Graph Learning Indexer: A Contributor-Friendly Platform for Better Curation of Graph Learning Benchmarks

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

Establishing common benchmarks has been a critical driving force behind the success of modern machine learning techniques. As machine learning is being applied in broader domains and tasks, there is a need to establish more and diverse benchmarks to better reflect the reality of the application scenarios. For graph learning, an emerging field of machine learning, the need of establishing better benchmarks is particularly urgent. Towards this goal, we introduce Graph Learning Indexer (GLI)[1], a benchmark curation platform for graph learning. In comparison to existing graph learning benchmark libraries, GLI highlights two novel design objectives. First, GLI is designed to incentivize *dataset contributors*. In particular, we incorporate various measures to minimize the effort of contributing and maintaining a dataset, increase the usability of the contributed dataset, as well as encourage better credits to different contributors of the dataset. Second, GLI is designed to curate a knowledge base, instead of a collection, of benchmark datasets. For this purpose, we come up with multiple sources of meta information of the benchmark datasets in order to better characterize the datasets.

1 Introduction

The practice of establishing common benchmarks in machine learning dates back to research programs of speech recognition in 1980s [1, 2], and has become a dominant paradigm of machine learning research. In the past, the community has been focusing on a handful of benchmarks in each major domain of machine learning applications[2], usually developed by few institutes or research groups. However, as machine learning is becoming a general-purpose technology[3], there are *new demands* from modern machine learning research that are not entirely met by the current common practice of benchmarking:

1. *Broad Application.* Machine learning is being applied to increasingly broad domains, where the emerging field of graph learning is an example with a variety of machine learning tasks in this domain. Representative new benchmarks are needed for such new domains and tasks. Furthermore, the development of good benchmarks often require inter-disciplinary knowledge and collaborations.
2. *Trustworthiness.* The collection of each individual benchmark datasets could be biased. Driving the development of machine learning technologies by a few fixed benchmark datasets may suffer from the biases in these datasets. It is therefore desirable to leverage a set of *diverse* benchmark datasets to expose the potential trustworthy concerns of the machine learning technologies.
3. *General Technology.* Towards more general-purpose artificial intelligence, there is a strong emerging interest in developing machine learning models that can perform well on a wide range

[1]The anonymized codebase for this platform is available here: https://anonymous.4open.science/r/gli-updated-3D70.

[2]For example, ImageNet [3] in Computer Vision, SuperGLUE [4] in Natural Language Processing, and Open Graph Benchmark [5] in Graph Learning.

[3]https://en.wikipedia.org/wiki/General-purpose_technology

of downstream tasks [6]. In conjunction with this interest, there have been efforts constructing benchmarks with *many tasks*, such as SuperGLUE [4], GEM [7], and BIG-Bench [8], where BIG-Bench consists of 204 tasks by more than 400 authors across 132 institutes.

These new demands, especially for emerging fields such as graph learning, require the development of a large quantity of diverse benchmark datasets, in order to better reflect the reality of machine learning applications. This requirement poses challenges in both creation and curation of the benchmarks.

In this paper, we introduce Graph Learning Indexer (GLI), a graph learning benchmark curation platform, to mitigate the aforementioned challenges. In particular, GLI highlights two novel design objectives that respectively mitigate the challenges in benchmark creation and benchmark curation.

First, GLI aims to leverage contributions from the broad graph learning community to establish a wide range of benchmarks. As a result, GLI is designed to be *contributor-centric*, where we treat benchmark contributors as our core users when designing the platform. Specifically, we incorporate various designs, such as file-based data API, automated test, and template files, to minimize the effort of contribution and maintenance by the benchmark contributors. We have also considered measures to incentivize research effort in benchmark contributions in general. For example, in order to encourage better credits to the benchmark contributors, GLI includes the chain of prior versions of each benchmark dataset in the bibliographic section of the dataset README file.

Second, with the increasing quantity and diversity of benchmark datasets, GLI aims to build a knowledge base of the datasets, instead of a simple collection of datasets. GLI includes a *Benchmark Indexing System*[4] with various sources of meta information about the benchmark datasets collected by GLI. Such meta information can be later used for better curation and retrieval of the benchmarks.

The rest of this paper is organized as following. We introduce the contributor-centric design and the benchmark indexing system respectively in Section 2 and Section 3. Section 4 reviews related prior work on benchmark collections and graph learning libraries. We also include a sketch of future plan for GLI in Section 5. Finally, in Section 6, we conclude this paper with some open questions on benchmark design.

2 Contributor-Centric Design

A central goal of GLI is to incentivize the graph learning community to put more effort on contributing high-quality benchmark datasets. To achieve this goal, we treat dataset contributors as the core users of GLI and come up with three contributor-centric design objectives. First, GLI aims to provide smooth user experience for contributors by minimizing the effort on submission and maintenance of the datasets. Second, GLI aims to increase the impact of the hosted datasets by improving their usability. Third, GLI aims to encourage better credits to the dataset contributors through tangible measures.

2.1 User Experience and Quality Assurance

A key challenge in the design of GLI is to minimize the effort by the dataset contributors while assuring a high quality of the contributed datasets. Our solution to this challenge is to first design a standard data management API that is both stable and extensible for graph learning datasets; and then design a GitHub-based contribution workflow with concise instructions and rich feedback for dataset contributors to convert the benchmark datasets into the standard API.

2.1.1 Data Management API

The GLI Data Management API (Figure 1) has two key design features: the API is file-based; there is an explicit separation of data and task.

