities, and a single tool is difficult to meet the detection needs of complex financial contracts simultaneously [4]. Formal verification can constrain state transitions with explicit specifications, but there is still a coverage gap between security design patterns and known vulnerabilities, and business logic defects cannot be eliminated solely by relying on general code patterns [5]. Based on this problem, this study connects code graph learning and formal verification into a complete link: the graph model locates high-risk functions and their categories, and the formal module verifies business specifications directly related to fund security.

This study focuses on the fund security issue of smart contracts in supply chain finance. The study constructs a state machine for order financing, preserving the necessary states and transition conditions for fund security. It establishes a clear mapping between "code structure pattern—business variables—CTL specification," enabling code-level detection results to enter the business-level verification stage. The study also uses a publicly available row-level labeled dataset to evaluate code detection capabilities and constructs an independent case set using real-world deployed contracts, conducting a hierarchical comparison of pure graph models, pure model checking, and collaborative methods.

2. Supply Chain Finance State Machine and Core Specifications

2.1 Order Financing State Machine

This study denotes the set of order financing states as $S = \{s_0, s_1, \ldots, s_6, s_f\}$. The states in the set sequentially represent order registration, order confirmation, credit approval, loan disbursement, production fulfillment, logistics verification, fund settlement, and abnormal termination. The study denotes the set of business events as E, and the state transition function as δ. Therefore, the order financing process can be represented as:

$$\mathcal{M} = (S, s_0, E, \delta)$$

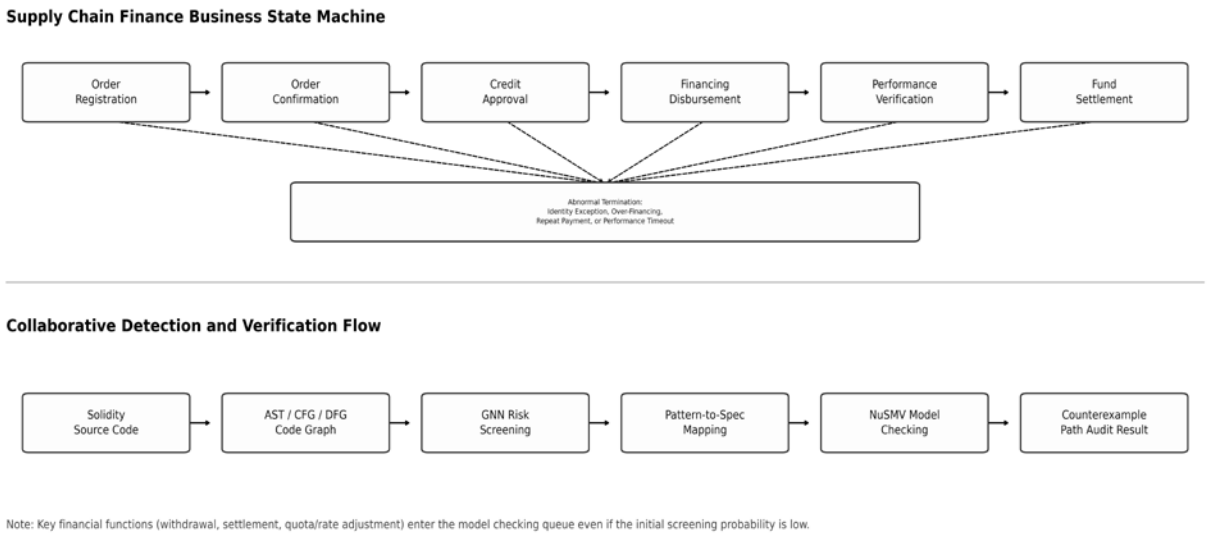

Figure 1. Supply Chain Finance State Machine and Collaborative Detection Process

The state machine stipulates that an order can only enter the confirmation state when the core enterprise and supplier identities are valid and the order information is complete. After order confirmation, financial institutions complete credit granting and loan disbursement based on available credit lines. The system simultaneously verifies performance and payment status before settlement. Over-financing, duplicate payments, role overreach, and performance timeouts will cause the business to enter an abnormal termination state. This model does not elaborate on logistics details and credit scoring mechanisms, but focuses on describing the state constraints required for fund disbursement and settlement. Figure 1 illustrates the supply chain finance state machine and collaborative detection process.

