SAGA: Synthetic Agentic Graph Architecture for Temporal Benchmark Generation

Jiacheng Ding
jding2@memphis.edu
University of Memphis
Memphis, Tennessee, USA

Xiaofei Zhang
xiaofei.zhang@memphis.edu
University of Memphis
Memphis, Tennessee, USA

ABSTRACT

Temporal graph benchmarks with rich semantics and ground-truth anomaly labels are essential for training graph neural networks, yet remain scarce due to privacy constraints and annotation costs. We present SAGA (**S**ynthetic **A**gentic **G**raph **A**rchitecture), a system for generating large-scale, semantically rich temporal graphs via a four-phase pipeline. Our *Skeleton-First, Semantics-Second* architecture decouples structure from semantics: (**S**) an $O(1)$-per-edge skeleton generator produces power-law graphs; (**A**) a dispatcher partitions causally ordered time blocks for parallel execution; (**G**) LLM agents inject domain semantics using RAG-based rule bases across four domains; and (**A**) a state alignment engine resolves conflicts via temporal replay, yielding anomaly labels as natural byproducts. Unlike structural generators (e.g., LDBC SNB, Kronecker/R-MAT) or purely LLM-based approaches, SAGA achieves structural realism, semantic richness, and automatic anomaly labeling in a unified framework. On a single H100 GPU with vLLM batching, SAGA generates 500,000 temporal edges with controlled anomalies in under 90 minutes, scaling to 100,000 nodes while maintaining high local clustering. The system supports real-time pipeline visualization, interactive multi-domain tuning (Finance/AML, Network/IDS, Cyber/APT, Traffic), and a CLI for large-scale GPU-based experiments.

PVLDB Reference Format:
Jiacheng Ding and Xiaofei Zhang. SAGA: Synthetic Agentic Graph Architecture for Temporal Benchmark Generation. PVLDB, 19(12): 4834 - 4837, 2026.
doi:10.14778/3827998.3828134

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/graphuofm/SAGA.

1 INTRODUCTION

Temporal graphs, where nodes and edges carry timestamps, underpin a wide spectrum of applications from anti-money laundering (AML) [4] and intrusion detection [2] to traffic forecasting [14]. Training temporal graph neural networks for these tasks requires benchmarks satisfying three properties simultaneously: *structural realism* (power-law degree distributions), *semantic richness* (domain-specific attributes at fine temporal granularity), and *ground-truth anomaly labels* (for supervised learning without manual annotation).

Table 1: Feature comparison: SAGA *vs.* others

Feature	SAGA	LDBC	AMLSim	Kron.	TGB
Power-law degree distribution	✓	✓	△	△	–
Community structure	✓	✓	✗	△	–
Temporal edges	✓	✓	✓	✗	✓
Configurable time granularity	✓	△	✗	✗	✗
Rich edge attributes	✓	△	✓	✗	–
Multi-domain support	✓	✗	✗	✗	✓
Custom domain (zero-code)	✓	✗	✗	✗	✗
Automatic anomaly labels	✓	✗	△	✗	△
Real-time visualization	✓	✗	✗	✗	✗
Multi-format export	✓	△	△	✗	△

✓ = full support, △ = partial, ✗ = none, – = depends on source data.

Existing approaches address at most two of these aspects. Structural generators such as Erdős–Rényi, Kronecker/R-MAT [3, 5, 13], and TrillionG [15] scale to massive graphs but produce topology only, without semantics or labels. Domain-specific benchmarks such as LDBC SNB and FinBench [6, 12] provide realistic schemas and temporal dynamics, yet remain fixed to predefined domains and lack anomaly annotations. Benchmark suites like TGB and TGB 2.0 [8, 9] standardize evaluation on curated real-world datasets but do not generate new data. Tabular generators such as CTGAN [20] and LLM-based agent systems [16] produce rich semantics, but fail to model graph structure, temporal dependencies, or global consistency.