File-based storage API. The data API for almost all existing graph learning libraries (such as DGL [9] and PyG [10]) are code-based, which means that each dataset is associated with an ad hoc class that is dedicated to represent this dataset. For example, DGL [9] defines a `CoraGraphDataset` class for the node classification task on the Cora dataset [11, 12]. This code-based API couples the datasets with the codebase and increases the difficulty of maintenance. In particular, changes to the

[4]Thus "Indexer" in the name of GLI.

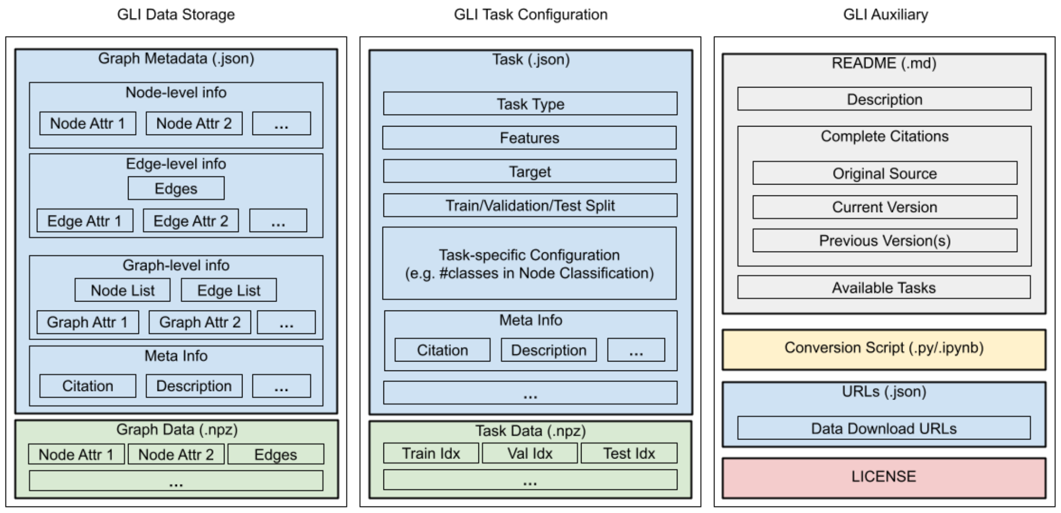


Figure 1: The file-based *GLI Data Management API* with explicit separation of data and task. The *GLI Data Storage* part contains all necessary information to construct the graph data, including three levels: node, edge, and graph information. Each level may have multiple features or labels as its attributes. The *GLI Task Configuration* part contains the necessary information to perform a predefined task. Both parts further compress big chunks of data (such as the attributes or edge list) into NumPy standard binary format, with indexes to these data stored in JSON files. The NumPy data files are hosted in an external storage system, while all other files are hosted in the GitHub repo of GLI. In addition, the *GLI Auxiliary* part contains a README document, a conversion script that converts the raw data into GLI file format, a LICENSE file, and a `urls.json` providing the URLs to the NumPy data in the external storage system.

graph learning library codebase may break the ad hoc dataset classes so additional maintenance effort is required for each dataset.

To avoid such unnecessary maintenance burden for dataset contributors, GLI adopts a *file-based* data storage API that is more stable compared to code-based APIs. While there has been file-based graph storage API, such as GraphML [13], they are not dedicated to graph learning datasets and lacks essential features such as storing the data splits. We therefore designed a novel file-based storage API for graph learning datasets.

Explicit separation of data and task. We recognize that there is a clear distinction between the information of the content in a dataset, i.e., the *data*, and the information about how to use the data to train and evaluate the models, i.e., the *task*. For example, in graph learning benchmarks, there could often be multiple tasks (e.g., node classification and link prediction) defined on the same dataset, or there could be multiple settings for the same task (e.g., random split or fixed split). From the persepctive of dataset contribution and curation, it is cumbersome to make a new version of dataset for each new task on top of the same data. Therefore, we propose to store the *data information* and the *task information* separately in our API. And we design a task-specific API for each type of tasks.

This explicit separation of data and task turns out to offer a number of benefits. First, it makes the API more extensible, as the introduction of a new type of task will not affect the API for the data. Second, this separation makes automated tests more modularized (see Section 2.1.2). Third, it allows the implementation of general data loading schemes (see Section 2.2). Finally, it leads to a bottom-up approach to grow the taxonomy of graph learning tasks (see Section 3.1).

Overview of the API. Figure 1 shows the architecture of the file-based API with explicit separation of data and task[5].

[5]A detailed document for the API is available at https://anonymous.4open.science/r/gli-updated-3D70/FORMAT.md.

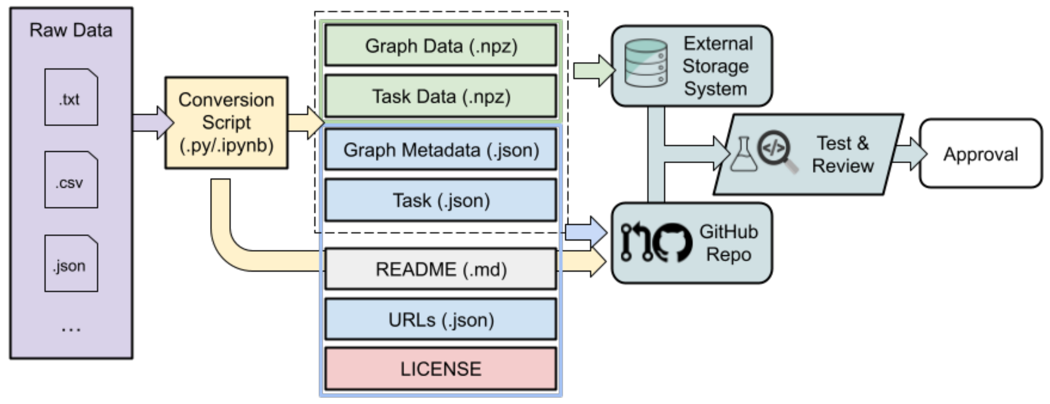


Figure 2: GLI Contribution Workflow. A contributor will first use the conversion script to convert the raw data into the GLI format. Then the contributor will fill in the templates of `README.md` and `urls.json`. The JSON files, auxiliary files (blue box), and conversion script will be uploaded to GitHub as a pull request and the NumPy data files (green box) will be uploaded to the external storage system. GLI will perform automated tests on the submitted datasets and the GLI development team will further review the pull request before approval.

