XGAP: Ambiguity-aware Cross-Platform Planning for Natural Language Graph Queries

Haolai Che
supervised by Quanqing Xu[†]*, Yinghui Wu*[‡]
OceanBase, Ant Group[†], Case Western Reserve University[‡]
Shenzhen, China
hxc859@case.edu

ABSTRACT

Natural-language queries over schemaless graph data inherently suffer from structural and semantic ambiguities. The challenge intensifies when graph data is isolated and accessed independently by heterogeneous engines with different graph query languages. We propose XGAP, an ambiguity-aware cross-platform graph query planner that integrates language models to generate structural interpretations and compiles them directly into distributed, compatible, and executable optimized plans. Without integrating raw data, XGAP solves a multi-criteria selection problem that balances interpretation plausibility, residual ambiguity, plan diversity, and cross-system execution cost, thereby effectively pushing execution down to distributed native engines. We outline our research plan for implementation and evaluation.

VLDB Workshop Reference Format:
Haolai Che. XGAP: Ambiguity-aware Cross-Platform Planning for Natural Language Graph Queries. VLDB 2026 Workshop: VLDB Ph.D. Workshop.

1 INTRODUCTION

Natural-language queries over schemaless graph data often admit multiple valid structural and semantic interpretations. This challenge becomes harder in cross-platform settings, where relevant graph data is distributed across isolated systems with different data models, query languages (*e.g.,* GQL, Cypher, SPARQL), and execution capabilities. Such scenarios leave open the intended entities, link semantics, relevance criteria, and execution strategy [1]. In response, data systems must address not only query optimization but also the resolution of ambiguities in the intended query plans.

Example 1: Consider a natural language (NL) query Q_N in Fig. 1 for a financial risk assessment task that aims to retrieve answers from accessible views independently hosted by a bank on a transaction network, and a FinTech that reports risk assessment information. Resolving Q_N in real-world settings involves three challenges.

First, the input query, when posed on graph data, exhibits ambiguity that goes beyond word-level interpretation. (1) The term "Alice" introduces *entity ambiguity* with little local context in Q_N (*e.g.,* distinguishing Alice Smith from Alice Zhang). (2) The vague term "close money transfers" bears *topological ambiguity*: it may have multiple interpretations, all expressible by path patterns from a one-hop direct link or a multi-hop variable-length path. (3) The criterion "high-risk" introduces *semantic ambiguity* as it can be quantified by different unaligned risk measurements and models (*e.g.,*'Suspect_High' in 'hasRisk' or 'Unknown' controllers).

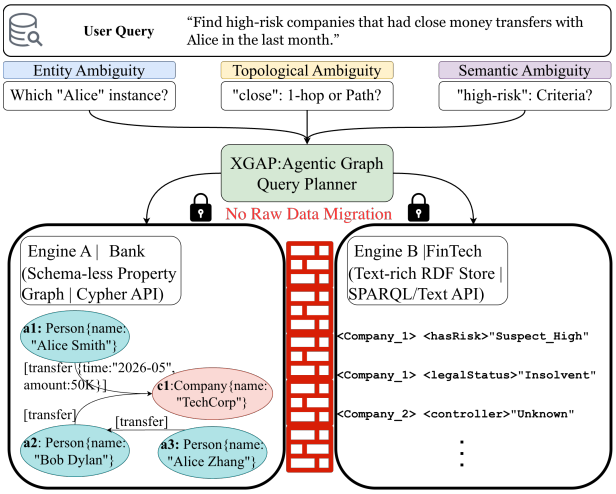

Figure 1: Real-world financial risk analysis requires ambiguity-aware, cross-platform search over independently hosted, isolated schemaless graph data. A graph query planner needs to generate and deploy executable query plans by simultaneously resolving all three challenges.

Second, accurate execution requires spanning a set of language-specific data systems. Due to data ownership, privacy, and service type, each institution prefers to deploy its own secured and optimized data infrastructure to process on-site querying. For example, Bank operates on property graph engines optimized for dense, real-time transactions, whereas FinTech adopts RDF/SPARQL store for monthly-updated corporate metadata with consistency validation. A query server therefore must bridge cross-platform query semantics (Cypher and SPARQL) and local engine capabilities to optimize cost and accuracy holistically.