To address this gap, we present SAGA (**S**ynthetic **A**gentic **G**raph **A**rchitecture), which uniquely unifies scalable structure generation, LLM-driven domain semantics, and automatic ground-truth anomaly labeling in a single system. SAGA introduces a four-phase hybrid pipeline: **S**keleton generation via igraph's C-core, **A**gentic dispatch with causal time-block ordering, **G**enerative injection by LLM agents with RAG rules, and **A**lignment settlement where a global state machine transforms conflicts into labeled anomalies. The key insight is *conflict-as-feature*: agents operate with intentionally incomplete state (future balances marked UNKNOWN), so they naturally generate transactions violating global constraints. The alignment engine labels these violations—overdrafts, frozen-account attempts, structuring patterns—producing ground-truth annotations without manual effort [7]. Table 1 summarizes the feature comparison of SAGA against existing benchmark generation approaches.

We summarize our main contributions as follows: (1) A *Skeleton-First, Semantics-Second* architecture combining $O(1)$-per-edge structure generation with parallel LLM semantic injection (Section 3). (2) A *conflict-as-feature* mechanism producing ground-truth anomaly labels without manual annotation (Section 3).

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828134

(3) Four pluggable RAG rule bases enabling domain-switching with zero code changes (Section 3). (4) An interactive demo with real-time pipeline visualization, multi-domain parameter tuning, and runtime event injection (Section 4).

2 MODEL AND DESIGN FOUNDATIONS

Before describing the pipeline, we formalize the three conceptual pillars underpinning SAGA: the *temporal graph model*, *domain rule bases*, and *conflict-driven labeling*.

Temporal Graph Model. SAGA generates a temporal attributed graph $G = (V, E, \mathcal{T})$, where each edge $e_i = (u, v, t_i, \mathrm{a}_i, \ell_i)$ carries a timestamp t_i, a domain-specific attribute vector a_i (amount, device, IP, risk score, ...), and a ground-truth label $\ell_i \in \{\mathrm{normal}\} \cup \mathcal{L}_{\mathrm{anomaly}}$. Time is organized in three user-configurable layers: *time span* Δ (e.g., 1 year) > *macro-block granularity* B (e.g., 1 day, yielding $K = \lceil \Delta / B \rceil$ blocks) > *micro-timestamp precision* δ (e.g., 1 minute). This hierarchy—directly exposed as three dropdowns in the UI (Figure 3d)—decouples coarse causal ordering from fine-grained temporal realism. The graph topology follows a power-law degree distribution via Barabási-Albert preferential attachment [1] with configurable exponent $\gamma \in [1.5, 3.5]$, implemented through igraph's C-core at $O(1)$ per edge.

Conflict-Driven Anomaly Labeling. Each rule base defines validation predicates Φ (sufficient balance, active account, no self-transfer, ...), and ground-truth labels arise from *validating* generated edges against Φ during replay, rather than from manual annotation. When an agent processes a task in block k, its view of node v is:

$$\hat{S}_k(v) = \begin{cases} f(\text{settled edges in blocks } 1..k-1) & \text{if settled,} \\ \text{UNKNOWN} & \text{otherwise.} \end{cases} \quad (1)$$

Block-1 tasks carry known initial states; later blocks carry UNKNOWN balances, forcing agents to estimate—and sometimes violate—true constraints. The alignment engine replays all micro-edges in global timestamp order and evaluates each against the true state S_{t_i}:

$$\ell_i = \begin{cases} \text{normal} & \text{if } \forall j\colon\ \phi_j(e_i, S_{t_i}) = 1, \\ \text{anomaly}_j & \text{if } \exists j\colon\ \phi_j(e_i, S_{t_i}) = 0. \end{cases} \quad (2)$$

Repeated violations escalate risk; exceeding θ_{freeze} freezes the account, causing all downstream edges to fail—producing cascading anomaly chains that mirror real-world fraud propagation [10]. The alignment engine assigns each label *type* by validating an edge against Φ during replay; to make label *volume* reproducible, the user sets a target anomaly rate r, and SAGA designates $\lfloor |E| r \rfloor$ edges to carry violations, so the labeled-anomaly rate is exact and deterministic regardless of LLM output variance.

