2023 IEEE 39th International Conference on Data Engineering (ICDE) | 979-8-3503-2227-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDE55515.2023.00134

GALE: Active Adversarial Learning for Erroneous Node Detection in Graphs

Sheng Guan
Case Western Reserve Univ.
sxg967@case.edu

Hanchao Ma
Case Western Reserve Univ.
hxm382@case.edu

Mengying Wang
Case Western Reserve Univ.
mxw767@case.edu

Yinghui Wu
Case Western Reserve Univ.
yxw1650@case.edu

***Abstract*—We introduce GALE, an active adversarial learning framework to detect nodes with erroneous information in attributed graphs. GALE is empowered by a new adversarial active error detection framework, which interacts active learning with a graph generative adversarial model to best exploit limited labeled examples of erroneous nodes. It dynamically determines diversified query nodes in batches with bounded size in terms of node typicality to enrich a pool of examples, which in turn provides representative examples to best train an adversarial classifier to capture different types of errors. Moreover, GALE provides an annotation algorithm to suggest a context of possible correct attribute values and error types, to facilitate the labeling of query nodes. We show that using limited queries and examples, GALE significantly improves competing methods such as constraint-based detection, outlier detection, and Graph Neural Networks (e.g. GCNs), with 32%, 31%, and 17% gain in F-1 score on average, and is feasible in learning cost for large graphs.**

I. Introduction

Error detection in attributed graphs is critical for high-quality graph data and downstream analysis in *e.g.,* knowledge graphs and social networks [44]. In practice, there often exist multiple types of errors in the node attribute values. Several methods have been studied to detect such erroneous nodes.

(1) Errors can be defined as violations of data constraints [18], anomalies [37], or inferred with statistical models [40]. These methods can be accurate for specific types of errors, yet may not achieve good recall to capture multiple types of errors [22].

(2) Node classifiers [22], [43] can be trained from examples of errors, by solving a node classification problem. Another approach is to train individual "base detector" with ensemble learning [57] to improve the recall. This often requires sufficient training examples from matching class of errors [56]. High-quality labeled examples require heavy manual effort, and remain a luxury for error detection in real-world scenarios.

Consider the example below from a real knowledge graph. Erroneous nodes are chosen from their real editing history[1].

Example 1: Fig. 1 illustrates a fraction of a knowledge graph about films (series) from Wikipedia. A node carries a type (*e.g.,*film) and a set of attributes (*e.g.,* "*name*") with values (e.g., "*Avengers: Infinity War*"). An edge denotes a relationship between nodes (e.g.,"*subsequent*'). There are four erroneous nodes with the type "film" (with the errors marked in red).

- *Case 1*: The Film node v_2 has an incorrect release year "2014", which should be "2015";

[1]https://www.wikidata.org/w/index.php?title=Q1765358&action=history

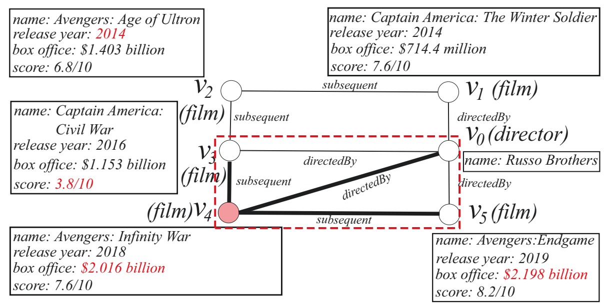


Fig. 1: Detecting Erroneous Nodes in Knowledge Graphs

- *Case 2*: The Film node v_3 has a wrong rate score "3.8/10" that should be "7.7/10";
- *Case 3*: The Film node v_4 has an inaccurate record of $2.048 billion at the worldwide box office; instead, a correct value should be $2.016 billion;
- *Case 4*: Film v_5 makes $2.798 billion at the worldwide box office instead of $2.198 billion, which is inaccurate.

Consider the following options.

<u>*Data constraints.*</u> Rules and dependencies that enforce equivalence on node attribute values [18] may not capture *Case 1*, as the true value "2015" cannot be inferred via node equivalence or is hard to be specified by rules for an individual node. A "negative" rule [8] may specify that "*if two films are connected by 'subsequent' relation, then their release years must be different*". Nevertheless, either v_1 or v_2 can have a wrong release year "2014", which cannot be distinguished by the rule due to the vagueness of undirected "subsequent" relation.

<u>*Outlier detection*</u> [37], [45] may capture *Case 2* error at node v_3, given its significantly low value (an outlier) over the attribute *"score"*. Nevertheless, the errors of the box office (*Case 3* and *Case 4*) at node v_4 and node v_5 cannot be easily captured by an outlier detection process, since both values are in a "normal" range of value distribution.

A single approach may not capture all the errors. Stacking them to a sequential process, or using simple ensemble strategies (*e.g.,* union or voting) may improve the recall but not the overall accuracy due to overlap or false positives [1], [22].

<u>*Train a classifier [12], [22].*</u> One may train a node classifier to distinguish correct and erroneous nodes given their learned representation ("node embeddings"). For example, films v_5 and v_4 are "close" in terms of similar embeddings from

2375-026X/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDE55515.2023.00134

"score", "release year" and *"box office gross"*. A classifier may infer that v_4 is likely to contain errors given v_5 as an example of an erroneous node [12], [22]. Learning-based methods, on the other hand, require a sufficient set of labeled examples. This can be hard to build from scratch. □

It has been verified that active learning [16], [42] can improve error detection especially when training examples are sparse. For example, one may choose a few nodes to query an oracle (*e.g.,* a data analyst) for a label ("correct" or "error"), and update the classifier to detect errors given the enriched examples. Although desirable, challenges remain for active learning-based error detection, especially for graphs.

Query Selection. Effective active learning often requires proper choices of queries [34], [48], [48]. This is in particular nontrivial for active learning in graph data, as traditional sampling strategies may fail [38]. Moreover, there are often more than one type of error in real-world graphs [1], [22], [44], [49]. *How to choose a representative set of queries* for the diversified error types to best exploit the oracle, especially with limited or even no labels to start with?

Labeling Effort. Most active learning assumes an oracle with the full capability to provide correct labels. In practice, it is not easy to obtain a true label by inspecting the query node and its attribute values alone. (1) The low-quality labels remain a major issue [13]. (2) Even with the new labels obtained from the oracles, the labeled nodes ("training examples") may still be sparse and biased. Adversarial learning has been verified to be effective by augmenting training examples with synthetic ones [22]. We are interested in combining active learning and adversarial detection to best exploit the limited examples.

Example 2: A desirable approach allows us to "cold start" error detection even when no example is available. One may iteratively choose unlabeled nodes and query an oracle (a validated detection method or a human expert) and obtain examples to *improve* a node classifier. (1) An initial round captures v_3 as an erroneous node (with *e.g.,* outlier detection). (2) Observe that (a) $\{v_1, v_3, v_4, v_5\}$ forms a cluster of films directed by the same director, and (b) v_4 is closer to the "center" of the cluster in terms of node similarity. Choosing $\{v_2, v_4\}$ as a query set improves the decision boundary of the classifier, which in turn distinguishes v_5 and v_1 as erroneous and correct nodes, respectively. (3) Moreover, the neighbors of v_4 contain rich information (highlighted in Fig. 1) and can be annotated with error types and statistics to reduce the labeling effort and optimize the choice of follow-up queries. □

These call for an effective detection framework with the following desirable properties. (1) It automatically selects representative query nodes that, if labeled, can best improve a classifier in distinguishing erroneous nodes from correct ones. (2) The query set properly covers diversified classes of errors. As suggested by the above example, the choice should balance both the diversity and the coverage of different types of errors. (3) The oracle is consulted by a *bounded* number of queries. This requires effective exploitation of limited examples via *e.g.,* data augmentation [51]. (4) Moreover, it actively provides useful auxiliary information to help the (human) oracle to derive the labels for "hard" queries. *Can we enable an iterative error detection framework with all these desirable properties?*

Contribution. This paper introduces **GALE**, a **G**raph **A**ctive **L**earning framework for **E**rrorenous node detection. Our approach is to *improve* a node classifier $\mathcal{M}$ for error detection, by building an iterative framework that interacts active learning, data augmentation, with adversarial learning that can maximally exploit the labeled examples obtained from an oracle. GALE has the following novel features.

An Active Adversarial Framework (Section III and IV). GALE interacts active learning with adversarial learning in a single framework, where a node classifier $\mathcal{M}$ is continuously improved by examples of erroneous nodes that are obtained and enriched from a query generator. By querying high-quality nodes for training $\mathcal{M}$, GALE improves error detection with gained new knowledge. By casting error detection as a "two-players game", GALE further improves the classifier with augmented examples. These allow GALE to cold start error detection without initial examples, achieve desirable performance in both precision and recall using a small amount of examples, and for capturing multiple types of errors.

Query Selection with Diversified Typicality. (Section V). GALE admits high-quality query nodes that are both representative ("typicality") and cover different type of errors. We formulate the intuition in Example 2 as a bi-criteria optimization problem captured by a diversified typicality function, and introduce a 2-approximation algorithm to approach the best queries. It then dynamically selects query nodes in small batches, to continuously improve the decision boundary of the classifier $\mathcal{M}$.

Query Annotation (Section VI). GALE also extracts useful auxiliary information and annotates the query nodes and their neighbors. We introduce an annotation algorithm to track such information. These information help users to verify and label the query nodes, understand possible error types, and facilitate further inspection for query selection and data repairing.

System Evaluation (Section VII and Section VIII). We have developed a prototype system GALE that nontrivially enhances a pilot prototype [22] with the active adversarial framework. Our experiments using real-world graphs verified the effectiveness of GALE. It improves GEDet with a gain, on average, of 4.98% and 2.2% in recall and F-1 score, respectively, with a small batch of queries in only 5 iterations and additional training cost less than 300 seconds (see details in Section VIII).

Related Work. We categorize the related work as follows.