In addition, the underlying graph data is text-rich and schemaless, lacking a global, predefined schema. Original attributes and data are often stored as *e.g.,* JSON objects within each domain site and remain unpublished. For instance, corporate risk profiles in FinTech are partially disclosed and expressed through disjoint predicates such as ex:hasRisk and ex:legalStatus. This absence of schema prevents a planner from mapping abstract NL statements directly to structured queries via schema matching or data integration. □

Proceedings of the VLDB Endowment. ISSN 2150-8097.

State of the Art. A host of research has been conducted independently for each of these challenges. (1) Text-to-query systems, including NL-to-SQL [8] and NL-to-graph queries [10], either assume available schema and perform schema matching or rule-based conversion, or instruction-guided language models and learning-based approaches, without rigorous evaluation metrics on query quality and ambiguity minimization [9]. (2) Existing graph query optimizers typically start with declarative, language-specific operators, and focus on generating optimized local logical plans (via cardinality estimation or pattern rewriting [11]) for a single site, rather than an executable plan that is ready for cross-platform deployment. These approaches face challenges in ambiguity-aware graph analysis tasks across isolated domains, where the planner must determine and deploy plausible logical operations to heterogeneous, isolated engines and balance execution costs and accuracy, admitting that schema-based data integration is a luxury.

Vision and Approach. We propose XGAP, an ambiguity-aware cross-platform graph query planner for natural-language graph intents over heterogeneous and isolated graph systems. XGAP jointly models query interpretation and physical planning through three components: an intermediate graph-pattern representation, backend capability descriptors, and a multi-criteria plan selection objective. Given an ambiguous NL query Q_N, XGAP generates plausible graph-pattern interpretations $\mathcal{I}$, validates their feasibility against native engine capabilities, estimates cross-system execution cost, and returns a compact top-K set of executable plans that balance confidence, ambiguity coverage, diversity, and cost.

Methodology. XGAP is built on a library of primitive graph operators $\mathcal{O}$ covering useful core fragments of Cypher, SPARQL, and GQL. These operators are parameterized and synthesized into graph-pattern representations $\mathcal{I}$ for Q_N, guided by graph algebra for topological constraints and language models for semantic constraints. Using cross-system metadata collected from backend descriptors, XGAP compiles each feasible interpretation into language-specific logical and execution plans without requiring bulk raw-data integration or full graph migration.

Framework Overview. XGAP proceeds in three stages. First, it decomposes Q_N into candidate graph patterns $\mathcal{I}$ that capture alternative entity, topology, and predicate interpretations. Second, it validates each candidate against available metadata and engine capabilities, and uses a learned cost estimator to rank feasible cross-platform plans. Third, it schedules executable subplans to native backends and assembles the returned results for the user.

Research questions. We focus on three questions. (1) *Query Modeling*. How can we design an abstract graph-query model whose operators $\mathcal{O}$ express useful graph-query fragments while exposing ambiguity in NL intents? (2) *Query Representation Generation.* How can we efficiently synthesize operators $\mathcal{O}$ into candidate interpretations $\mathcal{I}$ for schemaless graphs while reducing unsupported or redundant ambiguity? (3) *Plan Generation.* How can we compile and select optimized, deployable cross-platform plans from $\mathcal{I}$ under heterogeneous engine capabilities and cost models?

We formulate the problem in Sec. 3, present the initial design in Sec. 4, and outline the research plan in Sec. 5.

2 PRELIMINARIES

Graphs and patterns. We model each source as a text-rich, schemaless directed graph $G = (V, E, f_A)$, where V and $E \subseteq V \times V$ are nodes and edges, and f_A assigns properties, embeddings, or text attributes to graph elements. Following graph pattern languages [3, 4, 6], a pattern I is a graph (V_I, E_I, f_v, f_e), where f_v maps pattern nodes to predicate conditions (*e.g.,* entity bindings, keywords, or textual constraints), and f_e maps pattern edges to connectivity constraints (*e.g.,* direct edges, bounded paths, or regular path expressions). Such patterns capture useful fragments of the underlying modern graph query languages such as Cypher and GQL [3, 4].