3 SYSTEM OVERVIEW

Figure 1 illustrates the four-phase SAGA pipeline. Each phase operates on strictly typed data contracts, i.e., MacroEdge → Task → MicroEdge → FinalEdge. We now explain each phase in detail.

Phase S: Skeleton Generation. The skeleton generator produces a bare temporal graph $G_s = (V, E_s, \tau)$ where each macro-edge $e = (u, v, T, type)$ specifies endpoints and a coarse time block. We employ igraph's C-core Barabási-Albert preferential attachment [1] with configurable exponent $\gamma \in [1.5, 3.5]$, producing clean power-law distributions ($R^2 > 0.94$) without Kronecker artifacts [17]. Time is organized in three configurable layers: *time span* > *macro block* > *micro timestamp*.

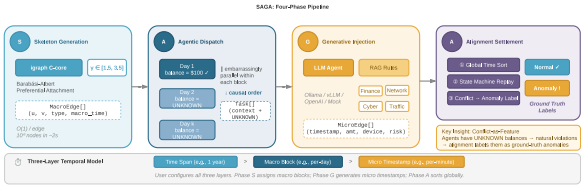

Figure 1: The SAGA pipeline. S: igraph C-core skeleton ($O(1)$/edge); A_1: causal time-block dispatch with UNKNOWN balances; G: LLM agents with RAG rule injection; A_2: temporal replay producing ground-truth anomaly labels.

Phase A_1: Agentic Dispatch. The dispatcher groups macro-edges into tasks by time block, processed in strict chronological order. Within a block, tasks are embarrassingly parallel. For Day k ($k > 1$), node balances are marked UNKNOWN—this *intentional information asymmetry* is the source of anomaly emergence.

Phase G: Generative Injection. LLM agents expand macro-edges into rich micro-transactions with timestamps, amounts, device fingerprints, and risk scores. SAGA supports Ollama, vLLM [11], OpenAI API, and deterministic Mock mode via a unified LLMCaller interface. Agents follow RAG-encoded temporal distributions (e.g., 70% business-hour weight for finance). To bound LLM hallucination, every generated edge is schema-validated and range-clamped (timestamps confined to their block, amounts to the available balance, attributes to legal enumerations); malformed outputs fall back to rule-based values, and the deterministic Mock backend reproduces structure with no LLM call.

Phase A_2: Alignment Settlement. All micro-edges are sorted by timestamp and replayed through a per-node state machine tracking balance, risk level, and account status. Each edge is validated against business rules; failed validations produce labeled anomalies (e.g., overdraft, structuring, large-amount). Risk escalation creates cascading chains: repeated anomalies freeze accounts, causing all downstream transactions to be flagged—mimicking real-world fraud propagation [10].

Pluggable RAG Rule Bases. Four rule bases provide node taxonomies, interaction patterns, temporal distributions, and anomaly conditions: **Finance/AML** [7, 10] (six AML patterns, twelve anomaly triggers), **Network/IDS** [2] (eight attack categories), **Cyber/APT** [18] (14 ATT&CK tactical phases), and **Traffic** [14] (Krauss car-following, cascading failures).

Beyond the four built-in domains, users can define entirely new scenarios—such as IoT networks or supply chains—by pasting domain rules in natural language (Figure 3a). The system sends the rule text to the LLM, which automatically extracts 10–25 typed parameters (sliders, toggles, dropdowns) with sensible defaults and valid ranges (Figure 3b). Users retain full control: they can modify,

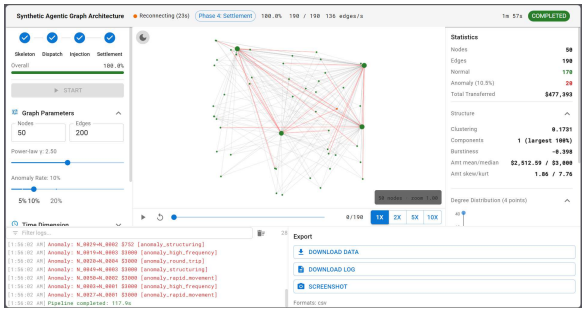

Figure 2: The SAGA dashboard after a completed financial AML run (50 nodes, 190 edges, 10.5% anomalies). Left: four-phase stepper and configuration panels. Center: Sigma.js WebGL graph with anomalous edges highlighted. Right: real-time statistics. Bottom: settlement event log with color-coded anomaly detections.

delete, or add parameters before generation begins. This mechanism enables domain-switching with zero code changes and zero retraining.