Error detection in graphs. While error detection for relational data has been extensively studied [26], effective error detection in emerging graph data is much less addressed. Several methods have been proposed to detect errors in graphs. (1) Data dependencies are extended to graphs [18], [28], [36] to capture errors as violations of value constraints, contextualized by graph patterns. (2) Embedding-based methods [12], [22] exploit node and topological features to learn node classifiers to detect errors in graphs. For example, GEDet [22] utilizes semi-supervised generative adversarial networks (GAN) to detect heterogeneous errors and to improve the recall of error detection. PGE [12] jointly leverages textual and graph structures to learn embeddings of knowledge graphs for error detection. Unlike prior methods, GALE is empowered by a generative active adversarial framework, which continuously exploits new knowledge from oracles to improve error detection.

Graph Representation Learning. Graph representation learning aims to properly map nodes to low-dimensional vectors in the embedding space [11], [53]. Graph neural networks (GNNs) [21] have been verified to be effective for graph representation learning. GNNs take the topological structure and nodal attributes as input and learn encoder functions that aggregate the node's local neighborhood, with variants such as GCN [30] (which applies a first-order approximation of spectral graph convolutions), Graph autoencoders (GAE) [31], [52] (that learns embeddings to minimize reconstruction error).

Adversarial Learning. Adversarial classification [15] aims to optimize classifiers given the optimal strategy of an adversary. It treats the classifier and an adversary as "Two-players", where the classifier (discriminator $\mathcal{D}$) tries to accurately predict the classes of instances, and the adversary (generator $\mathcal{G}$) attempts to "fool" the classifier by manipulating instances. The competition leads to better classifiers [3], [41]. Generative adversarial networks (GANs) are used to generate synthetic data that approximates real data distribution by jointly learning a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$. GALE nontrivially interacts with GAE to learn node representation for error detection and with GAN to mitigate the impact of biased labels.

Active Learning. Active learning has demonstrated its success in error detection for relational data [25]. It tries to overcome the labeling bottleneck by asking an oracle, *e.g.,* a data analyst, to label informative unlabeled data ("queries") chosen by a query selection strategy, to construct or enrich the training data. As such, it maximizes the performance gain of a model with as few annotated samples as possible. The main goal is thus to sample a bounded number of queries that can best (iteratively) improve model quality. Several query selection approaches include Random sampling [48] (selecting instances uniformly in random), Entropy-based sampling (favors instances with large information entropy), and Margin-based sampling [19] (samples instances that have lowest margin between the two highest softmax outputs of embeddings). It has been observed that traditional strategies may not work well for graph data [38], due to that the choice tends to be biased to "informative" yet suboptimal queries, which are not necessarily reflecting true distribution. In contrast, GALE (1) exploits adversarial learning to learn the real distribution of errors, and (2) uses diversified typicality to choose queries that are consistent with the distribution to avoid biased selection.

The recent Clustering-based sampling has gained attention in the low-budget active learning regime, and has been experimentally verified to outperform random- and uncertainty-based methods (*e.g.,* margin-based and entropy-based sampling) [46]. It picks unlabeled examples that are nearest to the clustering centroids. We propose a new query selection strategy that maximizes typicality and diversity in terms of both node similarity and topological features, and provide a greedy algorithm that optimizes the query selection with 2-approximation. These are not addressed by prior work.

Recent effort investigates active learning for graph analysis. GraphUCB [16] issues a limited number of queries in the unsupervised setting to detect node anomalies. RIM [55] derives a selection criterion for active graph representation learning based on graph convolutional networks or Label Propagation. It mainly resolves noisy labels from oracles, by maximizing a reliable influence from consistent labels given by the models and oracles. These methods are not designed to cope with multiple error scenarios in the graphs, and thus cannot be directly applied for error detection in graphs.

Adversarial active learning. Adversarial active learning is proposed to further integrate the adversarial learning with an active learning query engine. For example, SEAL [35] proposes a semi-supervised adversarial learning structure and shows that the graph embedding network and the discriminator can work together to improve the classifiers.

To the best of our knowledge, GALE is the first framework that integrates and interacts adversarial learning, graph representation learning and active learning with limited query budget in an iterative error detection process, for effective multi-type error detection in graphs. Moreover, GALE (1) provides new query selection strategy to sample typical and representative nodes for multiple types of errors; and (2) exploits query annotation that can help human oracles label the nodes and facilitate dynamic query selection during the active learning process. These are not addressed in prior work.

II. Error Detection in Graphs

We start with several notations used by GALE.

Attributed Graphs. A graph $G = (V, E)$ consists of a set of nodes V and a set of edges $E \subseteq V \times V$. Each node $v \in V$ is a tuple $(v.A_1, \dots v.A_n)$ defined on n attributes, where $v.A_i = a_i$ ($i \in [1, n]$) denotes that the attribute A_i of v has value a_i.

GALE processes attributed graphs in their feature representation. A *feature representation* of G is a pair $G = (X_G, A_G)$. (1) X_G is a matrix of node features, where each row X_v is a vector encoding of a node tuple $v \in V$. The encoding can be obtained by *e.g.,* word embedding or one-hot encoding [20]. (2) A_G is the adjacency matrix of G.

Erroneous Nodes. We assume the existence of a "ground truth" node v^* for each node $v \in V$ that carries correct values of all the attributes of v. A node v is *erroneous* if there exists an attribute A, such that $v.A \neq v^*.A$ ($v.A$ can be 'null' that denotes a missing value). Otherwise, v is correct.

Symbol	Notation
$G = (V, E)$	Heterogeneous graph with nodes V and edges E
$V_{\mathcal{T}} = V^e \cup V^c$	$V_{\mathcal{T}}$:Labeled training nodes; V^e: erroneous nodes; V^c: correct nodes
(X_G, A_G)	feature representation of G: (X_G: feature matrix; A_G: adjacency matrix)
$\mathbf{h}_n(\mathbf{x}_v)$	the embedding of node v (at layer n)
H_n	graph embedding matrix at layer n
$\mathcal{G}$, $\mathcal{D}$	generator and discriminator of SGAN
$\mathcal{L}(\mathcal{G})$, $\mathcal{L}(\mathcal{D})$	loss function of $\mathcal{G}$ and $\mathcal{D}$
$\mathcal{L}_s$, $\mathcal{L}_u$	supervised and unsupervised loss of $\mathcal{L}(\mathcal{D})$
$\mathcal{S}$	query selector
$\mathcal{A}$	query annotator

TABLE I: Table of Notations

An *example* denotes a labeled node in G, which is a pair (v, l), where l is either 'correct' for a correct node v, or 'error' if v is an erroneous node. v is unlabeled If l is unknown.

Error Detection. We formulate error detection in a graph G=(V, E) as a node classification problem. The *training nodes* $V_{\mathcal{T}} = V^e \cup V^c$ ($V_{\mathcal{T}} \subseteq V$) is a set of examples, where V^e (resp. V^c) refers to a set of examples of erroneous (resp. correct) nodes. Given G and $V_{\mathcal{T}}$, the error detection is to train a classifier $\mathcal{M}$ that accesses G to infer the labels of a set of unlabeled nodes from $V \setminus V_{\mathcal{T}}$.

Queries and Oracles. Given a node classifier $\mathcal{M}$, GALE aims to exploit active learning with an oracle $\mathcal{O}$ to improve the performance of $\mathcal{M}$. (1) A *query* q is an unlabeled node $(v, \emptyset)$ that requests for the real label of a node v. Once assigned a label, q is converted to an example (v, 'correct') or (v, 'error'). (2) An oracle $\mathcal{O}$ can be queried and returns the true label $\mathcal{O}(q)$ of a query q. In practice, an oracle can be a human expert, or be simulated by invoking and ensembling (with *e.g.,* majority voting) a set of user-defined classifiers called *base detectors*. GALE admits a library of such detectors Ψ.

III. Framework Overview

A. An Adversarial Active Learning Framework

Error Detection as "Two-players Game". GALE casts error detection into a two-players game between a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$, and derives the node classifier $\mathcal{M}$ from $\mathcal{D}$. It exploits adversarial detection, with the following principle.

- A generator $\mathcal{G}$ aims to "fool" the discriminator $\mathcal{D}$ by representing G with synthetically generated representations that as close to their true counterparts as possible.
- The discriminator $\mathcal{D}$ meanwhile learns to distinguish nodes with real labels from the synthetic ones from $\mathcal{G}$.

A "competition" between $\mathcal{G}$ and $\mathcal{D}$ *should* improve the performance of both: $\mathcal{G}$ learns how to better simulates the distribution of the real labels with synthetic ones, and $\mathcal{D}$ learns the node embeddings that better discriminates the synthetic errors, real errors, and correct nodes.

Intuition. Let us consider a labeled set $L = \{(x, y)\}$, where x represents a node and $y \in \{$'$error$'$(y = 1)$, '$correct$' $(y = 2)\}$. GALE introduces a *third type of label* called 'synthetic error' $(y = 3)$ to annotate nodes with synthetic errors from $\mathcal{G}$. We then enforce a learning objective of $\mathcal{D}$ in the form of:

$$\max_{\mathcal{D}} \mathbb{E}_{x\sim\mathbb{L}} \log P_{\mathcal{D}}(y|x, y \leq 2) + \mathbb{E}_{x\sim p} \log P_{\mathcal{D}}(y \leq 2|x) + \mathbb{E}_{x\sim p_{\mathcal{G}}} \log P_{\mathcal{D}}(3|x) \quad (1)$$

where p and $p_{\mathcal{G}}$ refers to the distribution of the real data and its synthetic counterpart from $\mathcal{G}$, respectively. The first, second, and third terms are to maximize the log conditional probability for (1) the labeled nodes with true labels (examples), (2) the unlabeled nodes (as either 'error' or 'correct'), and (3) the generated synthetic examples ('synthetic error'). As such, $\mathcal{D}$ is able to better distinguish synthetic errors, real errors, and real correct nodes. This yields more accurate classifiers $\mathcal{M}$.