2.2 Funding and Authority Regulations

This study $\mathrm{balance_i(t)}$defines the balance of a node iat time t tas and $\mathrm{balance_e(t)}$the contract custody balance as . During a business cycle without external funding injections or withdrawals, the total funds should satisfy the following:

$$\sum_{\mathrm{i}\in\mathrm{V}} \mathrm{b\,alance_i(t)} + \mathrm{balance_e(t)} = \mathrm{C_0}$$

This $\mathrm{C_0}$represents the total amount of funds after business initialization. This regulation is used to identify risks such as duplicate deductions, duplicate payments, and abnormal balances.

This study $\mathrm{Perm(f)}$defines the set of legal roles for functions fand $\mathrm{role(u)}$defines the role of the caller u. Functions involving withdrawals, settlements, limit or interest rate modifications should satisfy the following:

$$\mathrm{AG\big(Call(u,f) \rightarrow role(u) \in Perm(f)\big)}$$

Formula (3) requires that key financial functions only accept calls from pre-authorized entities. The contract should close withdrawal permissions for the same financing number after payment is completed. Before settlement, the contract should also confirm that the order has passed performance verification and that the corresponding payment has been received.

3. A Collaborative Approach Between Graph Neural Networks and Formal Verification

3.1 Code Graph Construction and Initial Risk Screening

This study represents the contract code as a heterogeneous graph. The nodes in the graph include functions, variables, statements and modifiers, and the edge relationships include abstract syntax trees, control flow, call relationships and data dependencies. The node features simultaneously retain syntax, control, call and business-sensitive markers. Words such as order, financing, payment and settlement are only used as auxiliary semantic markers to locate fund-related functions, and cannot be used alone to determine that the contract belongs to supply chain finance business. The fusion of AST, CFG and DFG can retain code relationships across statements and functions, providing a structural basis for multi-granularity vulnerability location [6].

In lthe message propagation at the level, the node vrepresentation is updated as follows:

$$h_v^{(l+1)} = \sigma\left(W_0^{(l)} h_v^{(l)} + \sum_{u\in N(v)} \alpha_{uv}^{(l)} W^{(l)} h_u^{(l)} \right).$$

$$\alpha_{uv}^{(l)} = \frac{\exp\left(e_{uv}^{(l)}\right)}{\sum_{w \in N(v)} \exp\left(e_{wv}^{(l)}\right)}, \qquad \sum_{u \in N(v)} \alpha_{uv}^{(l)} = 1.$$

In the formula, N(v) represents the neighborhood of node v, $e_{uv}$and represents the attention score of the node pair. The model uses Softmax to normalize the attention weights so that the sum of the weights in the same neighborhood is 1. This processing can reduce the impact of different node connection numbers. Studies on dual attention graph neural networks have shown that jointly modeling node semantics and relational features can improve the identification ability of complex vulnerability patterns [7]. The model outputs the probability of risk categories such as reentrancy, permission loss, state omission, and abnormal amount processing. Functions such as withdrawal, liquidation, limit modification, and guarantee release enter the formal verification queue even if the risk probability does not exceed the threshold.