4 DEMONSTRATION

Our demonstration is implemented as a React + Material Design 3 dashboard with real-time WebSocket streaming and Sigma.js WebGL rendering. Figure 2 shows the system after a completed run, while Figure 3 presents the interactive control panels.

We organize the demo around a realistic user workflow, guiding participants through three progressively deeper scenarios that illustrate SAGA's capabilities from rapid prototyping to large-scale deployment.

Scenario 1: Quick Start — From Zero to a Benchmark in Minutes. A researcher needs an AML transaction dataset but finds that existing benchmarks (e.g., Elliptic Bitcoin [19]) are fixed in scale and lack configurable anomaly rates. She opens the SAGA web dashboard, selects *Financial AML* from the scenario panel (Figure 3a), and the system instantly loads 28 domain rules and infers 10 typed parameters—sliders, toggles, and dropdowns—with sensible defaults (Figure 3b). She adjusts the graph to 50 nodes and 190 edges for a quick pilot run (Figure 3c), sets the time span to 7 days at day/minute granularity (Figure 3d), and clicks *Start*. The four-phase stepper animates in real time: skeleton edges appear in the graph canvas within a second, tasks dispatch by time block, LLM agents stream micro-edges with live speed counters, and settlement results flash green (normal) or red (anomaly) in the event log. The entire run completes in under 10 seconds in Mock mode. She inspects the statistics panel (Figure 3e): 190 edges, 20 anomalies (10.5%), clustering coefficient 0.17. She drags the anomaly rate slider from 10% to 25% and reruns—the anomaly breakdown pie chart (Figure 3f) updates immediately, confirming precise control. Satisfied with the pilot, she clicks *Download Data* (Figure 3h, i) and receives clean CSV files, where edges with timestamps, amounts, labels, and a saga_meta.json recording all parameters are ready to be loaded into PyG or DGL without any preprocessing.

Scenario 2: Domain Switching — Exploring Multiple Industries Without Code. Assume the same researcher now needs a network intrusion dataset for a comparative study. Rather than

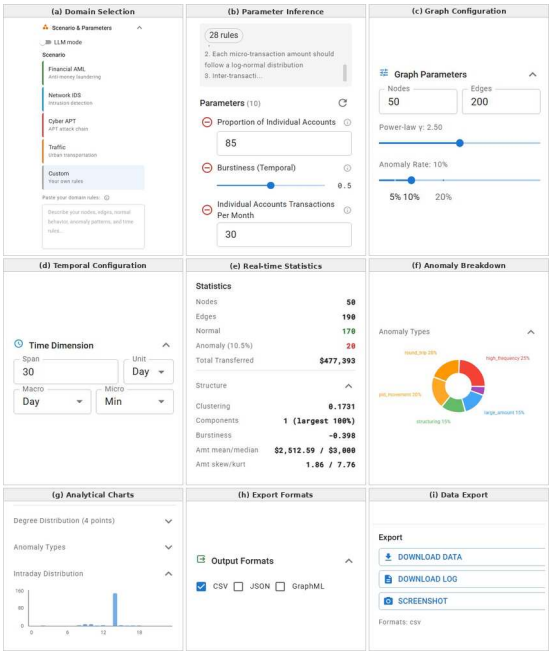

Figure 3: Interactive control panels: domain selection and custom rules (a), LLM-inferred typed parameters (b), graph topology configuration (c), three-layer temporal model (d), real-time statistics (e), anomaly type breakdown (f), intraday temporal distribution (g), export format selection (h, i).

searching for a new benchmark, she switches the domain dropdown to *Network IDS* (Figure 3a). The parameter panel regenerates automatically: *Proportion of Individual Accounts* becomes *Ratio of Internal Hosts*, *Burstiness* shifts to reflect diurnal packet patterns, and new parameters such as *Average Packet Size* appear—all inferred by the LLM from the IDS rule base [2]. No code changes, no configuration files, no domain expertise required. She can even paste custom rules in natural language (e.g., describing an IoT sensor network) and the system generates a fully typed parameter panel in seconds. This zero-code domain switching is what distinguishes SAGA from fixed-schema generators like LDBC SNB, which require schema redesign and code modifications for each new domain.