Joint Learning Scheme. In a nutshell, GALE improves $\mathcal{G}$ and $\mathcal{D}$ in a joint learning process computes a node embedding matrix (derived from the classification layer of $\mathcal{D}$) that minimizes a weighted combination of supervised loss (from examples $V_{\mathcal{T}}$) and an unsupervised (feature matching) loss. A node classifier $\mathcal{M}$ is yielded by converting the matrix (via a softmax function) to a matrix of class probabilities for the unlabeled nodes in G. The label with largest probability is chosen for each node (as either "error" or "correct" for error detection).

Graph Augmentation. This scheme has been verified to be effective for error detection in graphs [22], [23]. The specific scheme is enabled by a *graph augmentation* process [22], which (1) leverages a library Ψ of base detectors (rules, constraints or string transformations) to inject synthetic errors, and (2) convert the augmented graph to a compact featurized encoding (Section II) by learning a graph autoencoder GAE of G, yielding representations of real data X_R and synthetic data X_S, respectively. The generator and discriminator then play a competitive game over the compact encodings to improve the decision boundary by specifying and optimizing the above objective. GALE adapts the above scheme to iteratively update the discriminator $\mathcal{D}$ and the classifier $\mathcal{M}$ upon receiving new examples from the oracle $\mathcal{O}$ (see Section IV).

Active Adversarial Detection. Despite its effectiveness [22], [23], the above scheme requires sufficient high-quality examples. Moreover, the learning of $\mathcal{M}$ is a "one-shot" process, and cannot evolve upon new error types. To adapt $\mathcal{M}$ to sparse, biased and possibly new examples, GALE interacts active learning and adversarial learning in an *iterative* framework.
(1) Given an oracle $\mathcal{O}$, graph G, and a current classifier $\mathcal{M}$, it chooses from unlabeled nodes a set of queries and consults $\mathcal{O}$ to obtain their labels. The selection of the queries follows a general strategy that favors nodes that properly represents multiple error types in G. This enriches a pool of examples that are in turn used by the adversarial detection scheme. GALE then updates the discriminator $\mathcal{D}$ in the "two-players game", and derives the classifier $\mathcal{M}$ accordingly from $\mathcal{D}$.
(2) The learned classifier $\mathcal{M}$ is applied to detect new erroneous nodes in G. This in turn improves the quality of future query selection, mitigating the bias of active learning [38].

The above steps runs in multiple iterations. As such, GALE allows us to query the oracle and gain new knowledge that

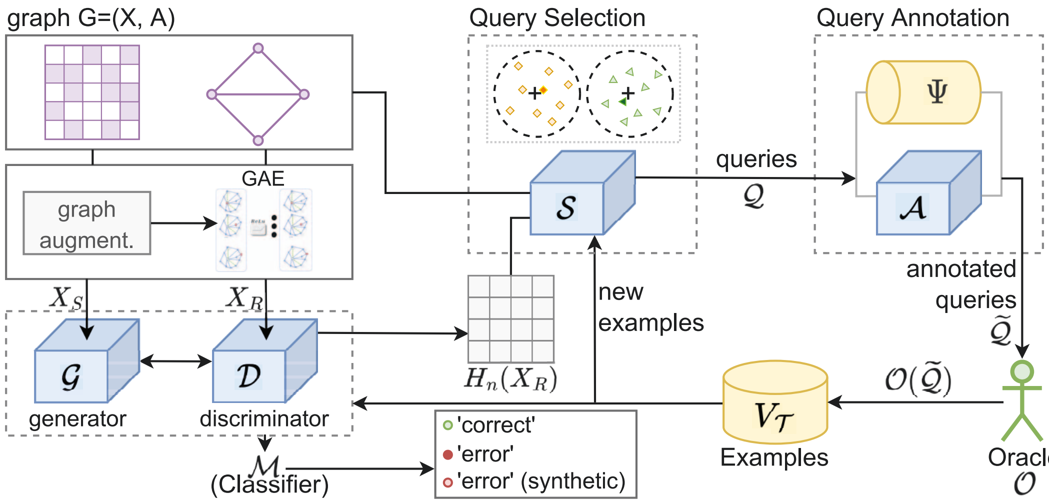


Fig. 2: Overview of GALE Framework

approach the real distribution of unknown erroneous nodes, thus improving the accuracy of error detection by continuously refining the decision boundary of $\mathcal{M}$.

B. Modules and Models

We next introduce the major modules in GALE. These modules and associated models specify the principled adversarial active framework into an *iterative* error detection process.

Semi-supervised Adversarial Module. *How to maintain and improve a node classifier $\mathcal{M}$ for error detection as "two-players game"?* GALE specifies the adversarial training of $\mathcal{M}$ in an *iterative semi-supervised* generative adversarial module (SGAN). This module is responsible for dynamically maintaining $\mathcal{M}$ upon new examples from the oracle $\mathcal{O}$ for error detection. Given a set of new examples, it works in two stages.

Initialization. SGAN initializes the joint learning of $\mathcal{G}$ and $\mathcal{D}$ from scratch, and derives the classifier $\mathcal{M}$ from $\mathcal{D}$. SGAN takes as input the node representation from graph G that includes synthetic erroneous node features X_S and real example features X_R via the graph augmentation process (see Section IV). The classification layer of $\mathcal{D}$ predicts the class label for unlabeled nodes ("real" or "error").

Update $\mathcal{D}$. Upon receiving changes in X_R due to the new examples obtained from the oracle $\mathcal{O}$, SGAN incrementally updates $\mathcal{D}$ by updating its parameters in response to the changed examples $V_\mathcal{T}^i$. $\mathcal{M}^i$ is derived accordingly.

That is, $\mathcal{D}^i$ is maintained at iterations i as follows:

$$(\mathcal{G}, \mathcal{D}^0) = \mathsf{SGAN}(G, V_\mathcal{T}^0, X_R, X_S); \quad \mathcal{D}^i = \mathsf{SGAN}(G, V_\mathcal{T}^i, X_R, X_S)$$

The module also forwards $H_n^i(X_R)$, the node embeddings learned on an intermediate n-th layer of the discriminator $\mathcal{D}$ to Query Selection module $\mathcal{Q}$ to facilitate the selection of queries from unlabeled nodes in future iterations.

Query Selection Module. *How to select a small set of queries to improve error detection?* GALE aims to improve $\mathcal{M}$ by obtaining additional labels from oracle $\mathcal{O}$. The query selection module utilizes a Query Selector $\mathcal{S}$ to generate promising queries from the unlabeled training node set, and query $\mathcal{O}$ for additional true labels. The model $\mathcal{S}$ initializes k queries with clustering-based sampling [46] from unlabeled nodes. Here k is a *local budget* that specifies the number of queries allowed per iteration. In the follow-up iteration i, it exploits the node embeddings $H_n^{i-1}(X_R)$ learned by $\mathcal{D}^{i-1}$, and examples $V_\mathcal{T}^{i-1}$ from the last iteration, and returns query set $\mathcal{Q}^i$ with bounded size k, with a goal to represent diversified errors.

$$\mathcal{Q}^0 = \mathcal{S}(\emptyset, \emptyset, G, k); \quad \mathcal{Q}^i = \mathcal{S}(H_n^{i-1}(X_R), V_\mathcal{T}^{i-1}, G, k)$$

The queries are sent to be enriched with available auxiliary information (see Query Annotation Module), assigned with true labels (thus converted to examples $\Delta V_\mathcal{T}$), and featurized to enrich the example features X_R to improve $\mathcal{M}$.

Query Annotation Module. *How to adapt active adversarial framework for iterative error detection?* To escape from locally biased choices [38], GALE dynamically chooses promising queries by extracting and aggregating useful annotated information of the queries $\mathcal{Q}$, and enrich $\mathcal{Q}$ to an *annotated* counterpart $\tilde{\mathcal{Q}}$. By default, GALE collects and encodes the following information, whenever applicable: 1-hop neighborhood induced subgraph of $\mathcal{Q}$, most influential labeled nodes to $\mathcal{Q}$, degree assortativity [38], erroneous attribute values detected by base detectors from Ψ [22], among others (see Section VI). The benefit is twofold: it provides useful local and global structural information of query nodes in G that facilitate the oracle $\mathcal{O}$ for labeling, and allow query selector $\mathcal{S}$ to re-estimate the node importance for dynamic selection in future iterations.

Specifically, a query annotator $\mathcal{A}$ provides $\mathcal{O}$ and $\mathcal{S}$ with annotated $\mathcal{Q}$ in each iteration as follows:

$$\tilde{\mathcal{Q}}^0 = \mathcal{A}(\mathcal{Q}^0, \Psi, G); \quad \tilde{\mathcal{Q}}^i = \mathcal{A}(\mathcal{Q}^i, \Psi, G)$$

The module also naturally augments the examples with the new ones by querying the oracle $\mathcal{O}$ with the queries $\mathcal{Q}$:

$$\boxed{\begin{aligned} V_{\mathcal{T}}^0 &= \mathcal{O}(\tilde{\mathcal{Q}}^0); \\ V_{\mathcal{T}}^i &= V_{\mathcal{T}}^{i-1} \cup \mathcal{O}(\tilde{\mathcal{Q}}^i) \end{aligned}}$$

C. *Workflow of* GALE

Putting the modules together, we next describe the workflow of GALE (illustrated in Fig. 2, with an algorithmic description of the learning process in Fig. 3). It works with a graph G, an oracle $\mathcal{O}$, and (optionally) a library of base detectors Ψ.

Initialization. GALE "cold starts" the framework by initializing a set of queries $\mathcal{Q}$ with annotated information from Ψ, which provides an initial set of detected errors in attributes, whenever applicable (lines 1-2). It then consults an initial verification of $\mathcal{Q}$ with oracle $\mathcal{O}$ (line 3) to initialize the examples. It also invokes a procedure GAugment to preprocess G and generates the feature representation of real (resp. synthetic) examples X_R (resp. X_S) following [22], to enable the adversarial learning of the generator $\mathcal{G}$ and discriminator $\mathcal{D}$. This yields an initial node classifier $\mathcal{M}$ (lines 5-6).