3.2 Schema-Reduction Mapping and Model Check

If code detection only outputs the risk probability, business verification will find it difficult to determine which specification to choose. To solve this problem, this study maps high-risk structures to the corresponding set of verification specifications. Existing studies have improved vulnerability identification capabilities by using source code, bytecode and control flow features in combination, which also shows that a single representation is difficult to cover the complete semantics of contract behavior [8]. Based on this, this study establishes a clear correspondence between high-risk structures and funding variables, state slices and CTL specifications. Table 1 lists the mapping from code structure patterns to business specifications.

$$\Psi(r)=\{\text{BusinessVariables},\text{StateSlicing},\text{CTLSpecification}\}$$

Table 1. Mapping from Code Structure Patterns to Business Specifications

Code structure pattern	Key business variables	State slice	CTL Specification
External calls precede the writing of payment status.	paid, withdrawable, withdrawCount	Withdrawal function and callback path	AG(Withdraw → AX(paid ∧ ¬withdrawable))
Key parameter writing lacks role restrictions	role, rate, quota, orderAmount	Parameter modification functions and modifiers	AG(Call(u,f) → role(u) ∈ Perm(f))
The liquidation function lacks performance/repayment preconditions.	DeliveryVerified, Repaid, Settled	Clearing functions and order status	AG(Settled → DeliveryVerified ∧ Repaid)
The amount of financing is not subject to quota restrictions.	amount, quota, balance	Loan disbursement function and amount dependency	AG (Fund → amount ≤ quota)

Taking the risk of duplicate withdrawals as an example, if a function has a structure where "an external low-level call is executed before the payment status is written," the model will extract variables such as paid[id] and withdrawCount[id], withdrawable[id] and select the following specification:

$$\mathrm{AG}\big(\mathrm{Withdraw}(\mathrm{id}) \rightarrow \mathrm{AX}(\mathrm{paid}[\mathrm{id}] \wedge \neg \mathrm{withdrawable}[\mathrm{id}])\big).$$

NuSMV encodes state variables, function preconditions, and permission judgments into a finite state system. Model checking methods have been used to verify the dynamic behavior and business logic of composite smart contracts, focusing on unifying contract calls, user behavior, and security

attributes into computable state transition relationships [9]. Formal studies of loan management systems also show that role permissions, balance changes, and state closure conditions can be written into executable specifications [10]. When the specification is not valid, the system outputs a counterexample path from the initial state to the violation state. To control verification overhead, the framework only performs program slicing on high-risk functions and retains variables such as order status, payment flag, role permissions, balance range, and withdrawal count. The system divides the amount variable into three ranges: "zero, normal, and excess". It should be noted here that the reduction of the state space comes from function slicing and variable abstraction, rather than the graph neural network itself.

4. Data Sources and Experimental Design

4.1 Public Data and Real-World Deployment Case Studies

Code-level vulnerability detection uses 2,182 line-level manually annotated Solidity instances released by Salzano et al. as the main baseline [11]. This dataset provides vulnerability location and category information, which is suitable for evaluating the model's ability to identify code defects such as reentrancy, access control, dangerous external calls, and state errors. The study divides the dataset by contract family or project level, with the training set, validation set, and test set containing 1,528, 218, and 436 instances, respectively. The test set contains 184 risky instances and 252 normal instances. This division avoids highly similar contracts from entering the training set and test set at the same time. The study uses the OpenSCV classification system to uniformly map vulnerability categories [12].

The study also screened independent case studies from publicly verified Ethereum source code and deployment records to test the applicability of the method in real-world deployment environments. Only compileable contracts with version information were retained. Contracts were required to include order, accounts receivable, or payment semantics, as well as key functions such as withdrawal, liquidation, or approval. Contracts also needed explicit role modifiers or permission checks, and pure token, game, NFT, and generic contracts without business state were excluded through manual review. These rules resulted in an independent case study set of 100 candidate contracts, with 40 cases of duplicate withdrawals, 30 cases of unauthorized adjustments, and 30 cases of premature liquidation due to non-performance. The study uniformly registered the contract addresses, compiled versions, selection reasons, and state machine extraction results for subsequent verification of sample sources and model extraction processes.