Scenario 3: Scaling Up — From Prototype to Large-Scale Experiments. Having validated her approach on small graphs, the researcher scales to 10,000 nodes and 50,000 edges. The web dashboard remains interactive: Sigma.js renders the graph at 60fps via WebGL, and the time-axis scrubber enables replay at 1×/2×/5×/10× speed. For even larger experiments (100K+ nodes), she switches to the SAGA CLI, which accepts the same parameters via YAML configuration files and runs on GPU clusters. On a single H100 with vLLM continuous batching (256 concurrent requests), SAGA generates 50,000 edges in 9 minutes and 500,000 edges in 89 minutes, scaling linearly with edge count. A deterministic Mock mode bypasses LLM calls entirely, producing 500K edges with full structural annotations in 30 seconds for rapid prototyping and CI testing. Throughout, the exported data maintains consistent quality: power-law $R^2 > 0.94$ and exact anomaly-rate control from

0% to 30%. Each export includes multi-format outputs (CSV, JSON, GraphML) directly loadable by pandas, Neo4j, and igraph, plus a metadata file for full reproducibility. SAGA shifts the researcher's effort from *data acquisition* to *data specification*—freeing them to focus on mining rather than curation.

Implementation. The backend is built on Python 3.8+ with python-igraph, asyncio, orjson, and numpy. The frontend uses React 18, Vite, MUI v5, Sigma.js, and Recharts. SAGA supports four LLM backends (Ollama, vLLM [11], OpenAI API, Mock) via a unified LLMCaller interface, switchable through a single environment variable. The system has been tested on NVIDIA Quadro RTX 6000 (24 GB) and H100 (80 GB).

5 EVALUATION

We evaluate SAGA on *structural fidelity*, *controllability*, and *scalability* using an NVIDIA H100 (80 GB) GPU running vLLM with Qwen2.5-3B-Instruct and 256 concurrent agent requests. All experiments use seed = 42 for reproducibility.

Structural fidelity. We compare SAGA against NetworkX-BA and igraph-BA across seven scales (100 to 100K nodes). Both baselines consistently undershoot the target edge count by 15–25 edges due to BA initialization effects; SAGA delivers exactly the requested number at every scale (e.g., 25,000/25,000 at 5K nodes, 500,000/500,000 at 100K nodes). More importantly, NetworkX and igraph produce bare topology with no timestamps, no semantic attributes, and no anomaly labels—SAGA provides all three. As a control, replacing the BA skeleton with an Erdős-Rényi random graph collapses the local clustering coefficient to 0.002, confirming that community structure originates from preferential attachment rather than downstream processing.

Controllability. Precise control over output properties is essential for benchmark generation. SAGA achieves exact anomaly-rate control by decoupling anomaly selection from edge generation: anomaly edges are selected first (count fixed as $\lfloor |E| \times r \rfloor$), then normal edges fill the remainder. Across six target rates (0%, 5%, 10%, 15%, 20%, 30%) on 5K-node / 25K-edge graphs, every run matches the target exactly—0.0%, 5.0%, 10.0%, 15.0%, 20.0%, 30.0%—regardless of LLM output variance. The power-law exponent is also controllable: varying γ from 1.5 to 3.5 shifts the fitted exponent monotonically ($\hat{\gamma} = 2.20 \rightarrow 1.19$), allowing users to dial the degree distribution from highly heterogeneous (more hubs) to more uniform.

Scalability. Generation time scales linearly with edge count. On a single H100 with vLLM continuous batching at 256 concurrency, SAGA generates 500 edges in 5.3 seconds, 5,000 edges in 53 seconds, 25,000 edges in 4.5 minutes, 50,000 edges in 9 minutes, and 500,000 edges in 89 minutes. The bottleneck is LLM inference in Phase G, which accounts for over 99% of wall time at scales above 1K nodes; skeleton generation (Phase S) takes under 1 second even at 100K nodes. A deterministic Mock mode bypasses LLM calls entirely, generating 500K edges with full structural annotations in 30 seconds—useful for rapid prototyping and CI testing.