Logical plans. Let $\mathcal{O}$ be a vocabulary of logical graph operators, such as EntityLookup, NodeScan, EdgeExpand, PathExpand, Filter, Join, Aggregate, Rank, and Project. Given a pattern I, XGAP compiles it into a language-agnostic logical operator DAG $L = (\Omega, D)$, where Ω is a set of operator calls instantiated from $\mathcal{O}$, and $D \subseteq \Omega \times \Omega$ specifies data-flow dependencies. While I captures abstract graph-search semantics, L specifies how I is evaluated.

Cross-platform execution plans. A cross-platform environment is described by backend descriptors $B = \{b_1, \ldots, b_n\}$, where each b_i records a native engine's supported operators, query language, input/output format, and performance constraints. Given $L = (\Omega, D)$, a cross-platform execution plan is $\xi = (L, \Pi, \Delta)$, where $\Pi : \Omega \to B$ assigns each operator call to a backend, and Δ records cross-backend dependencies such as data movement, serialization, and result assembly. A plan is executable only if each assigned backend supports the corresponding operator and generated query fragment.

Unified query plans. The compiler output is a unified graph query plan $\Psi = (I, L, \xi, Q)$, where I is the graph-pattern interpretation of the input query Q_N, L is the logical plan, ξ is the cross-platform execution plan, and $Q = \{q_1, \ldots, q_m\}$ is the set of backend-specific target queries. This representation allows XGAP to compare alternative interpretations and deployments before issuing native queries to isolated backends.

3 PROBLEM FORMULATION

Input and output. Given a natural-language graph query Q_N, isolated source graphs $\mathcal{G} = \{G_1, \ldots, G_n\}$, and backend descriptors $B = \{b_1, \ldots, b_n\}$, XGAP returns a top-K set of executable plans $\mathcal{S}^* = \{\Psi_1, \ldots, \Psi_K\}$. Each plan represents one plausible interpretation of Q_N and one feasible deployment over the available backends.

Feasible plans. Given a unified query plan $\Psi = (I, L, \xi, Q)$ generated by XGAP, we say Ψ is *feasible* if it satisfies the following structural and syntactic constraints: (1) L is acyclic and every operator call in Ω instantiates an operator in $\mathcal{O}$; (2) $\Pi : \Omega \to B$ assigns each operator call to a backend that supports it; (3) Δ resolves all cross-backend dependencies, such as data movement and result assembly; and (4) each $q_i \in Q$ is syntactically valid for the query language of its assigned backend.

Interpretation ambiguity. Let $\mathcal{I}(Q_N)$ be the candidate interpretation space for Q_N. Although $\mathcal{I}(Q_N)$ may contain many alternatives, XGAP only returns a bounded top-K set of executable plans. We therefore do not optimize the raw number of interpretations

directly; instead, we evaluate each interpretation by *plausibility* and *residual ambiguity*, and use *diversity* to avoid returning near-duplicate plans. Specifically,

(1) For each interpretation $I \in \mathcal{I}(Q_N)$, let $\rho(I; Q_N) \in (0, 1]$ be its normalized plausibility score, estimated from model likelihood, metadata matching, and validation, with $\sum_{I \in \mathcal{I}(Q_N)} \rho(I; Q_N) = 1$.

(2) We define its residual ambiguity as

$$A(I; Q_N) = w_e A_e(I; Q_N) + w_p A_p(I; Q_N) + w_t A_t(I; Q_N),$$

where A_e, A_p, and A_t measure entity, predicate/semantic, and topological ambiguity, respectively. Each component is normalized to $[0, 1]$, and $w_e, w_p, w_t \geq 0$ with $w_e + w_p + w_t = 1$.

The interpretation penalty is defined as

$$C_{\text{int}}(I; Q_N) = -\log \rho(I; Q_N) + \lambda_A A(I; Q_N),$$

where $\lambda_A \geq 0$ controls the penalty for residual ambiguity.

(3) To avoid returning near-duplicate plans, let $D(I_i, I_j) \in [0, 1]$ measure the diversity between two interpretations, such as pattern edit distance or semantic distance between their bindings.