4.2 Fair Comparison Design and Evaluation Indicators

The experiment divides the task into two levels: code-level inspection and business logic verification. Code-level inspection compares the precision, recall, and F1-score of Slither, Mythril, pure GNN, and collaborative frameworks on the same test set. Business logic verification compares the reachable state count, verification time, and counterexample generation time of pure model checking and "GNN slicing + model checking" on the same batch of candidate cases. The study was run on Ubuntu 22.04, Python 3.10, PyTorch 2.1, Solidity 0.8.20, and NuSMV 2.7, with an AMD Ryzen 9 7950X processor, RTX 4090 graphics card, and 64 GB of RAM. Slither and Mythril are not included in the state space overhead comparison, and pure model checking is not included in the general code classification metric comparison. This hierarchical design avoids direct comparison between different tasks.

Recall and F1 score are defined as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad \text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Recall represents the model's ability to detect high-risk functions, while F1-score represents the balance between detection coverage and false positive control. Supply chain finance contracts directly handle fund operations, so code-level detection needs to control false negatives; business logic verification uses the accessibility of the violation path as the final criterion.

5. Experimental Results and Analysis

5.1 Code-level inspection results

Table 2 summarizes the comparative metrics under unified testing conditions. The test set contains 436 instances. The collaborative framework identified 169 risky instances and missed 15. The framework also generated 27 false positives and correctly identified 225 normal instances, resulting in an accuracy of 90.37%. This study does not use accuracy as the sole criterion but rather combines recall, F1-score, and coverage of different vulnerability types for evaluation. For pre-deployment auditing, the model's ability to identify high-risk functions related to funding is more practically significant than simply improving the overall classification accuracy. Table 2 presents the results of code-level detection and business logic verification.

Table 2. Results of Code-Level Inspection and Business Logic Verification

level	method	Precision	Recall/Path Coverage	F1 score	Number of reachable states	Time/s
Code-level inspection	Slither	67.15%	50.00%	57.32%	—	1.34
Code-level inspection	Mythril	71.33%	55.43%	62.39%	—	42.76
Code-level inspection	Pure GNN	80.30%	88.59%	84.24%	—	1.82
Code-level inspection	GNN+ model check	86.22%	91.85%	88.95%	—	5.96
Business logic verification	Pure model check	—	100.00%	—	1 267 584	96.40 / 2.71
Business logic verification	GNN slicing + model checking	—	100.00%	—	84 672	10.80 / 1.06
illustrate	Complete testing of high-risk samples	—	—	—	—	14.72

Note: Code-level testing was performed on 436 test instances; business logic verification was performed on 100 candidate cases. The "Verification/Negative Example" column in the time column represents the model check time and negative example generation time, respectively.

From the code-level detection results, Slither and Mythril can quickly detect some known code patterns, but their recall rates for cross-function state dependency issues are 50.00% and 55.43%, respectively. Pure GNN improves recall to 88.59% by learning control flow and data dependencies, but the model still generates 40 false positives. The collaborative framework feeds high-risk functions and key funding functions into the model's inspection queue, increasing recall to 91.85%, achieving an F1-score of 88.95%, and reducing false positives to 27. This change indicates that graph models can broaden the scope of risk detection, and specification verification can provide secondary confirmation of funding path-related alerts. For auditors, subsequent checks can focus on

a small number of interpretable high-risk paths, rather than having to review all code alerts item by item.

5.2 Business Logic Case Studies and Counterexamples

This study sets up three types of cases in the business logic layer: duplicate withdrawals, unauthorized adjustments, and premature liquidation due to non-performance. Among 100 candidate cases, both pure model checking and collaborative methods found 100 violation paths, with a path coverage rate of 100%. Taking duplicate withdrawals as an example, when an external transfer occurs paid[id]before the update, the attacking contract may trigger another withdrawal before the state lock. When formula (7) is not true, NuSMV will output a counterexample state sequence of "initial unpaid - first withdrawal - withdrawal again before external call returns - withdrawal count exceeds 1". This path can illustrate the violation conditions and triggering order, and provide a direct basis for developers to adjust the state write position, add reentrant locks, or limit the number of withdrawals.