Iterative Improvement. At iteration i, GALE interacts the modules as follows. (1) With the learned embeddings $H_n(X_R)$ of the real examples from $\mathcal{D}$ in the last round, it selects a representative query set $\mathcal{Q}^i$ (line 8), and enrich the queries with their auxiliary annotated information (line 9). (2) It consults the oracle $\mathcal{O}$ with the batch of new queries and augments a sampled set $\tilde{V_{\mathcal{T}}}$ from initially labeled examples $V_{\mathcal{T}}$ as $V_{\mathcal{T}}'$. The purpose of sampling is to let the representative query set $\mathcal{Q}^i$ weigh more when updating the classifier $\mathcal{M}$ compared to using the whole $V_{\mathcal{T}}$ (line 10-11). (3) The adversarial learning is then activated to update $\mathcal{D}$ with newly updated examples; The classifier $\mathcal{M}$ along with the learned embeddings are then updated accordingly (lines 12-13). Here the procedure SGAND is a variant of SGAN. Instead of retraining both $\mathcal{G}$ and $\mathcal{D}$ from scratch, it incrementally updates $\mathcal{D}$ in response to the necessary change of the examples $V_{\mathcal{T}}$ (see Section IV).

The above process repeats up to (user-defined) T iterations, and issues at most $T \cdot k$ queries. The improved classifier $\mathcal{M}$ is then returned (line 14). In practice, GALE can be "interrupted" at any iteration to respond to error detection with a current M.

We next introduce the details of iterative semi-supervised GAN mododule, query selection module and annotation module, in Sections IV,V, and VI, respectively.

Algorithm GALE
Input: $G(V, E)$, library Ψ, integers k, sampling rate η and T;
Output: (improved) SGAN classifier $\mathcal{M}$.
1. set $V_{\mathcal{T}} := \emptyset$; set $\mathcal{Q} := \emptyset$; integer i:=1;
   matrix $X_S := \emptyset$, matrix $X_R := \emptyset$; matrix $H_n := \emptyset$;
   / preprocess G and initialize models */*
2. $\mathcal{Q} := \mathcal{S}(\emptyset, \emptyset, G, k)$; $\tilde{\mathcal{Q}} := \mathcal{A}(\mathcal{Q}, \Psi, G)$;
3. $V_{\mathcal{T}} := \mathcal{O}(\tilde{\mathcal{Q}})$;
4. (X_R, X_S) := GAugment (G,Ψ);
5. $(\mathcal{G}, \mathcal{D})$:= SGAN $(G, V_{\mathcal{T}}, X_R, X_S)$;
6. derive an initial classifier $\mathcal{M}$ and H_n from discriminator $\mathcal{D}$;
   / iteratively improve the classifier */*
7. **while** $i < T$ **do**
8. set $\mathcal{Q}^i := \mathcal{S}(H_n(X_R), V_{\mathcal{T}}, G, k)$;
9. set $\tilde{\mathcal{Q}^i} := \mathcal{A}(\mathcal{Q}^i, \Psi, G)$;
10. set $\tilde{V_{\mathcal{T}}}$:= sample($V_{\mathcal{T}}$, η)
11. set $V_{\mathcal{T}}' := \tilde{V_{\mathcal{T}}} \cup \mathcal{O}(\tilde{\mathcal{Q}^i})$;
12. $\mathcal{D}^i$:= SGAND$(G, V_{\mathcal{T}}', X_R, X_S)$;
13. update $\mathcal{M}$ and H_n from $\mathcal{D}^i$;
14. **return** updated $\mathcal{M}$;

Fig. 3: GALE: Learning Framework

IV. Iterative Adversarial Detection

SGAN enforce $\mathcal{G}$ and $\mathcal{D}$ to compete in a two player's game, yet assumes fixed examples. We formulate an optimization problem to minimize a bi-criteria loss in an *iterative* semi-supervised setting. We use the following construction.

Loss Function of $\mathcal{D}$. To adapt the discriminator $\mathcal{D}$ to recognize new erroneous nodes and differentiate real and synthetic labels, we quantify the loss of $\mathcal{D}$ as $\mathcal{L}^i(\mathcal{D}) = \mathcal{L}_s^i + \lambda\mathcal{L}_u$, where (1) a *supervised loss* $\mathcal{L}_s^i$ quantifies the loss of accuracy on node label classification ('error' or 'correct') for a *current* set of examples $V_{\mathcal{T}}^i$; and (2) an *unsupervised loss* $\mathcal{L}_u$ quantifies the loss of accuracy on distinguishing synthetic or real labels.

Unlike [22] that assume a fixed learning objective, we guide the learning of $\mathcal{D}$ to minimize the supervised and unsupervised loss given the current examples and latest error distribution.

Supervised Loss. We define the supervised loss as:

$$\mathcal{L}_s^i = \sum_{v \in V_{\mathcal{T}}^i} \ell(f_\theta(\mathrm{x}_v), l_v)$$

where $V_{\mathcal{T}}^i$ is the i-th batch of the examples obtained from the oracle $\mathcal{O}$, with l_v the ground truth label of query v. θ is the learnable parameter and $f_\theta(\mathrm{x}_v)$ is the prediction of node v. $\ell(\cdot, \cdot)$ measures the difference between prediction and ground truth label using *e.g.,* cross entropy.

Unsupervised Loss. The discriminator $\mathcal{D}$ also aims to classify the real or synthetic examples as accurately as possible by minimizing the unsupervised loss $\mathcal{L}_u$. We define $\mathcal{L}_u$ as:

$$\mathcal{L}_u = -\mathbb{E}_{x_v \sim X_R}[\log(1 - \mathcal{D}(x_v))] - \mathbb{E}_{x_v \sim \mathcal{G}(X_S)}[\log \mathcal{D}(x_v)]$$

where $\mathcal{D}(x_v)$ is the predicted synthetic probability of v.

Loss Function of $\mathcal{G}$. The generator $\mathcal{G}$ aims to minimize the difference between the synthetic graph embeddings X_S and the real counterpart from X_R. We define the loss function $\mathcal{L}(\mathcal{G})$ of $\mathcal{G}$ to measure the feature matching loss [47].

$$\mathcal{L}(\mathcal{G}) = \left\| \mathbb{E}_{\mathbf{x}_v \in X_R} \mathbf{h}(\mathbf{x}_v) - \mathbb{E}_{\mathbf{x}_v' \in X_S} \mathbf{h}(\mathcal{G}(\mathbf{x}_v')) \right\|_2^2$$

where $\mathbb{E}(\cdot)$ computes the expected value of a graph feature matrix. The generator is trained to match the expected value of the embeddings on an intermediate layer of $\mathcal{D}$ between the features of real examples $\mathbf{h}(\mathbf{x}_v)$ and synthetic erroneous counterpart $\mathbf{h}(\mathcal{G}(\mathbf{x}_v'))$. Intuitively, $\mathcal{G}$ has higher chance to fool

Procedure: SGAN $(G, V_{\mathcal{T}}, X_R, X_S)$
1. **initialize** the parameters of $\mathcal{G}$ and $\mathcal{D}$;
2. update $\mathcal{L}(\mathcal{D})$ with $V_{\mathcal{T}}$ X_R, and X_S;
3. **while** $\mathcal{G}$ and $\mathcal{D}$ do not reaches a Nash equilibrium **do**
4. update $\mathcal{G}(\cdot)$ by descending gradients of losses:
5. $\nabla_{\theta_{\mathcal{G}}}\mathcal{L}(\mathcal{G})$
6. update $\mathcal{D}(\cdot)$ by descending gradients of losses:
7. $\nabla_{\theta_{\mathcal{D}}}\mathcal{L}(\mathcal{D})$
8. reduce learning rate β
9. **return** $\mathcal{D}$;

Procedure: SGAND $(G, V_{\mathcal{T}}, X_R, X_S)$
1. update $\mathcal{L}^i(\mathcal{D})$ with $V_{\mathcal{T}}$ X_R, and X_S;
2. Update $\mathcal{D}(\cdot)$ by descending gradients of losses:
3. $\nabla_{\theta_{\mathcal{D}}}\mathcal{L}(\mathcal{D})$
4. **return** $\mathcal{D}$;

Fig. 4: Procedure SGAN and SGAND

the discriminator $\mathcal{D}$ by faking node embeddings closer to the real counterpart, by minimizing $\mathcal{L}(\mathcal{G})$.

The models $\mathcal{G}$ and $\mathcal{D}$ consist of a sequence of transpose convolution and batch normalization layers, adding regularization layers *e.g.,* dropout layers to prevent overfitting. $\mathcal{D}$ takes real feature matrix X_R and its synthetic counterpart X_G that is generated from $\mathcal{G}$ as inputs. Letting $\mathbf{h}_n(\mathbf{x}_v)$ be the embeddings on an intermediate layer of $\mathcal{D}$. $H_n^i(X_R)$ refers to the learned real node embeddings for real feature matrix X_R at iteration i. The matrix $H_n^i(X_R)$ is then passed to Query Selection module.

Procedures SGAN **and** SGAND. GALE trains generator $\mathcal{G}$ and discriminator $\mathcal{D}$ by iteratively minimizing their corresponding losses in procedure GANDet, as shown as Fig. 4. The main training loop (lines 3-8) jointly optimize the generator loss $\mathcal{L}(\mathcal{G})$ and discriminator loss $\mathcal{L}(\mathcal{D})$. We optimize the gradients by applying Adam [29] and update $\mathcal{D}$ and $\mathcal{G}$.

We distinguish SGAND with SGAN, a variant that follows the joint learning scheme as in SGAN but only maintains the discriminator $\mathcal{D}$, by iteratively minimizing the corresponding supervised loss. It *incrementalizes* the learning of SGAN, by performing necessary updates to $\mathcal{D}$ in response to the change of the *supervised loss* $\mathcal{L}^i(\mathcal{D})$ (line 1) due to enriched $V_{\mathcal{T}}$.