The information of the graph data is divided into three levels: node, edge, and graph level. Each level can be assigned multiple attributes as features or labels and can be further divided into multiple sub-levels to represent heterogeneous graphs. The attributes support both dense and sparse tensors to allow efficient storage and fast loading. The GLI data format has a strong representative power to accommodate most graph-structured data.

For the task, we have predefined a number of graph learning task types, such as `NodeClassification`, `LinkPrediction`, `GraphClassification`, etc. The information in the task configuration can be divided into two kinds: general configuration and task-specific configuration. General configurations are commonly required by all tasks, including features that are allowed to use during prediction, train/validation/test split, etc. On the contrary, the contents of task-specific configurations depend on task types. For example, both `NodeClassification` and `GraphClassification` requires to specify the number of possible classes (`num_classes`), and `LinkPrediction` provides an optional configuration on negative samples during validation and test (`val_neg` and `test_neg`).

Overall, the file-based design improves the stability of the API while the separation of data and task makes the API more extensible, both in turn improves the user experience for dataset contributors.

2.1.2 Contribution Workflow

In companion with the data management API, we designed a GitHub-based contribution workflow (Figure 2) to ease the dataset contribution process.

Template files. To begin with, GLI provides a list of well-commented template files[6] for all the required files in our API. The contributor only needs to fill in all the blanks to convert a dataset into the GLI format.

Dataset submission and review. After finishing converting the dataset, the contributor will submit the required files as a pull request to the GitHub repository of GLI. The large NumPy binary files will be uploaded to an external storage system[7]. The GLI development team or other researchers can provide detailed and interactive feedback in the pull request.

Automated tests. In addition to the manual peer review, the pull request will also trigger automated tests with detailed error feedback to help the contributors debug their implementation. The tests

[6]See `https://anonymous.4open.science/r/gli-updated-3D70/templates/template/README.md`.

[7]Currently we use Google Drive and Dropbox accounts owned by the GLI development team as the storage system.

include the standard `pycodestyle`, `pydocstyle`, and `pylint` for syntax and style check. We have also implemented a wide range of in-depth tests with `pytest` to check the correctness of dataset format and to expose potential errors during runtime by sanity check with short model training. Contributors can also use several well-documented utility functions to test the correctness of their data format locally.

2.2 Dataset Usability

```
import gli

cora_node_dataset = gli.get_gli_dataset(dataset="cora",
                                        task="NodeClassification")
```

Demo 1: Example usage of the general data loading scheme. `cora_node_dataset` is an instance of `dgl.data.DGLDataset`, thus it can be fit into DGL dataloader seamlessly.

To increase the impact of the datasets hosted on GLI, we implemented a general data loading scheme that can seamlessly integrated into major graph learning libraries for downstream experiments. At the time of writing this paper, we have implemented data loading for DGL [9]. We also strive to accommodate other major libraries in the future. Demo 1 demonstrates an example of the general data loading scheme. Once a contributed dataset (and the task defined on it) is merged into the GLI repository, the dataset can be retrieved by calling `gli.get_gli_dataset` with the dataset name and task type as arguments.

Under the hood, as shown in Demo 2, `gli.get_gli_dataset` calls `gli.get_gli_graph` and `gli.get_gli_task` to respectively load the GLI Data Storage and GLI Task Configuration shown in Figure 1. Thanks to the explicit separation of data and task, we only need to maintain a general graph loading function and a set of task loading functions with each function dedicated to a task type, which is much less effort than maintaining a separate dataset class for each task and dataset combination.

```
graph_cora = gli.get_gli_graph(dataset="cora")
task_node = gli.get_gli_task(dataset="cora",
                             task="NodeClassification")
cora_node_dataset = gli.combine_graph_and_task(graph_cora, task_node)
```

Demo 2: The innerworkings of `gli.get_gli_dataset`.

2.3 Credits to Contributors

An important aspect to incentivize the dataset contributors is to ensure that they get the proper credits. For this purpose, we have made a couple of designs to help the community cite properly. There is citation information in the README file of each dataset listing the BibTex of the work relevant to the dataset. Specifically, the citation information is split into dataset and tasks, as there could often be multiple tasks defined on top of a graph dataset, and the definition of tasks could come from work that is different from the one contributing the dataset. Moreover, the citation information for the dataset is further split into three parts:

- *Original Source*: The first work that created the dataset.
- *Current Version*: The work that is directly responsible for the dataset stored in GLI.
- *Previous Versions*: Any intermediate versions between Original Source and Current Version. There can be multiple citations in Previous Versions.

The paper popularizing a benchmark dataset is often not the paper originally contributing the dataset. And it is not uncommon that the former gets most of the citations while the latter gets few. This phenomenon is possibly due to two factors. First, tracking the chain of contributions to a dataset through literature search is a tedious work. Second, researchers tend to get information about a dataset from the methodology papers that cite the dataset rather than the original paper creating the dataset. So the mistakes in citation accumulate.

By providing succinct bibliographic information relevant to the dataset in the README file, we hope to help the community better recognize the contributions of all contributors, with a particular emphasis on crediting the original source.