For unauthorized adjustment cases, the validator checks whether non-financial institution nodes can modify rateor quota. For early liquidation cases, the validator checks Settledwhether it is possible for such cases to occur when DeliveryVerifiedboth and RepaymentCompletedare false. The counterexample generation times for the three types of cases—duplicate withdrawals, unauthorized adjustments, and early liquidation—are 0.94 s, 1.03 s, and 1.21 s, respectively. All three types of results correspond to the mapping table in Section 3.

5.3 Collaborative Validation Overhead

Pure model checking directly models all functions and variables in the same case, while "GNN slicing + model checking" only retains functions and data dependencies related to risk patterns. The two methods were compared on the same 100 candidate cases in terms of reachable states and validation time. Pure model checking retained an average of 17 state variables, resulting in 1,267,584 reachable states and an average validation time of 96.40 s; the collaborative method retained 9 state variables, reducing the number of reachable states to 84,672 and the average validation time to 10.80 s, achieving a state compression rate of 93.32%. Both methods captured 100 violation paths, demonstrating that function slicing and variable abstraction can alleviate state space expansion without reducing path coverage. In the test set, 296 ordinary samples underwent only initial graph model screening, while 140 high-risk or critical funding samples entered the model checking queue. Based on this processing flow, the average end-to-end detection time for the entire sample was 5.96 s, and the complete detection time for high-risk samples was 14.72 s. This timeframe is more suitable for pre-launch audits and version update checks than for directly embedding millisecond-level transaction execution processes.

6. Conclusions and Outlook

This study proposes a collaborative method for initial screening and formal verification of smart contracts in supply chain finance using graph neural networks. Based on an order financing state machine, the study retains core constraints such as capital conservation, access control, and duplicate withdrawals. Graph neural networks locate high-risk code structures, and pattern-specification mapping transforms code anomalies into explicit financial security objectives. NuSMV then verifies the reachability of violating states and generates counterexample paths. Under unified testing conditions, the collaborative framework achieves a recall rate of 91.85% and an F1-score of 88.95%, with an average reduction of 93.32% in the number of reachable states for

candidate cases. These results demonstrate that this method reduces reliance on individual model probabilities and avoids the high overhead of directly performing model checks on all contracts.

Future research should publicly disclose the addresses of real-world deployment case studies, state machine extraction records, and complete runtime logs, and continue to address scenarios such as cross-contract calls, cross-chain interactions, external oracles, and dynamic interest rate adjustments. The research also needs to establish hierarchical state abstractions around multi-contract dependencies to improve the applicability of the method in practical supply chain finance platforms.

References

[1] *Zhu, P. (2025, December). Construction of Multi-Scale Biostatistical Analysis Framework and its Application in Biomedical Signal Feature Recognition and Classification. In 2025 5th International Conference on Mobile Networks and Wireless Communications (ICMNWC) (pp. 1-7). IEEE.*

[2] *Gao, Y. (2026). Research on the Design of Governance Structure for Private Equity Funds and the balance of GP and LP Rights.*

[3] *Chang, C. W. (2026). Privacy Infrastructure Vulnerability Mining and Automated Framework Based on Multimodal AI. Procedia Computer Science, 279, 702-711.*

[4] *Chen, M. (2026). Real-Time Compliance Monitoring Engine Based on eBPF: Privacy-Enabled Data Tracking for Edge Computing. Procedia Computer Science, 281, 216-225.*

[5] *Wang, Y. (2026). Construction of Supply Chain Demand Forecasting Algorithm Based On Time Series Data Analysis and Improvement of Its Management Efficiency. Procedia Computer Science, 281, 484-493.*