Execution cost. For a feasible plan $\Psi = (I, L, \xi, Q)$, we model cross-platform execution cost as

$$C_{\text{exec}}(\Psi) = \max_{q_i \in Q} T_{\text{loc}}(q_i) + \sum_{\Delta_j \in \Delta} C_{\text{move}}(\Delta_j).$$

Here, $T_{\text{loc}}(q_i)$ estimates the local runtime of target query q_i on its assigned backend, and $C_{\text{move}}(\Delta_j)$ captures serialization, transfer, and result assembly for cross-backend dependency Δ_j. The max term models parallel execution over isolated engines; a sum can be used when subplans are scheduled sequentially.

Plan selection. Let $\mathcal{A}(Q_N, \mathcal{G}, B)$ be the feasible plan space. Each plan $\Psi = (I, L, \xi, Q)$ is assigned a total cost

$$C(\Psi; Q_N) = \alpha C_{\text{exec}}(\Psi) + \beta C_{\text{int}}(I; Q_N),$$

where $\alpha, \beta \geq 0$ balance execution efficiency and interpretation quality. XGAP selects

$$\mathcal{S}^* = \underset{\mathcal{S} \subseteq \mathcal{A}(Q_N, \mathcal{G}, B),\, |\mathcal{S}| = K}{\arg\min} \left(\sum_{\Psi \in \mathcal{S}} C(\Psi; Q_N) - \lambda_D \sum_{\substack{\Psi_i, \Psi_j \in \mathcal{S} \\ i<j}} D(I_i, I_j) \right),$$

where $\lambda_D \geq 0$ controls the preference for covering meaningfully different interpretations. This objective favors plans that are executable, cost-effective, plausible, and collectively diverse under the ambiguity of the original user intent.

4 XGAP: ALGORITHMIC FRAMEWORK

We next outline how XGAP algorithmically generates cross-platform plans. The three conceptual steps in Sec.1 are organized into two algorithmic stages: (1) interpretation generation and logical compilation, and (2) physical grounding and top-K plan selection.

Interpretation Generation and Logical Compilation. Given an NL query Q_N, XGAP first generates a set of candidate interpretations $\mathcal{I}(Q_N)$. Each interpretation $I \in \mathcal{I}(Q_N)$ is an intermediate representation and $L = (\Omega, D)$ is a logical operator DAG compiled from I. To construct I, XGAP retrieves a compact metadata context from the backend descriptors B, and available source summaries,

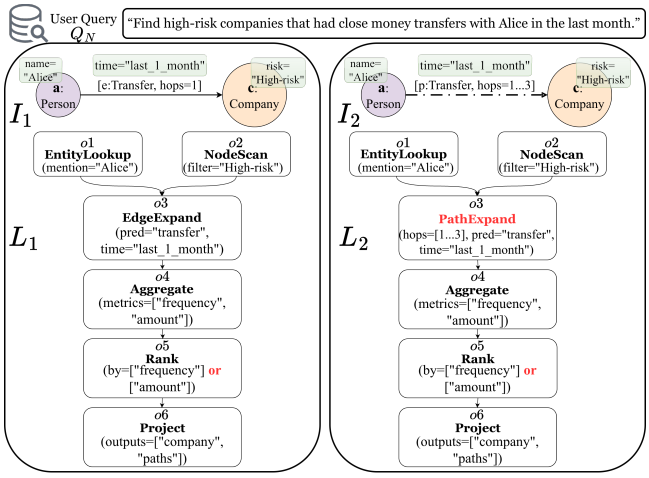

Figure 2: Running example of interpretation generation and logical compilation for query Q_N. Two candidate interpretations, represented by patterns I_1 and I_2, capture topological ambiguity and compile into logical plans L_1 and L_2 that differ in their expansion operator.

including candidate entity mentions, predicate lexicons, textual attributes, and supported connectivity patterns. This step avoids exposing the full schemaless graph or large text-embedding space to the interpretation generator. The LLM is invoked only over this bounded context: one call generates at most M sketches, plus at most one batch-level repair call. All later steps are deterministic, so LLM use is bounded per query and independent of the number of backends or physical plans. Backend descriptors prune unsupported capabilities, while source summaries enable domain-aware augmentation and bounded-beam ranking.