V. Query selection Module

We next describe the query selection strategy of GALE. We first introduce a notion of *diversified typicality* to measure the quality of the queries in terms of "typicality" and diversity.

A. Diversified Typicality

Clustering Typicality. The *typicality* of a query in active learning strategies [24], [46] measures the quality of a query in terms of its closeness to the centroid of a proper clustering.

An intuition is that when the clustering is treated as a density function mapping, the data point that is closer to the centroid indicates that it falls into a high density region that is easier to be classified. It has been observed that such queries [24], [46] achieves good performance especially in the low-budget regime (where only limited queries are allowed). Example 2 consistently illustrates this intuition. Based on this intuition, we define the *clustering typicality* of a node v as follows:

$$\mathrm{clusT}(v) = \left(\|\mathbf{h}(v) - c(v)\|_2\right)^{-1}$$

where (1) $c(v)$ is the centroid of the cluster that v belongs to, where the clustering is performed on the node embedding space, and (2) $\mathbf{h}(v)$ refers to the embedding of v. The embedding is readily obtained from $H_n^i(X_R)$ at the i-th round of execution of SGAND (line 11 of Fig. 3). Intuitively, it computes the inverse of the Euclidean distance of the node embeddings between v and its cluster centroid.

Topological Typicality. The influence of topology plays an important role in message-passing based graph learning [10], [38], [50]. Given an unlabeled training node $v \in V$, the label of v is influenced by the labels of other nodes that can reach v via aggregation through edges. This can be characterized by label propagation [50], approximated by performing a random walk from the labeled nodes to v to derive an aggregated "soft label". The node v bears "influence conflict" [10], if other nodes with different labels are likely to cause a "conflict" to the soft label. The smaller the likelihood is, the closer v is to the center of its topologically determined label class.

Given an unlabeled node v, the topological typicality of v, denoted as $\mathrm{topoT}(v))$, is defined as:

$$\mathrm{topoT}(v) = 1 - \mathbb{E}_{x \sim \boldsymbol{P}_{v,:}}\left[\sum_{l \in [1,2], l \neq L_s(v)} \frac{1}{|\mathcal{C}_l|} \sum_{i \in \mathcal{C}_l} \boldsymbol{P}_{i,x}\right]$$

where $l \in \{$'error':1, 'correct':2$\}$, which is predicted by discriminator $\mathcal{D}$; $\mathcal{C}_l$ denotes the unlabeled training node set with predicted label l. $|\mathcal{C}_l|$ denotes the total number of nodes in $\mathcal{C}_l$. The normalization item $1/|\mathcal{C}_l|$ is added to make the influence from the "error" and "correct" classes comparable when computing $\mathrm{topoT}(v)$. $L_s(v)$ denotes the estimated class type of node v with soft-label obtained in Label Propagation algorithm [58], and $\boldsymbol{P}_v$ is the Personalized PageRank probability vector for node v, which indicates a distribution of influence exerted outward to all the nodes in G from node v.

Updating soft labels. In each iteration, GALE maintains the soft labels for unlabeled nodes as follows.

- $L_s^0(v) = \arg\max_j \boldsymbol{Y}^0$, where $\boldsymbol{Y}^0$ is the initial label distribution from X_R;
- $L_s^i(v) = \arg\max_j \boldsymbol{P}\boldsymbol{Y'}^i$; where (1) Y'^i is obtained by incorporating new node labels from oracles (if any) to Y^i in iteration $i-1$, and $\boldsymbol{P} = \alpha(\boldsymbol{I} - (1-\alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})^{-1}$. Here $\boldsymbol{I}$ is the identity matrix, α is the random walk restart probability, $\tilde{A} = A + \boldsymbol{I}$, A is the adjacency matrix, and $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$.

Typicality. We define the *typicality* of a node v as

$$\mathcal{T}(v) = \mathrm{clusT}(v) \cdot \mathrm{topoT}(v)$$

Given a set of queries $\mathcal{Q}$, the typicality of $\mathcal{Q}$ is defined as $\mathcal{T}(\mathcal{Q}) = \sum_{v \in \mathcal{Q}} \mathcal{T}(v)$.

```
Procedure: QSelect (H_n(X_R), V_T, G, k)
1.    set Q := ∅; set U = V \ V_T;
2.    C(U) := ClusterU(U, H_n(X_R));
3.    while |Q| < k do
4.        compute v ∈ U \ Q that maximizes B'_v(Q, C);
5.        Q = Q ∪ {v}; U = U \ Q;
6.    return Q ;
```

Fig. 5: Algorithm QSelect

Remark. Clustering-based strategies [24], [46] captures typicality by choosing queries that are close to their cluster centroids, yet ignores the impact via the topological properties. PageRank centrality [38] measures the level of topological influence of a node and favors nodes with large PageRank centrality. Nevertheless, node attribute values are ignored [38]. We introduce diversity typicality that (1) integrates clustering typicality, node attribute embeddings (with feedback from $H_n^i(X_R)$), and topological influences as a whole; (2) dynamically determines node importance in each iteration. This makes GALE more robust to various error distributions and as new examples arrive, as verified in Section VIII.

Diversified Typicality Selection. Given a graph G with unlabeled training node set U, a graph node embedding $H_n(X_R)$, and query budget k, the *diversified typicality selection* problem is to compute a set of queries $\mathcal{Q} \subseteq U$ such that

$$\mathcal{Q} = \underset{|\mathcal{Q}'|=k}{\arg\max}(\mathcal{T}(\mathcal{Q}') + \lambda \sum_{v,v' \in \mathcal{Q}'} d(\mathbf{h}(v), \mathbf{h}(v'))) \quad (2)$$

Here $d(\cdot)$ computes the Euclidean distance of the node embeddings of v and v'.

While the diversified typicality selection problem is NP-hard, we next present an efficient query selection algorithm.

B. Query Selection Algorithm

We present an algorithm, denoted as QSelect, to implement the query selector $\mathcal{S}$ (line 8 in Fig. 3). We focus on the query selection per iteration.

Lemma 1: *Given a graph G with unlabeled node set U, a graph node embedding $H_n(X_R)$, a query budget k, there exists a 2-approximation algorithm that selects a set of query $\mathcal{Q}$ for the diversified typicality selection problem.* □

Proof sketch: We show that our diversified typicality selection problem can be reduced to the max-sum p-diversification problem [6]. The latter is to choose a size-p subset S from a set that maximizes the sum of a utility function and pairwise distances of the elements in S. We show that $\mathcal{T}(\mathcal{Q})$ is a monotone (submodular) set function, and there is an approximation preserving reduction to the max-sum p diversification. □

Algorithm. We next outline the algorithm QSelect (Fig. 5). It first performs a clustering algorithm ClusterU over the node features $H_n(X_R)$, and induces the clusters $\mathcal{C}(U)$ for the unlabeled nodes U (line 2). It then adopts a greedy strategy to prioritize the selection of an unlabeled node v that has the largest marginal gain $\mathcal{B}'_v(\mathcal{Q}, \mathcal{C}) = F_v(\mathcal{Q}) + \lambda \sum_{v,v' \in \mathcal{Q}} d(\mathbf{h}(v), \mathbf{h}(v'))$, where $F_v(\mathcal{Q}) = \frac{1}{2}\mathcal{T}(\mathcal{Q} \cup \{v\})$ - $\frac{1}{2}\mathcal{T}(\mathcal{Q})$. By default, GALE implements ClusterU with k'-means clustering with k' between k and $3k$.

```
Procedure: QAnnotate (Q, Ψ,G)
    Q̃ := ∅;
    for each v ∈ Q do
        map v.M := ∅;
        GetAnnotate (v, G, Ψ); /* Type 1*/
        for each base detector f ∈ Ψ do
            detect erroneous attribute values; /*Type 2*/
            infer correct values at v; /*Type 3*/
        update error distribution Π_ψ at v; /*Type 4*/
        update v.M; Q̃ := Q̃ ∪ {v};
    return Q̃;
```

Fig. 6: Algorithm QAnnotate

The 2-approximability can be verified by the approximation preserving reduction to the max-sum p-dispersion problem. For the time cost, it takes $O(|U|k'm)$ times for m rounds of k'-means grouping. The cost of the greedy selection process is in $O(k|U|)$ time [4], [6]. The total time cost of query selection is thus in $O(k|V|m)$, with $k' \leq 3 * k$ in our setting.

VI. Query Annotation

While active learning often assumes oracles to have full capability to provide correct labels, it is not an easy task for error detection. Meanwhile, downstream analysis such as error diagnosis [1], descriptive [9] and statistics-based repairing [13], [32], [54] benefits from useful auxiliary information that are generated from error detection. Accordingly, the role of a query annotator $\mathcal{A}$ is to collect, profile and share useful data from $\mathcal{O}$ and base detectors Ψ to components of GALE, to (a) reduce manual effort of labeling (for oracle $\mathcal{O}$), and (b) reduce the cost of query selection (for query selector $\mathcal{S}$). The auxiliary data can also be re-used to facilitate follow-up data repairing, and to reduce unnecessary computation. We next describe the query annotation module $\mathcal{A}$ in GALE.

Auxiliary Information. GALE initializes a map structure $v.M$ for unlabeled nodes in $\mathcal{Q}$. Given a node $v \in \mathcal{Q}$, it collects and stores in $v.M$ the following information, *whenever* applicable:

- (**Type 1: "soft subgraphs"**) a subgraph that is induced by the neighbors influenced by or to v via random walks, and induced soft labels (Section V);
- (**Type 2: detected errors**) a list of erroneous values $v.A$ = 'a' that are captured by base detectors in Ψ, with confidence scores;
- (**Type 3: suggested corrections**) for each erroneous value $v.A$ = 'a', a set of suggested correct counterparts, which may be obtained from the value bindings enforced by data constraints [14], [18], [36]; and
- (**Type 4: error distribution**) a statistical error distribution detected by the base detectors Ψ alone at v.