Ablation. We disable each component on a 5K-node / 25K-edge graph: removing semantic injection (Mock mode) reduces generation time from 263 s to 0.4 s, a 250× speedup for users who need structure only or lack GPU resources; removing alignment leaves edges unvalidated, so balance-consistency and the cascading risk structure are lost; removing RAG rules preserves the anomaly rate but reduces attribute diversity, indicating that domain knowledge improves semantic quality rather than label correctness.

6 CONCLUSION

SAGA demonstrates that treating logical conflicts between independent LLM agents as features naturally produces ground-truth anomaly labels. The pluggable RAG architecture enables domain-switching with zero engine changes, and LLM-driven parameter inference lowers the barrier for non-expert users. In the future, we will support adaptive feedback loops where settlement results inform subsequent agent prompts, and temporal graph learning along generation.

REFERENCES

[1] Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.

[2] Canadian Institute for Cybersecurity. 2017. CIC-IDS-2017: Intrusion Detection Evaluation Dataset. https://www.unb.ca/cic/datasets/ids-2017.html.

[3] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A Recursive Model for Graph Mining. In *Proc. SIAM SDM*. 442–446.

[4] Dawei Cheng et al. 2024. Graph Neural Networks for Financial Fraud Detection: A Survey. *Comput. Surveys* (2024).

[5] Paul Erdős and Alfréd Rényi. 1960. On the Evolution of Random Graphs. *Publ. Math. Inst. Hung. Acad. Sci.* 5 (1960), 17–61.

[6] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat-Pérez, Minh-Duc Pham, and Peter Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *Proc. ACM SIGMOD*. 619–630.

[7] Financial Action Task Force (FATF). 2020. *Money Laundering and Terrorist Financing: Methods and Trends*. Technical Report. FATF.

[8] Julia Gastinger, Shenyang Huang, Mikhail Galkin, Erfan Loghmani, Ali Parviz, Farimah Poursafaei, Jacob Danovitch, Emanuele Rossi, Ioannis Koutis, Heiner Stuckenschmidt, Reihaneh Rabbany, and Guillaume Rabusseau. 2024. TGB 2.0: A Benchmark for Learning on Temporal Knowledge Graphs and Heterogeneous Graphs. In *NeurIPS Datasets and Benchmarks Track*.

[9] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2023. Temporal Graph Benchmark for Machine Learning on Temporal Graphs. In *NeurIPS Datasets and Benchmarks Track*.

[10] IBM Research. 2021. AMLSim: Anti-Money Laundering Transaction Simulator. https://github.com/IBM/AMLSim.

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. vLLM: Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proc. ACM SOSP*. 611–626.

[12] LDBC Financial Benchmark Task Force. 2022. The LDBC Financial Benchmark. In *LDBC Technical Report*.

[13] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker Graphs: An Approach to Modeling Networks. *Journal of Machine Learning Research* 11 (2010), 985–1042.

[14] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. SUMO: Simulation of Urban Mobility. https://sumo.dlr.de/.

[15] Himchan Park and Min-Soo Kim. 2017. TrillionG: A Trillion-scale Synthetic Graph Generator Using a Recursive Vector Model. In *Proc. ACM SIGMOD*. 913–928.

[16] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proc. ACM UIST*. 1–22.

[17] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. 2011. An In-depth Study of Stochastic Kronecker Graphs. In *Proc. IEEE ICDM*. 587–596.

[18] The MITRE Corporation. 2024. MITRE ATT&CK Enterprise Framework v18. https://attack.mitre.org/.

[19] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I. Weidele, Claudio Bellei, Tom Robinson, and Charles E. Leiserson. 2019. Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics. In *KDD Workshop on Anomaly Detection in Finance*.

[20] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling Tabular Data using Conditional GAN. In *NeurIPS*, Vol. 32.