Using the above pruned context, XGAP instantiates alternative graph patterns that capture ambiguity in entity bindings, predicate mappings, and structural constraints. It then applies syntax-guided compilation to translate each pattern I into a language-agnostic logical plan $L = (\Omega, D)$, where each operator call in Ω is instantiated from the operator vocabulary $\boldsymbol{O}$. This compilation step filters out patterns that cannot be expressed by $\boldsymbol{O}$, and ensures that each surviving interpretation has a well-formed logical evaluation plan.

Example 2: Consider the query Q_N in Fig. 2. The phrase "close money transfers" is structurally ambiguous: it may refer to a direct transfer edge or to a bounded multi-hop transfer path. XGAP therefore constructs two candidate patterns, I_1 and I_2. Both patterns share the same terminal predicates, such as name="Alice" on v_1, risk="High-risk" on v_2, and the temporal filter time="last_1_month". They differ in the connectivity constraint: I_1 uses a direct edge (hops=1), whereas I_2 uses a bounded path (hops=1..3).

The two patterns compile into logical plans L_1 and L_2 with similar surrounding operators but different expansion kernels. After o_1 (EntityLookup) and o_2 (NodeScan) resolve text-rich terminal predicates, L_1 invokes o_3 (EdgeExpand) for direct-neighbor expansion, while L_2 uses a bounded PathExpand operator with hops=[1,3]. Both plans then apply o_4 (Aggregate) to compute transfer features including frequency and amount, o_5 (Rank) to rank candidates, and o_6 (Project) to a common pattern ["company", "paths"]. □

Table 1: Planned heterogeneous backends.

Execution Engine	Target operations
Neo4j / Kùzu [7]	Pattern matching; bounded traversal
Jena Fuseki	RDF-style semantic lookup
GraphScope [5]	Large traversal; aggregation; ranking

Physical Grounding and Top-K Selection. After logical compilation, XGAP grounds each candidate logical plan into executable backend-specific queries. For each feasible interpretation I with logical plan $L = (\Omega, D)$, the optimizer uses backend descriptors B to assign operator calls to native engines through $\Pi : \Omega \to B$, inserts cross-backend dependencies Δ, and generates target queries $Q = \{q_1, \ldots, q_m\}$ in the corresponding native languages (*e.g.,* Cypher, SPARQL, or graph-processing APIs). This produces a complete plan $\Psi = (I, L, \xi, Q)$, where ξ is cross-platform execution plan.

To rank feasible plans, XGAP estimates both interpretation quality and execution cost. Interpretation quality is measured by the penalty $C_{\text{int}}(I; Q_N)$ defined in Sec. 3. Execution cost is estimated by $C_{\text{exec}}(\Psi)$, which combines local runtime and cross-backend data-movement cost. Since L is an operator DAG, the cost estimator can use features from operator types, estimated cardinalities, backend assignments, and dependency edges in Δ; a learned model, such as a graph neural network over the annotated operator DAG, can be used to predict local runtime and data-transfer overheads.

Finally, XGAP evaluates the total plan cost $C(\Psi; Q_N)$ and selects a top-K set $\mathcal{S}^*$ using the objective in Sec. 3. The returned set favors plans that are feasible, low-cost, plausible under the user intent, and diverse enough to cover meaningfully different interpretations.

5 RESEARCH PLAN

Research tasks. Our plan follows the three questions in Sec. 1 : modeling ambiguous graph intents, generating interpretations, and selecting executable cross-platform plans.

Task 1: Ambiguity-aware query modeling. We will formalize the graph-pattern interpretation I, the logical plan L, and operator vocabulary O for a practical core of graph-query features, including node/edge predicates, bounded path expansion, filtering, aggregation, ranking, and projection. Instead of claiming full language coverage, we will characterize which fragments of Cypher, SPARQL, and GQL-style path patterns can be compiled into XGAP's logical plan DAGs. We will also define ambiguity components A_e, A_p, and A_t for entity, predicate, and topological uncertainty.