GALE shares these global structures with the oracle $\mathcal{O}$ and the query selector $\mathcal{S}$ to facilitate label generation and query selection. We remark that the auxiliary data can be user-defined to satisfy the need of other typicality measures, query selection strategies, and oracles.

Annotation Generation. Algorithm QAnnotate (illustrated in Fig. 6) implements the annotator $\mathcal{A}$. (1) For each query node in $\mathcal{Q}$, it invokes procedure GetAnnotate to incrementally update the neighborhood information (Type 1: soft graphs) via Label Propagation [58]. (2) QAnnotate utilizes Ψ to generate error probability distribution of each training node and derive Types 2-4 data as follows. (a) For each base detector $f_i \in \Psi$ of a particular class C_i, it runs f_i over the nodes V to evaluate the number of erroneous nodes that are captured by f_i (denoted as $|\Psi_i|$), and computes the counterpart $|\Psi_{C_i}|$ that counts all the erroneous nodes captured by the same class of base detectors. This yields a normalized confidence score for each base detector f_i as $\frac{|\Psi_i|}{|\Psi_{C_i}|}$ (Type 2: detected errors). (b) The overall error distribution is then estimated as a weighted sum of the scores to represent the probability that v is "polluted" by particular error type (Type 4: error distribution). (3) When applicable, a list of correct values for a detected error is attached to $v.M$ by *e.g.,* "enforcing" the data constraints [5], suggesting majority of domain values [7], or applying string transformations at v (Type 3: suggested corrections).

VII. Implementation and Optimization

Memorization. GALE optimizes the iterative learning of the active adversarial framework (lines 7-12 in Fig. 3) with a *memorization* strategy. The general idea is to record previously computed model parameters and reduce unnecessary calculation, and avoid unnecessary update of the model parameters if the changes to the node embeddings are small. We make a case of the optimization for the query selector $\mathcal{S}$ (algorithm QSelect), a major source of redundant computation. Indeed, it requires (a) pairwise comparison of the large amount of unlabeled nodes, and (b) repeated computation of Personalized PageRank matrix P. On the other hand, we observe that the distance measure satisfies triangle inequality, and P remains static once computed. GALE thus maintains the following structures: (a) a matrix to store the node pair embedding distance $d(u, v)$ for unlabeled node pair $(v,\ v')$ whenever $u, v \in \{U \cup C\}$; (b) a vector of flags to indicate whether the learned embedding of unselected nodes changes or not in two consecutive iterations, (c) a dictionary that records the typicality of $\mathcal{Q}$, where the key is the size of the $\mathcal{Q}$, and (d) pre-computed matrix P. GALE simply retrieves an "approximate" distance $d'(u, v)$ that is stored before if the embeddings of node v and v' are not significantly changed even $\mathcal{D}^i$ is updated. If the embedding of u and v are element-wise equal within a tolerance, then $d(u, v)$ is not updated.

The optimization is quite effective. As shown in Section VIII, GALE incurs a feasible additional learning cost to improve error detection with significant gain on accuracy, and the optimization reduces the learning cost by 64%.

Dataset	$\|V\|$	$\|E\|$	# node types	# edge types	avg. # attr
DBP	2.2M	7.4M	73	584	4
OAG	0.6M	1.7M	5	6	2
Yelp	1.5M	1.6M	42	20	5

TABLE II: Overview of Real-world Graphs

Dataset	$\|V_T\|$	$\|E_T\|$	avg.# attrs	$\|V_{\mathcal{T}}\|$	$\|V^e\|$
Species(SP:DBP)	17.7K	20K	4	1062	134
Data Mining(DM:OAG)	11.2K	12.9K	3	670	158
Machine Learning(ML:OAG)	3.4K	3.3K	3	203	54
UserGroup1(UG1:Yelp)	3.4K	2.6K	3	202	57
UserGroup2(UG2:Yelp)	3.3K	2.5K	3	196	45

TABLE III: Examples of Processed Graphs

System Implementation. We have developed a prototype system GALE, built on our pilot system GEDet [22].

Feature Engineering. GALE adopts word embedding [20] that maps attribute-level tokens (e.g., words) to vectors of real numbers. We firstly feed the node attribute embedding to a graph autoencoder GAE, which exploits graph structure information to learn node representations. GALE concatenates the attribute-level representation and node-level representation as the input of the SGAN. Furthermore, Principal Component Analysis (PCA) [33] is used to reduce training cost.

Built-in Library. By default, GALE has three classes of built-in base detectors: (1) " constraints-based detectors", which detects errors as violations of data constraints, *e.g.,* graph functional dependencies [18], (2) "outlier detectors", which encodes the algorithm in *e.g.,* [7] to detect errors. (3) "string error detectors", including the detection of string noises such as spelling errors. GALE keeps track of a set of "invertable" base detectors f, such that their consequence can be used to generate Type 3: suggested corrections (See Section VI).

Deployment. GALE builders and servers are deployed in Google Colab environment with Tensorflow libraries and NVIDIA TESLA P100 with 16GB GPU memory. The base detector library, graph augmentation module, and an initial GUI has been well supported by the established implementation based on HTML5, Bootstrap and Vis.js in a pilot system GEDet [23]. Resources of GALE are made available [2].

VIII. Experiment

We experimentally verify the effectiveness and the training cost of GALE for error detection in real-world graphs.

A. Experiment settings

Datasets. We used the following real-world graph data. (1) DBP[3] is a knowledge graph extracted from Wikipedia, includes entities such as "plant" or "animal" and relationships among entities such as "studiedBy"; (2) OAG, a fraction of the open academic graph[4], contains nodes such as *e.g.,* articles, authors, organizations, and edges such as "cite" or "affiliatedTo"; (3) Yelp[5] is a graph that contains entities such as users and services (*e.g.,* plumbers, restaurants), and edges

[2]https://github.com/CWRU-DB-Group/GALE/tree/main/code
[3]https://wiki.dbpedia.org/develop/datasets
[4]https://www.aminer.org/open-academic-graph
[5]https://www.yelp.com/dataset

such as “friendWith” and “reviews”. We also show the sizes of graphs that are induced by specific types (*e.g.,* ‘Machine Learning’ from OAG denotes an induced subgraph with nodes that belong to the topic of ‘Machine Learning’). Tables II and III provide the details of the original and several examples of processed graphs, respectively.

Error Generation. We injected erroneous values into the real graphs with a configurable error generator. The generator enhances BART [2], a tool that pollutes attributes to evaluate error detection. Three error types were introduced with the help of stand-alone base detectors in a built-in library Ψ.

(1) *Constraint violations* are violations of a set of data constraints Σ [18]. We discovered data constraints Σ by invoking the algorithm in [17]. Matching the condition in the data constraints, the generator perturbed values to violate the consequent. We ensure meaningful constraints Σ with high minimum support (the number of node matches) and confidence (the nodes that satisfy the consequent of a data constraint). Setting minimum support as 1000, 10, and 20 and confidence as 0.9, 0.8, and 0.85 for DBP, OAG, and Yelp, we discover 89, 21, and 112 data constraints, respectively.

(2) *Outliers* are injected disturbed values that are significantly different from the distribution of peer attribute values. Some can be captured by outlier detectors in Ψ.

(3) *String noises* are generated from multiple scenarios including misspelling, missing values (‘null’), and random string disturbance. We ensured that injecting these errors alone are not leading to violations of Σ or as detectable outliers.

We set the node error rate, which denotes the probability that a node in the graph is erroneous; attribute error rate, which denotes the probability that an attribute is perturbed as an erroneous attribute; and detectable rate, which denotes the chance an error is captured by a base detector. The node error rate, attribute error rate and detectable rate are set as 0.01, 0.33, and 0.5 by default, respectively. For each graph, we randomly partition the nodes to obtain 6 folds for training examples, 1 fold for the validation set, and 3 folds for testing nodes (see Table III). The constraint-based detectors and outlier detectors are integrated into the annotation module for attribute value suggestion if consulted, to test how well GALE exploits external knowledge.

Oracle. For controlled tests, we simulated the oracle with the set of base detectors Ψ. An ‘error’ label is assigned if a base detector identified erroneous attribute values of the query. For our case study, we have invited students who are experienced in verifying knowledge graphs and Wikipedia editing history to manually verify the errors and correct nodes.

Algorithms. We implemented GALE, along with 3 other variants that interacts with the adversarial learning with different query selection strategies, one variant that removes the memoization optimization, and compared their performances with 5 additional baselines, all categorized as follows.

(1) Four variants of GALE, that replace query selector $\mathcal{S}$ with different strategies, including (a) GALE (-Ran.), a uniformly sampling of the unlabeled nodes; (b) GALE (-Ent.), entropy-based uncertainty sampling using the softmax outputs; (c) GALE (-Kme.), sampling the unlabeled nearest nodes to K-means clustering centroids [27], and (d) *U*_GALE, a variant without the memorization strategy for query selection.

(2) Alad [37], a state-of-the-art anomaly node detection framework that measures normality of the nodes by considering both the topological structures of the graph and attribute distribution estimation within local context of nodes.

(3) VioDet, a constraint-based error detection that detects errors as the union of the violations of a set of graph data constraints Σ mined from the original datasets.

(4) Raha [39] is a state-of-the-art method to detect errors in relational data. It configures a library of built-in detectors *e.g.,* outlier detection, to generate error detection strategies.

(5) Two Graph Neural Network-based error detectors: (a) GCN [30] applies a graph convolutional architecture to encode local graph structure and features of nodes as a semi-supervised node classifier; (b) GEDet [22], a state-of-the-art few-shot learning based error detection framework. It uses graph augmentation to enhance examples with synthetic ones, and adopts adversarial learning to train a classifier.