3 Benchmark Indexing System

With the growing quantity and diversity of benchmark datasets, it is important for the benchmark curation platform to help users navigate through the large collection of datasets. For this purpose, GLI is designed to serve as an "indexer" that builds a database consisting of various meta information of the benchmark datasets. And we name the database as *Benchmark Indexing System*. To some extent, this is in a similar vein as the idea of *Datasheets for Datasets* [14]. Datasheets for Datasets focus more on the characteristics of each individual dataset while our design of the database also cares about the synergy among different datasets.

Ultimately, we hope to use this database to help users 1) retrieve the right benchmarks that match the context of the applications of their interest; 2) identify potential biases and trustworthy issues existed in the datasets; or 3) motivate the development of new methodology based on the characteristics of tasks and datasets.

At the current stage, however, we focus on coming up with different sources of meta information to be included in the database. The current implementation consists of three types of meta information, which are detailed in the following subsections.

3.1 Task Types

The task types come as meta information naturally from the implementation of data management API in GLI. Graph data are ubiquitous but also diverse and so are the graph learning tasks defined on top of graph data. Different graph learning tasks may have distinct nature and thus require very different methodology. Therefore the task type is an important source of meta information for each benchmark dataset.

In GLI, the definition of task types is driven by the contributed benchmarks. When a contributor is contributing a new benchmark, they will first check if their benchmark belongs to one of the existing task types in GLI. If none of the existing task types can accommodate the new benchmark, the contributor can initiate the definition of a new task type. The GLI development team and the contributors will implement the support for the new task type, including dataset class, documentation, and automated tests.

This bottom-up approach of developing task types not only makes GLI highly extensible to new benchmark datasets, but also gradually grows a taxonomy of graph learning tasks as more benchmarks are being collected. A list of currently supported task types is given in Appendix A.

3.2 Graph Data Properties

Another type of meta information included in GLI is various graph data properties, such as average degree or average clustering coefficient. In classical network science literature [15, 16], the graph data properties have been shown to be informative about the characteristics of the graph data. In a recent study, Palowitch et al. [17] empirically demonstrated that there are clear patterns in the graph neural network performance associated with certain graph data properties of the benchmark datasets.

GLI integrates a function that can calculate a list of graph data properties for each contributed dataset. The graph data properties integrated in this function can be categorized into 6 groups.

- *Basic*: Is Directed, Number of Nodes, Number of Edges, Edge Density, Average Degree, Edge Reciprocity, Degree Assortativity;
- *Distance*: Diameter, Pseudo Diameter, Average Shortest Path Length, Global Efficiency;
- *Connectivity*: Relative Size of Largest Connected Component (LCC), Relative Size of Largest Strongly Connected Component (LSCC), Average Node Connectivity;
- *Clustering*: Average Clustering Coefficient, Transitivity, Degeneracy;
- *Distribution*: Power Law Exponent, Pareto Exponent, Gini Coefficient of Degree, Gini Coefficient of Coreness;

- *Attribute*: Edge Homogeneity, Feature Homogeneity, Homophily Measure, Attribute Assortativity.

The formal definitions of these graph data properties can be found in Appendix C.

3.3 Model Performance

The third type of meta information included in GLI is the performance of various popular models on the datasets. It is common to use a model's performance on different experiment settings and datasets to understand the model characteristics. Recently, it is shown that one can also use the performance of different models to characterize the datasets and obtain meaningful clusters of the datasets [18].

In GLI, we provide a benchmark suite that can benchmark a few popular machine learning models on the contributed benchmarks. The benchmark suite implements a separate set of training and hyperparameter tuning functions for each task type. Thanks to the general data loading scheme (as introduced in Section 2.2), the benchmark code can be easily extended to new datasets with the same task type. We currently have supported `NodeClassification` and `GraphClassification` in the benchmark suite.

Below, we provide an example to showcase how the model performance could provide useful information to characterize the datasets. Using the benchmark suite in GLI, we provide the performance of several popular models on a set of node classification datasets in Table 1. This experiment is a rough replication of Lim et al. [19], with extension to more datasets enabled by GLI. The detailed experiment setup (and citations to models and datasets) can be found in Appendix D.

Readers who are familiar with the recent graph learning literature may find that, not surprisingly, the best and second best performing models on each dataset are a good indicator of how "homophilous" [20] the dataset is. The early graph neural network models, GCN, GAT, and GraphSAEG, have better performance on more homophilous datasets, such as cora, citeseer, and pubmed. LINKX performs better on most of the remaining non-homophilous datasets. A few datasets, texas, cornell, and wisconsin, lead to notoriously unstable model performance, as shown by the large standard deviations for most models. It also seems that the graph structure does not help much for the task, as the models (MLP, CatBoost, and LightGBM) that do not utilize the graph structure perform the best on these datasets.

In general, the GLI API makes it easier to implement the benchmark suite for a wide range of models and datasets in well-controlled experiment setups, which enables the use of model performance as a way to characterize the datasets.

Table 1: Benchmark experiment results for node classification datasets. Test accuracy is reported for most datasets, while test ROC AUC is reported for binary classification datasets (genius, twitch-gamers, penn94, pokec). Standard deviations are over 5 runs. The best result on each dataset is bolded, and the second best result is underlined.