Task 2: Interpretation generation and validation. We will generate a compact candidate space $\mathcal{I}(Q_N)$ for an NL query Q_N over schemaless, text-rich graphs, using limited metadata from isolated backends without assuming a global schema or raw data integration. We will design metadata pruning, entity/predicate retrieval, and syntax-guided compilation methods that produce well-formed patterns I and logical plans $L = (\Omega, D)$. Candidate interpretations will be validated against operator signatures and backend descriptors B to remove unsupported plans.

Task 3: Cross-platform plan selection. We will develop cost models and selection algorithms for feasible plans $\Psi = (I, L, \xi, Q)$. This includes estimating local runtime T_{loc}, cross-backend movement cost C_{move}, and interpretation penalty $C_{\text{int}}(I; Q_N)$, and combining them using the objective in Sec. 3. We will first study top-K selection with duplicate removal, then consider diversity-aware or Pareto-style variants balancing cost, plausibility, ambiguity, and coverage.

Evaluation plan. We will prototype XGAP over heterogeneous graph backends (Table 1), routing operator fragments to native engines, generating backend-specific target queries Q, and assembling cross-backend results through dependencies Δ. We will evaluate on financial transaction graphs [13], OpenAlex-derived bibliographic graphs [12] and cybersecurity provenance graphs [2].

Evaluation metrics. We will use four metric groups: (1) *interpretation quality*, measured against acceptable interpretations derived from executable gold queries or query graphs in graph-query benchmarks. Top-K precision, recall, and ranking evaluate multiple valid interpretations; (2) *plan feasibility*, the fraction of plans passing operator, backend, and syntax validation; (3) *planning quality*, including planning latency, top-K diversity, and predicted/observed cost agreement; and (4) *execution performance*, including latency, data movement, and throughput against centralized, pipeline-style, and native-engine plan baselines.

Implementation roadmap. Months 1–3 will define the supported query fragment and implement metadata-guided interpretation generation. Months 4–7 will add backend descriptors, feasibility validation, target-query generation, and initial cost estimators. Months 8–12 will integrate heterogeneous backends, complete top-K selection, and evaluate against baselines.

ACKNOWLEDGMENTS

This work was supported by Ant Group Research Intern Program.

REFERENCES

[1] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal* 28, 5 (2019), 793–819.
[2] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z Berkay Celik, Xiangyu Zhang, and Dongyan Xu. 2021. {ATLAS}: A sequence-based learning approach for attack investigation. In *(USENIX)*.
[3] Renzo Angles, Angela Bonifati, Roberto García, and Domagoj Vrgoč. 2024. Path-based algebraic foundations of graph query languages. *arXiv preprint* (2024).
[4] Pablo Barceló, Leonid Libkin, Anthony W Lin, and Peter T Wood. 2012. Expressive languages for path queries over graph-structured data. *TODS* 37, 4 (2012), 1–46.
[5] Wenfei Fan, Tao He, Longbin Lai, Xue Li, Yong Li, Zhao Li, Zhengping Qian, Chao Tian, Lei Wang, Jingbo Xu, et al. 2021. GraphScope: a unified engine for big graph processing. *PVLDB* 14, 12 (2021), 2879–2892.
[6] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. 2011. Adding regular expressions to graph reachability and pattern queries. In *ICDE*.
[7] Xiyang Feng, Guodong Jin, Ziyi Chen, Chang Liu, and Semih Salihoğlu. 2023. Kùzu graph database management system. In *CIDR*.
[8] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-generation database interfaces: A survey of llm-based text-to-sql. *TKDE* (2025).
[9] Fatma HÖzcan, Abdul Quamar, Jaydeep Sen, Chuan Lei, and Vasilis Efthymiou. 2020. State of the art and open challenges in natural language interfaces to data. In *SIGMOD*.
[10] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Complex knowledge base question answering: A survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 11 (2022), 11196–11215.
[11] Abiram Mohanaraj, Matteo Lissandrini, and Katja Hose. 2025. Planrgcn: predicting sparql query performance. *PVLDB* 18, 6 (2025).
[12] Jason Priem, Heather Piwowar, and Richard Orr. 2022. OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *arXiv preprint* (2022).
[13] 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. *arXiv preprint* (2019).