Configuration. We use consistent settings for fair comparison. (1) As Alad ranks nodes and is evaluated by AUC-PR curve [37], we applied the default setting to learn Alad, selected the thresholds that enable its best performance in terms of AUC-PR curve, and derived anomalies as errors. (2) We used the same set of data constraints Σ for GALE variants, GEDet, and VioDet. We used the same settings for shared hyperparameters in variants of GALE. GALE and its variants are consistently trained with the same number of iterations T and local budget k, within ranges $[7, 17]$ and $[5, 100]$, respectively. (3) As Raha is designed for relational data, we applied it to node tables with one table per node type.

Evaluation metrics. We evaluate the performance in effectiveness and efficiency. For effectiveness, we report precision, recall, and F_1-score. Denote $\mathsf{Err_d}$ as the set of erroneous nodes detected from the graph, and Err as the set of nodes that are erroneous in the graph. The precision, recall and F_1-score are defined as $\mathsf{P} = \frac{|\mathsf{Err_d} \cap \mathsf{Err}|}{|\mathsf{Err_d}|}$, $\mathsf{R} = \frac{|\mathsf{Err_d} \cap \mathsf{Err}|}{|\mathsf{Err}|}$, and $\mathsf{F} = \frac{2\mathsf{PR}}{\mathsf{P}+\mathsf{R}}$, respectively. For efficiency, we report the training time of GALE (-Ent.), GALE (-Ran.), GALE (-Kme.), and GALE.

All Experiments were executed on a Unix environment with Intel 2.6GHz CPUs, and 16GB memory. All the algorithms were implemented in Python on Tensorflow. Each experiment was run 5 times and the median results were reported.

B. Experiment results

We first evaluate the effectiveness of GALE and baseline algorithms, and the impact of several factors. We then evaluate the training cost of GALE. In addition, we conduct case studies to evaluate the usability of query annotation.

Exp-1: Accuracy of GALE. We report the accuracy of the methods over all the datasets in Table IV. GALE variants are

Data	Met.	VioDet	Alad	Raha	GCN	GEDet	GALE (-Ent.)	GALE (-Ran.)	GALE (-Kme.)	GALE
SP	P	0.85	0.26	0.40	0.57	**0.9085**	0.8237	0.8831	0.8530	0.8173
	R	0.24	**0.80**	0.60	0.35	0.6009	0.6801	0.6311	0.6859	0.7219
	F_1	0.38	0.39	0.48	0.43	0.7233	0.7451	0.7361	0.7604	**0.7666**
DM	P	0.26	0.23	0.50	0.35	0.9812	**0.9814**	0.9813	**0.9814**	**0.9814**
	R	0.30	**0.77**	0.43	0.74	0.4541	0.4578	0.4566	0.4578	0.4578
	F_1	0.28	0.35	0.47	0.47	0.6209	**0.6244**	0.6232	**0.6244**	**0.6244**
ML	P	0.24	0.23	0.62	0.63	**0.9725**	**0.9725**	0.9561	0.9643	0.9487
	R	0.27	0.40	0.45	0.43	0.4569	0.4569	0.4698	0.4655	**0.4784**
	F_1	0.25	0.30	0.52	0.51	0.6217	0.6217	0.6301	0.6279	**0.6361**
UG1	P	0.33	0.27	0.63	0.51	**0.7764**	0.7640	0.7640	0.7755	0.7586
	R	0.55	0.55	0.60	0.52	0.6389	0.6632	0.6632	0.6597	**0.6875**
	F_1	0.41	0.36	0.62	0.52	0.7010	0.7100	0.7100	0.7129	**0.7213**
UG2	P	0.31	0.27	0.59	0.66	**0.9576**	0.8881	0.8627	0.8836	0.8599
	R	0.54	**0.73**	0.56	0.33	0.4502	0.5060	0.5259	0.5139	0.5378
	F_1	0.39	0.39	0.57	0.44	0.6125	0.6447	0.6535	0.6499	**0.6618**

TABLE IV: Performance of Error Detection. **Bold**: best result; Underlined: second best.

initialized by using 10% of the training nodes V_T (summarized in Table III). The total budget size for Species(DBP), Data Mining (OAG), Machine Learning (OAG), UserGroup1 (Yelp), and UserGroup2 (Yelp) are 800, 490, 25, 50, and 50, respectively. We report our findings below.

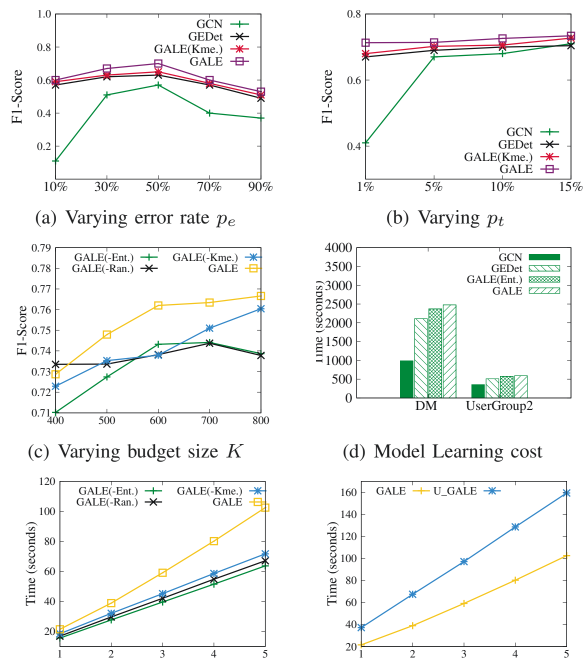
(1) We first inspect the 5 competing methods (all excluding GALE variants). The low recall of VioDet suggests the errors are quite diversified. GCN (graph neural network learning) and Raha (assembles multiple learning methods) are able to improve F_1-scores, but may come with a cost of significant precision drop (*e.g.,*GCN on 'UG2' dataset). Among the competing methods, GEDet achieves the best F_1-scores since its few-shot learning module and graph augmentation module ensure the coverage of heterogeneous errors.

(2) Active learning can effectively improve error detection. Despite the diversified errors, GALE variants achieve either the top or the second best results in precision, recall or F_1-score in almost all the cases. This is not recognized in individual competing methods. On average, GALE has improved the F_1-score of VioDet, Alad, Raha, GCN and GEDet, with a margin as 0.33, 0.31, 0.14, 0.20, and 0.02, respectively.

(3) In general, GALE variants with active adversarial framework achieve desirable, and robust performance in accuracy despite of the presence of multiple types of errors and datasets. For example, in all cases, GALE and the three variants achieve at least 0.74 in precision, 0.45 in recall and 0.62 in F_1. Other methods demonstrate higher variance given different datasets. For example, VioDet has a high precision of 0.85 on SP, and only achieves 0.24 precision on ML; similarly for GCN.

(4) For *all* the datasets, GALE achieves the best performance among all the GALE variants. This verifies the effectiveness of the typicality-based query selection. We found that typical nodes from diversified clusters should be recommended to be queried in the low budget regime, instead of choosing those with the high uncertainty, or random selection (GALE (-Ran.)).

Exp-2: Impact of factors. We next investigate the impact of the following factors: (1) the data imbalance $p_e = \frac{|V^e|}{|V_T|}$, and (2) the training data ratio $p_t = \frac{|V_T|}{|V|}$, and (3) a cumulative query size K (the total # queries $T \cdot k$).



(a) Varying error rate p_e (b) Varying p_t

(c) Varying budget size K (d) Model Learning cost

(e) AL cost: varying iteration T (f) Optimization: varying T

Fig. 7: Impact of factors to model performance

Impact of Data Imbalance. Fixing $p_t = 10\%$ and K = 80, we vary the imbalance p_e from 0.1 to 0.9 over Machine Learning (OAG). Fig. 7(a) tells us that while all methods achieve better performance over more balanced data, GALE (-Ent.), GALE (-Ran.), GALE (-Kme.), GALE, GEDet, are more stable than GCN, due to the graph augmentation can counteract imbalanced examples. Table IV consistently justifies this as the fraction of $|V^e|$ varies over different datasets (summarized in Table III): compared with GCN, the methods GALE (-Ent.), GALE (-Ran.), GALE (-Kme.), and GALE are less sensitive.

Impact of Error Distribution. We also evaluated the accuracy of GALE under different error distribution: violations-heavy, where 50% of injected errors are constraint violations, and the rest two are of equal chances; similarly for outliers-heavy, and string-noise-heavy. GALE is able to pertain robust performance as $82.59 \pm 1.15\%$ in F_1-score on 'UG1' datasets with different error distributions since its adversarial active learning consistently learns the real error distribution.

Varying example size. Fixing $K = 80$ and $p_e = 50\%$, we vary p_t over dataset UserGroup1(Yelp) from 15% to 1% and report the result in Fig. 7(b) (the result of VioDet and Alad are insensitive and constantly 0.41 and 0.36, respectively; not shown). While the accuracy decreases for all models as less training data is available, GALE (-Ent.), GALE (-Ran.), GALE (-Kme.), GEDet is the least sensitive (remains a recall above 0.6; not shown). Indeed, active learning variants in GALE effectively counteract the impact of lack of labels, with the graph augmentation model that improves recall, and the adversarial module that improves the accuracy.

Varying cumulative budget K. Keeping other parameters by default, we varied cumulative budget K from 400 to 700 (with a fixed k=100) and report the result in Fig. 7(c). While the F_1-scores increase as more unlabeled nodes are queried and get labeled by oracles for all active learning sampling strategies in general, GALE and GALE (-Kme.) achieves the better F_1-scores compared with the other two methods, which affirms our claim that the typical and diversified nodes should be preferred to get selected over atypical (uncertain) nodes when the query budget is low. Hence, K-means clustering based sampling strategies (including GALE and GALE (-Kme.)) help active learning select typical graph nodes that are suited for low budgets. Furthermore, GALE outperforms GALE (-Kme.), which indicates the diversity that makes selected nodes be far apart benefits active learning in low-budget regime.

Exp-3: Learning cost. We compare the learning costs of GALE variants and other baselines. GALE and variants do not incur much additional cost especially in a desirable low-budget regime when local budget k is small. We compare the learning cost of different query selection strategies in both low-budget regime and high-budget regime among GALE variants.