	GCN	GAT	GraphSAGE	MoNet	MLP	CatBoost	LightGBM	LINKX	MixHop
cora	81.03±0.82	**83.0±0.62**	81.46±0.74	76.44±1.85	59.1±2.3	59.38±1.25	36.40±0.00	59.36±2.41	79.64±1.55
citeseer	72.28±0.56	69.9±1.54	**73.38±0.82**	64.4±0.62	54.62±6.26	59.18±0.58	39.34±0.77	42.5±7.88	69.64±1.2
pubmed	**79.44±0.43**	79.04±0.76	78.4±0.35	76.18±0.84	73.7±0.5	69.96±1.15	54.86±0.33	56.49±7.92	76.61±1.35
texas	61.08±3.07	67.02±1.21	66.48±1.48	55.13±7.04	78.92±2.25	77.84±1.21	**83.78±0.00**	76.57±4.87	77.84±1.7
cornell	52.97±4.09	48.64±1.9	47.02±3.08	51.89±2.25	68.64±7.78	69.19±2.42	**77.30±1.48**	65.46±5.85	66.48±5.43
wisconsin	56.46±3.5	54.89±1.96	52.54±1.63	36.86±3.22	78.82±4.24	81.18±2.24	**81.96±0.88**	78.62±1.94	76.9±5.61
actor	29.36±0.73	30.15±0.56	29.26±0.5	26.35±1.01	**37.11±0.54**	34.57±1.44	32.12±0.24	33.56±1.84	34.77±0.94
squirrel	32.4±1.18	29.14±1.55	31.64±1.93	27.14±2.34	34.87±0.47	34.37±0.37	33.89±0.69	**62.43±1.23**	33.37±1.45
chameleon	45.92±2.61	46.18±0.93	48.72±0.47	32.54±1.24	49.16±0.66	41.89±2.54	30.92±1.24	**67.08±1.69**	48.72±1.39
arxiv-year	49.6±0.16	34.91±0.56	43.39±0.74	40.19±0.48	36.49±0.19	35.76±0.60	36.17±0.29	**52.73±0.34**	40.63±0.12
snap-patents	**55.46±0.11**	36.34±0.6	43.33±0.27	43.48±0.73	31.32±0.04	30.96±0.55	31.48±0.06	53.43±0.32	43.27±0.03
penn94	88.79±0.6	66.29±12.21	85.0±0.53	73.92±3.71	83.92±0.32	73.21±2.20	73.62±0.05	**93.47±0.27**	91.62±0.11
pokec	71.17±10.76	53.03±0.4	63.02±5.68	53.65±2.17	64.69±4.92	62.55±0.38	62.77±0.03	**90.54±0.12**	86.84±0.2
genius	84.15±1.71	49.86±28.68	80.31±0.23	63.23±2.39	84.42±0.2	82.48±0.00	82.48±0.00	**90.88±0.1**	90.04±0.12
twitch-gamers	62.4±0.22	59.57±0.88	61.68±0.3	58.02±1.26	59.66±0.09	61.57±0.05	61.62±0.02	**66.21±0.3**	64.22±0.08

4 Related Work

In this section, we review prior work on graph learning benchmarks, graph learning libraries, and other relevant effort on machine learning benchmark infrastructures.

4.1 Graph Learning Benchmarks and Graph Learning Libraries

Recently, there have been many infrastructural efforts on developing benchmark collections for graph learning [5, 21–24]. Among which the most widely-used ones at present are perhaps Open Graph Benchmark [5] and Benchmarking Graph Neural Networks [22]. GLI differs from the prior work in two key aspects.

1. GLI is specifically optimized to better serve the dataset contributors. Most existing graph learning benchmarks are designed with the "dataset consumers", instead of contributors, as the core users. To our best knowledge, dedicated designs to optimize contribution workflow of graph learning datasets were essentially non-exist prior to this work. For example, the contribution workflow for Open Graph Benchmark is to pack the dataset in a fixed format and email it to the maintenance team[8]. In comparison, our GitHub-based contribution workflow is more interactive and potentially more scalable.
2. GLI maintains a bottom-up dynamic task taxonomy while most of the existing benchmark collections have a top-down static taxonomy of graph learning tasks. The static taxonomy of graph learning tasks may limit the type of dataset and tasks could be contributed to the benchmark collections.

There are also a few workshops and conference tracks dedicated to research on benchmarks and datasets, such as the Workshop on Graph Learning Benchmarks[9] and the NeurIPS Datasets and Benchmarks Track[10]. These venues are friendly to the publications of benchmark contributions and have successfully solicited a number of new graph learning benchmark datasets. The development of GLI shares the same motivation as these endeavors towards incentivizing more contributions on benchmarks. And GLI could be used as an infrastructural tool for these publication venues to better evaluate and curate the collected benchmarks.

4.2 Graph Learning Libraries

In addition, there are a few general-purpose graph learning libraries, such as PyG [10], DGL [9], and TF-GNN [25], that are relevant to this work. While the primary focus of these libraries is not benchmark datasets, they also provide graph data API at the dataloader level. We suggest that the file-based API design in GLI is more contributor-friendly because 1) it is easier to convert the data to files than to implement a dataset class; 2) the file-based API does not rely on any software dependency and is less likely to break; 3) the GLI developers will take care of the maintenance of the data loading code.

4.3 Other Relevant Benchmark Infrastructures

Outside the area of graph learning, there are various machine learning benchmark infrastructures that are remotely relevant to this work.

One relevant machine learning benchmark infrastructure is Papers With Code[11], which has a database of datasets in different domains of machine learning. Each dataset in this database is associated with types of machine learning tasks and a massive record of machine learning model performances, similarly as our design in Section 3. However, the performances are directly taken from papers or self-reported, and the experiment setups and data versions may not be well controlled.

More generally, there are a number of dataset search engines, such as Google Dataset Search[12], Microsoft Research Open Data[13], and DataMed[14]. These search engines index a huge and growing amount of datasets in various domains but does not contain detailed domain-specific dataset characteristics, such as the graph metrics as described in Section 3.2. These dataset are also usually not machine-learning ready, i.e., there is no data loading code that transforms these datasets into machine learning data loaders.