[6] *Yu, X. (2026). Application of Time Series Data Analysis Algorithm Combined with Attention Mechanism in Growth Marketing User Lifetime Value Prediction. Procedia Computer Science, 281, 57-64.*

[7] *Wang, Z. (2026). Mineral Commodity Time-Series Cycle Modeling and Feature Learning Algorithm Based On Improved Transformer. Procedia Computer Science, 281, 1347-1356.*

[8] *Wang, Z. (2026). Relationship between Supply–Demand Structures of Base Metals and the Evolution of Corporate Value.*

[9] *Chen, J. (2026). Construction of a Cloud-Native High-Performance Service Engineering System for Real-Time Decision-Making Platforms.*

[10] *Chen, J. (2026). Elastic Scaling and Stability Assurance Mechanisms for Distributed Systems under High-Throughput Scenarios.*

[11] *Qian, X. (2026). Supply Chain Collaboration Mechanisms Driven by Demand Orchestration in Omni-Channel Retail Environments.*

[12] *Liang, Q. (2026). How Architectural Design and Utility Infrastructure Impacts AI Supporting Campus and Drive Future Innovation, Operational Efficiency and Sustainable Advancement in Utility-Critical Environment. European Journal of Engineering and Technologies, 2(2), 71-77.*

[13] *Wang, Y. (2025). Application of Data Completion and Full Lifecycle Cost Optimization Integrating Artificial Intelligence in Supply Chain.*

[14] *Sun, J. (2025). Quantile Regression Study on the Impact of Investor Sentiment on Financial Credit from the Perspective of Behavioral Finance.*

[15] *Chen, M. (2025). Research on Automated Risk Detection Methods in Machine Learning Integrating Privacy Computing.*

[16] *Wu, Y. (2025). Optimization of Generative AI Intelligent Interaction System Based on Adversarial Attack Defense and Content Controllable Generation.*

[17] Wu, Y. (2025). Software Engineering Practice of Microservice Architecture in Full Stack Development: From Architecture Design to Performance Optimization. Machine Learning Theory and Practice, 5(1), 64-75.

[18] Liang, Q. (2026). Research on the Renovation Design Path for Enhancing the Efficiency of Mixed-Use Office and Retail Spaces. European Journal of AI, Computing & Informatics, 2(2), 163-170.

[19] Liang, Q. (2026). Reconstruction of Commercial Building Space Reuse Mode Driven by Composite Business Types. International Journal of Engineering Advances, 3(1), 107-113.

[20] Gao, Y. (2026). The Role and Practice of Delaware Law in Global Cross-border M&A Transactions. International Journal of Law, Policy & Society, 2(1), 38-46.

[21] Gao, Y. (2026). Application of Delaware Corporate Law in Cross-Border M&A: Business Structures, Contractual Risk, and Case-Based Lessons. Economics and Management Innovation, 3(2), 128-136.

[22] Wang, N. (2026). Research on Evaluation and Optimization Models of Enterprise Resource Allocation Efficiency from a Data-driven Perspective. Advances in Computer and Communication, 7(1).

[23] Su, J. (2026). Research on Android Real-time Communication System Architecture and High-reliability Assurance Pathways Integrating AI-based Anomaly Detection Mechanisms. Engineering Advances, 6(2).

[24] Wu, Y. (2025). Software Engineering Practice of Microservice Architecture in Full Stack Development: From Architecture Design to Performance Optimization. Machine Learning Theory and Practice, 5(1), 64-75.

[25] Yin, J. (2025). Research on Financial Time Series Prediction Model Based on Multi Attention Mechanism and Emotional Feature Fusion. Socio-Economic Statistics Research (2025), 6(2), 161-169.

[26] Huang, J. (2026). Accessory Dwelling Units as a Policy Execution Challenge: Feasibility, Regulatory Risk, and Early-Stage Decision Modeling. Global Journal of Science & Innovation, 3(1), 1-8.