Model Learning cost. We set the number of epochs as 220 for all the methods (200 epochs for the GAN in GALE variants to reach a Nash equilibrium and 20 epochs for the generative adversarial active learning module to query unlabeled nodes in the graph) and apply an "early-stop" strategy based on validation performance and make GEDet and GALE variants terminate early if no improvement is observed within consecutive 20 epochs. As shown in Fig. 7(d), (1) it is quite feasible to learn GALE. For example, it takes 520 seconds to learn GALE to achieve a recall at 0.48 over UG2; (2) GALE although is with the most sophisticated optimization goal, it still has a comparable running time with other GALE variants and improves the accuracy of error detection at a cost of small overhead. For example, it introduces on average 33%, 45%, 15% and 62% additional cost compared with GALE (-Kme.), GALE (-Ent.) (the running time of GALE (-Ran.) is close to GALE (-Ent.), not shown), GEDet and GCN, respectively.

Active Learning cost. We fix the queried nodes as 10 in each epoch on the Data Mining (OAG) of all GALE variants. After the labels of these queried nodes are provided by the oracle, we keep updating the model parameters of GALE variants. In the low-budget regime of Fig. 7(e), GALE introduces on average 53.8%, 42.9%, and 33.3% additional cost compared with GALE (-Ent.), GALE (-Ran.) and GALE (-Kme.). In general, the additional overhead of GALE is not significantly increased due to our memorization strategies to skip unnecessary computation in the iterative learning and model updating.

Optimization of GALE. Using the same low-budget regime, we compare the cost of GALE and U_GALE. As Fig. 7(f) shows, the optimization strategy significantly reduces the learning cost. For example, it reduces the cost of U_GALE by 40% on the Data Mining (OAG) when $k = 10$.

Exp-4: Case study: Usability of Query Annotation. We illustrate a "hard" test case when detecting errors in Species(DBP). The test node v has attributes v.name = "cavanillesia" and v.order with a wrong value "Lepidoptera", which should be "Marvales". No graph constraint or outlier detector in Ψ detects this error. The student who is requested to label the nodes also has little knowledge of species. (1) In an iterative process, GALE selected a typical node v' that is semantically similar to v, where v and v' were from the same cluster after embedding-based clustering. Finer analysis showed that v and v' share one common attribute kingdom ="plantae", which indicates their semantic similarity. (2) The annotator $\mathcal{A}$ successfully associated node v' with the following auxiliary data: (a) a detected error "Melvaceae" that violates a graph constraint in Ψ, (b) a suggested correct value "Malvaceae" by enforcing the graph constraint, (c) the distribution of errors at v', and (d) the distribution of influence that comes from the labeled nodes in Personalized PageRank matrix P from $\mathcal{A}$, where the most influential labeled node also came with an "error" label. The student was able to correctly label node v' as "error". (3) With v' correctly labeled, the node classifier is further improved, which successfully detected the test node v as an erroneous node in the next iteration.

IX. Conclusion

We have introduced GALE, a graph error detection framework empowered by an active adversarial learning framework. GALE exploits active learning to best exploit the new knowledge from oracles by issuing a bounded number of queries. Moreover, we introduce new quality measures for selecting queries in graph data in terms of diversified typicality, and introduced approximation algorithms for query selection, query annotation schemes to facilitate oracle and query selection, and optimization strategies. Our experimental study verifies that the active learning and adversarial error detection of GALE achieve significant gain on accuracy compared with state-of-the-art baselines. A future topic is to enhance GALE for large-scale and more types of errors with distributed learning.

Acknowledgment

This work is supported by NSF under CNS-1932574, OIA-1937143, ECCS-1933279, CNS-2028748, OAC-2104007 and DoE under DE-EE0009353 and DE-NA0004104 .

REFERENCES

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *VLDB*, 2016.

[2] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with bart: error generation for evaluating data-cleaning algorithms. *VLDB*, 2015.

[3] B. Biggio, G. Fumera, and F. Roli. Multiple classifier systems for adversarial classification tasks. In *International Workshop on Multiple Classifier Systems*, pages 132–141. Springer, 2009.

[4] B. Birnbaum and K. J. Goldman. An improved analysis for a greedy remote-clique algorithm using factor-revealing lps. *Algorithmica*, 55(1):42–59, 2009.

[5] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.

[6] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166, 2012.

[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, 2000.

[8] G. Castagna, D. Colazzo, and A. Frisch. Error mining for regular expression patterns. In *Italian conference on Theoretical Computer Science*, pages 160–172. Springer, 2005.

[9] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *SIGMOD*, 2014.

[10] D. Chen, Y. Lin, G. Zhao, X. Ren, P. Li, J. Zhou, and X. Sun. Topology-imbalance learning for semi-supervised node classification. *Advances in Neural Information Processing Systems*, 34:29885–29897, 2021.

[11] F. Chen, Y.-C. Wang, B. Wang, and C.-C. J. Kuo. Graph representation learning: a survey. *APSIPA Transactions on Signal and Information Processing*, 9, 2020.

[12] K. Cheng, X. Li, Y. E. Xu, X. L. Dong, and Y. Sun. Pge: Robust product graph embedding learning for error detection. *arXiv preprint arXiv:2202.09747*, 2022.

[13] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *SIGMOD*, 2016.

[14] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, 2013.

[15] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108, 2004.

[16] K. Ding, J. Li, and H. Liu. Interactive anomaly detection on attributed networks. In *Proceedings of the twelfth ACM international conference on web search and data mining*, 2019.

[17] W. Fan, C. Hu, X. Liu, and P. Lu. Discovering graph functional dependencies. In *SIGMOD*, 2018.

[18] W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *SIGMOD*, 2016.

[19] Y. Fu, X. Zhu, and B. Li. A survey on instance selection for active learning. *Knowledge and information systems*, 35(2):249–283, 2013.

[20] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*, 2018.

[21] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2005.

[22] S. Guan, P. Lin, H. Ma, and Y. Wu. Gedet: Adversarially learned few-shot detection of erroneous nodes in graphs. In *IEEE Big Data*, 2020.

[23] S. Guan, H. Ma, S. Choudhury, and Y. Wu. Gedet: detecting erroneous nodes with a few examples. *Proceedings of the VLDB Endowment*, 14(12):2875–2878, 2021.

[24] G. Hacohen, A. Dekel, and D. Weinshall. Active learning on a budget: Opposite strategies suit high and low budgets. *arXiv preprint arXiv:2202.02794*, 2022.

[25] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *SIGMOD*, 2019.

[26] I. F. Ilyas, X. Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 2015.

[27] X. Jin and J. Han. K-means clustering. In *Encyclopedia of Machine Learning*, 2010.

[28] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1215–1230, 2015.

[29] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[31] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[32] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12):948–959, 2016.

[33] R. Lebret and R. Collobert. Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*, 2013.

[34] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *SIGIR'94*, 1994.

[35] Y. Li, J. Yin, and L. Chen. Seal: Semisupervised adversarial active learning on attributed graphs. *IEEE Transactions on Neural Networks and Learning Systems*, 32(7):3136–3147, 2020.

[36] P. Lin, Q. Song, Y. Wu, and J. Pi. Repairing entities using star constraints in multirelational graphs. In *ICDE*, 2020.

[37] N. Liu, X. Huang, and X. Hu. Accelerated local anomaly detection via resolving attributed networks. In *IJCAI*, 2017.

[38] K. Madhawa and T. Murata. Active learning for node classification: an evaluation. *Entropy*, 22(10):1164, 2020.

[39] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *SIGMOD*, 2019.

[40] C. Mayfield, J. Neville, and S. Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In *SIGMOD*, 2010.

[41] B. Miller, A. Kantchelian, S. Afroz, R. Bachwani, E. Dauber, L. Huang, M. C. Tschantz, A. D. Joseph, and J. D. Tygar. Adversarial active learning. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 3–14, 2014.

[42] F. Neutatz, M. Mahdavi, and Z. Abedjan. Ed2: A case for active learning in error detection. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2249–2252, 2019.

[43] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[44] H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 2017.

[45] Z. Peng, M. Luo, J. Li, H. Liu, and Q. Zheng. Anomalous: A joint modeling approach for anomaly detection on attributed networks. In *IJCAI*, 2018.

[46] K. Pourahmadi, P. Nooralinejad, and H. Pirsiavash. A simple baseline for low-budget active learning. *arXiv preprint arXiv:2110.12033*, 2021.

[47] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NeurIPS*, 2016.

[48] B. Settles. Active learning literature survey. 2009.

[49] D. J. Wang, X. Shi, D. A. McFarland, and J. Leskovec. Measurement error in network data: A re-classification. *Social Networks*, 2012.

[50] H. Wang and J. Leskovec. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755*, 2020.

[51] Y. Wang, Q. Yao, J. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. In *arXiv: 1904.05046*. 2019.

[52] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[53] M. Xu. Understanding graph embedding methods and their applications. *SIAM Review*, 63(4):825–853, 2021.

[54] J. N. Yan, O. Schulte, M. Zhang, J. Wang, and R. Cheng. Scoded: Statistical constraint oriented data error detection. In *SIGMOD*, 2020.

[55] W. Zhang, Y. Wang, Z. You, M. Cao, P. Huang, J. Shan, Z. Yang, and B. Cui. Rim: Reliable influence-based active learning on graphs. *Advances in Neural Information Processing Systems*, 34:27978–27990, 2021.

[56] C. Zhao, Y. Xin, X. Li, Y. Yang, and Y. Chen. A heterogeneous ensemble learning framework for spam detection in social networks with imbalanced data. *Applied Sciences*, 10(3):936, 2020.

[57] Y. Zhong, W. Chen, Z. Wang, Y. Chen, K. Wang, Y. Li, X. Yin, X. Shi, J. Yang, and K. Li. Helad: A novel network anomaly detection model based on heterogeneous ensemble learning. *Computer Networks*, 169:107049, 2020.
[58] X. Zhu. *Semi-supervised learning with graphs*. Carnegie Mellon University, 2005.