[8] https://ogb.stanford.edu/docs/dataset_overview/
[9] https://graph-learning-benchmarks.github.io/
[10] https://neurips.cc/Conferences/2021/CallForDatasetsBenchmarks
[11] https://paperswithcode.com/
[12] https://datasetsearch.research.google.com/
[13] https://msropendata.com/
[14] https://datamed.org/

5 Future Plan

In the future, there are a few directions that the GLI development team will focus on.

User experience. There is still room to further simplify the dataset contribution workflow, which will be one of the major focuses in our future development plan. As examples, we have planned to work on the following concrete improvements.

- *Helper functions for dataset conversion.* We plan to implement a few helper functions that can automatically convert commonly seen raw data formats into the GLI format.
- *Automatic generation of README documents.* We would like to implement a function that can automatically generate the README document for a dataset based on dataset characteristic and a few structured survey questions for the contributors.
- *Simplified submission interface*. While the Pull Request functionality of GitHub offers many advantages for dataset review (such as providing tests and reviews, and preserving review and commit history), the additional technical complexity brought by this process may be a concern. In the future, we may want to explore methods to automatically construct a Pull Request based on a simpler dataset submission interface.

Automatic benchmarking popular models. We plan to implement a service that can automatically benchmark popular models on new contributed datasets such that the model performance can be directly leveraged into the meta information of the datasets.

Citation tracking. We plan to track the citations to each dataset hosted on GLI. In this way, we can send an alert to the authors citing a dataset when critical issues/bugs are identified for the dataset.

Dataset exploration. We plan to implement an interface to explore and retrieve the datasets hosted on GLI, based on the database of the datasets described in Section 3.

Dataset license. A surprisingly large number of commonly used datasets lack an explicit license associated with them. Moreover, getting the right license for many existing datasets is a complicated task for a few reasons. First, many commonly used datasets, especially those created in early years, do not have a license. Second, while some datasets have a license, they are repurposed from an early version that does not have a license. It is unclear if such licenses are still valid. Finally, many datasets are released within a code repo. It is unclear if the license of the code repo could be viewed as the license to the datasets. As an important future step, we plan to take various measures to mitigate the license problem for datasets hosted on GLI. In particular, we will implement automated tools to enforce the license coverage for newly contributed datasets. We will also provide guidance on license choices for dataset contributors.

6 Conclusion

In this paper, we have introduced Graph Learning Indexer (GLI), a benchmark curation platform for graph learning. GLI is designed to solicit and curate massive benchmark datasets contributed by the community. With the contributor-centric design, we hope that GLI can better assist the community contribution on the development of benchmark datasets. We also hope that GLI can help improve our understanding on the taxonomy of graph learning tasks based on the rich meta information about the datasets.

References

[1] Patti J. Price, William M. Fisher, Jared Bernstein, and David S. Pallett. The darpa 1000-word resource management database for continuous speech recognition. In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, pages 651–654 vol.1, 1988. 1

[2] William M. Fisher, George R. Doddington, and Kathleen M. Goudie-Marshall. The darpa speech recognition research database: specifications and status. In *Proceedings of DARPAR Speech Recognition Workshop*, pages 93–99, 1986. 1

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1

[4] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019. 1, 2

[5] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020. 1, 8, 15

[6] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 2

[7] Sebastian Gehrmann, Tosin Adewumi, Karmanya Aggarwal, Pawan Sasanka Ammanamanchi, Anuoluwapo Aremu, Antoine Bosselut, Khyathi Raghavi Chandu, Miruna-Adriana Clinciu, Dipanjan Das, Kaustubh Dhole, et al. The gem benchmark: Natural language generation, its evaluation and metrics. In *Proceedings of the 1st Workshop on Natural Language Generation, Evaluation, and Metrics (GEM 2021)*, pages 96–120, 2021. 2

[8] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022. 2

[9] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019. 2, 5, 8, 18

[10] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 2, 8

[11] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 2, 15

[12] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016. 2, 15

[13] Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. Graph markup language (graphml), 2013. 3

[14] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021. 6

[15] Mark Newman. *Networks*. Oxford university press, 2018. 6

[16] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge university press, 2010. 6

[17] John Palowitch, Anton Tsitsulin, Brandon Mayer, and Bryan Perozzi. Graphworld: Fake graphs bring real insights for gnns. *arXiv preprint arXiv:2203.00112*, 2022. 6, 16

[18] Renming Liu, Semih Cantürk, Frederik Wenkel, Dylan Sandfelder, Devin Kreuzer, Anna Little, Sarah McGuire, Leslie O'Bray, Michael Perlmutter, Bastian Rieck, Matthew Hirn, Guy Wolf, and Ladislav Rampášek. Taxonomy of benchmarks in graph representation learning. *arXiv:2206.07729*, 2022. 7

[19] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021. 7, 15, 16, 17, 18

[20] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020. 7

[21] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020. 8

[22] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020. 8, 15

[23] Meng Liu, Youzhi Luo, Limei Wang, Yaochen Xie, Hao Yuan, Shurui Gui, Haiyang Yu, Zhao Xu, Jingtun Zhang, Yi Liu, Keqiang Yan, Haoran Liu, Cong Fu, Bora M Oztekin, Xuan Zhang, and Shuiwang Ji. DIG: A turnkey library for diving into graph deep learning research. *Journal of Machine Learning Research*, 22(240):1–9, 2021. URL http://jmlr.org/papers/v22/21-0343.html.

[24] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzman Lopez, Nicolas Collignon, and Rik Sarkar. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, page 4564–4573, 2021. 8

[25] Oleksandr Ferludin, Arno Eigenwillig, Martin Blais, Dustin Zelle, Jan Pfeifer, Alvaro Sanchez-Gonzalez, Sibon Li, Sami Abu-El-Haija, Peter Battaglia, Neslihan Bulut, et al. Tf-gnn: Graph neural networks in tensorflow. *arXiv preprint arXiv:2207.03522*, 2022. 8

[26] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and mining*, pages 807–816, 2009. 15

[27] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020. 15

[28] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh SPappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. In *Chemical Science*, pages 513–530, 2018. 15

[29] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. in international conference on learning representations. In *ICLR*, 2020. 15

[30] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. In *Quantitative Science Studies*, pages 396–413, 2020. 15

[31] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021. 15

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013. 15

[33] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. 15

[34] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL http://manikvarma.org/downloads/XC/XMLRepository.html. 15

[35] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019. 15

[36] TGene Ontology Consortium. The gene ontology resource: 20 years and still going strong. In *Nucleic Acids Research*, page 330=338, 2018. 15

[37] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019. 15

[38] WebKb Group. Cmu world wide knowledge base. URL http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/. 15

[39] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2011.12.021. URL https://www.sciencedirect.com/science/article/pii/S0378437111009186. 15

[40] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 1247–1250, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581026. doi: 10.1145/1376616.1376746. URL https://doi.org/10.1145/1376616.1376746. 15

[41] Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*, 2018. 15

[42] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26, 2013. 15

[43] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014. 15

[44] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1499–1509, 2015. 15

[45] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, 2005. 15

[46] Derek Lim and Austin R Benson. Expertise and dynamics within crowdsourced musical knowledge curation: A case study of the genius platform. In *ICWSM*, pages 373–384, 2021. 15

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. 15

[48] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*, 2017. 15

[49] Ankur Padia, Konstantinos Kalpakis, Francis Ferraro, and Tim Finin. Knowledge graph fact prediction via knowledge-enriched tensor factorization. *Journal of Web Semantics*, 59:100497, 2019. 15

[50] Benedek Rozemberczki and Rik Sarkar. Twitch gamers: a dataset for evaluating proximity preserving and structural role-based node embeddings. *arXiv preprint arXiv:2101.03091*, 2021. 15

[51] George A Miller. *WordNet: An electronic lexical dat @articlebordes2013translating, title=Translating embeddings for modeling multi-relational data, author=Bordes, Antoine and Usunier, Nicolas and Garcia-Duran, Alberto and Weston, Jason and Yakhnenko, Oksana, journal=Advances in neural information processing systems, volume=26, year=2013 abase*. MIT press, 1998. 15

[52] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013. 15

[53] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007. 15

[54] Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *7th biennial conference on innovative data systems research*. CIDR Conference, 2014. 15

[55] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 17

[56] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 17

[57] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 17

[58] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017. 17

[59] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019. 17

[60] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 18

[61] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 18

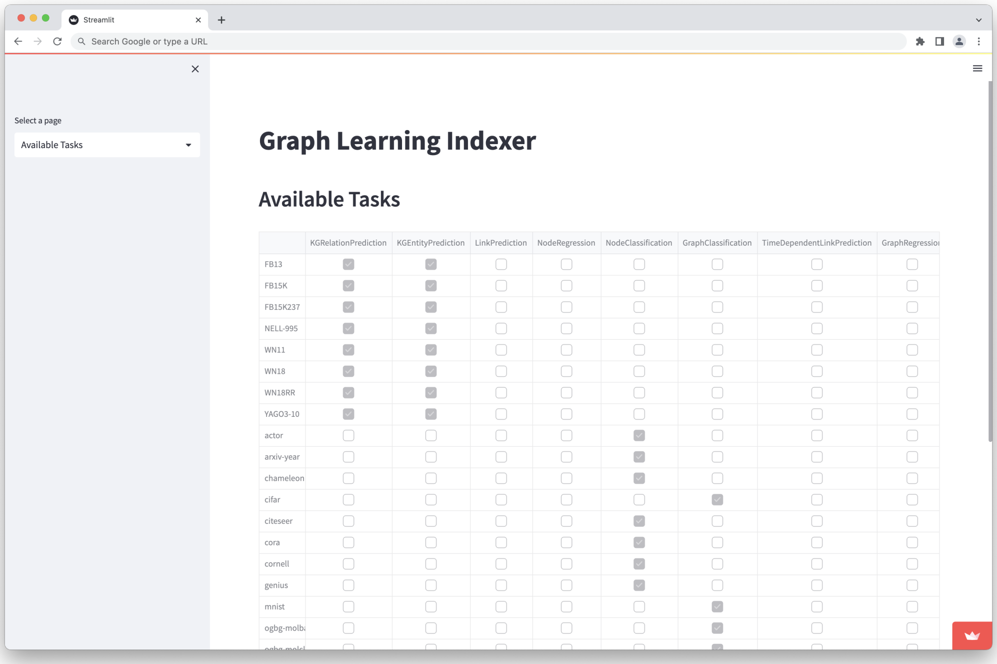
[62] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf. 18

[63] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf. 18

[64] Sergei Ivanov and Liudmila Prokhorenkova. Boost then convolve: Gradient boosting meets graph neural networks. In *International Conference on Learning Representations*, 2020. 18

[65] Jiuhai Chen, Jonas Mueller, Vassilis N Ioannidis, Soji Adeshina, Yangkun Wang, Tom Goldstein, and David Wipf. Does your graph need a confidence boost? convergent boosted smoothing on graphs with tabular node features. In *International Conference on Learning Representations*, 2021.

[66] Dmitrii Gavrilev, Nurlybek Amangeldiuly, Sergei Ivanov, and Evgeny Burnaev. High performance of gradient boosting in binding affinity prediction. *arXiv preprint arXiv:2205.07023*, 2022. 18



Graph Learning Indexer

Available Tasks

	KGRelationPrediction	KGEntityPrediction	LinkPrediction	NodeRegression	NodeClassification	GraphClassification	TimeDependentLinkPrediction	GraphRegressio
FB13	✓	✓	☐	☐	☐	☐	☐	☐
FB15K	✓	✓	☐	☐	☐	☐	☐	☐
FB15K237	✓	✓	☐	☐	☐	☐	☐	☐
NELL-995	✓	✓	☐	☐	☐	☐	☐	☐
WN11	✓	✓	☐	☐	☐	☐	☐	☐
WN18	✓	✓	☐	☐	☐	☐	☐	☐
WN18RR	✓	✓	☐	☐	☐	☐	☐	☐
YAGO3-10	✓	✓	☐	☐	☐	☐	☐	☐
actor	☐	☐	☐	☐	✓	☐	☐	☐
arxiv-year	☐	☐	☐	☐	✓	☐	☐	☐
chameleon	☐	☐	☐	☐	✓	☐	☐	☐
cifar	☐	☐	☐	☐	☐	✓	☐	☐
citeseer	☐	☐	☐	☐	✓	☐	☐	☐
cora	☐	☐	☐	☐	✓	☐	☐	☐
cornell	☐	☐	☐	☐	✓	☐	☐	☐
genius	☐	☐	☐	☐	✓	☐	☐	☐
mnist	☐	☐	☐	☐	☐	✓	☐	☐
ogbg-molb	☐	☐	☐	☐	☐	✓	☐	☐
ogbg-molcl	☐	☐	☐	☐	☐	✓	☐	☐

Figure 3: Available tasks table on GLI web page. The rows are datasets and the columns are pre-defined task types.

A List of Task Types

Currently, GLI supports the following task types[15]:

1. `NodeClassification`: Node classification task. This task aims to predict categorical node properties based on other nodes and its features in a graph.
2. `NodeRegression`: Node regression task. This task aims to predict continuous node properties based on other nodes and its features in a graph.
3. `GraphClassification`: Graph regression task. This task aims to predict categorical graph properties based on known graph's features.
4. `GraphRegression`: Graph classification task. This task aims to predict continuous graph properties based on known graph's features.
5. `LinkPrediction`: Link prediction task. This task aims to predict the existence of a link between two nodes in a graph.
6. `TimeDependentLinkPrediction`: Link prediction task, split by time. This task is the special case of `LinkPrediction`. Its train-validation-test split depends on the creation time of links.
7. `KGEntityPrediction`: Knowledge graph entity prediction task. This task aims to predict the tail or head node for a triplet in the graph.
8. `KGRelationPrediction`: Knowledge graph relation prediction task. This task aims to predict the relation type for a triplet in the graph.

We also add a table that lists all available tasks on our web page, as shown in Figure 3. This page will be updated periodically once new task configurations are contributed to GLI.

[15]See details at `https://anonymous.4open.science/r/gli-updated-3D70/FORMAT.md`.

B Reference of Datasets

Table 2 summarizes the original source, current version and previous versions of the datasets that we have incorporated.

Table 2: Reference of datasets.

Dataset	Original	Cur	Prev	Dataset	Original	Cur	Prev
actor	[26]	[27]	/	ogbg-molpcba	[28]	[5]	[29]
arxiv-year	[30]	[19]	[5]	ogbl-collab	[30]	[5]	/
chameleon	[31]	[27]	/	ogbn-arxiv	[30]	[5]	[32]
cifar	[33]	[22]	/	ogbn-mag	[30]	[5]	/
citeseer	[11]	[12]	/	ogbn-products	[34]	[5]	[35]
cora	[11]	[12]	/	ogbn-proteins	[36]	[5]	[37]
cornell	[38]	[27]	/	penn94	[39]	[19]	/
FB13	[40]	[41]	[42]	pokec	[43]	[19]	/
FB15K	[40]	[41]	[44]	pubmed	[11]	[12]	/
FB15K237	[40]	[41]	[44]	snap-patents	[43]	[19]	[45]
genius	[46]	[19]	/	squirrel	[31]	[27]	/
mnist	[47]	[22]	/	texas	[38]	[27]	/
NELL-995	[48]	[49]	[41]	twitch-gamers	[50]	[19]	/
ogbg-molbace	[28]	[5]	[29]	wiki	[19]	[19]	/
ogbg-molclintox	[28]	[5]	[29]	wiscousin	[38]	[27]	/
ogbg-molfreesolv	[28]	[5]	[29]	WN11	[51]	[41]	/
ogbg-molhiv	[28]	[5]	[29]	WN18	[51]	[41]	[52]
ogbg-molmuv	[28]	[5]	[29]	WN18RR	[51]	[41]	[52]
ogbg-molsider	[28]	[5]	[29]	YAGO3-10	[53]	[41]	[54]

C Definitions of Graph Data Properties

Here we introduce the formal definitions of the graph data properties mentioned in Section 3.2. Given a graph $G = (V, E)$, where $V = \{1, 2, \ldots, N\}$ is the set of N nodes and $E \subseteq V \times V$ is the set of edges. Denote $M = |E|$. Assume $X = \mathbb{R}^{N\times D}$ is the matrix of node features, where D is the feature dimension. Also assume $Y = \{1, 2, \ldots, C\}^N$ is the vector of node labels, where C is the number of classes.

C.1 Basic

Is Directed: Whether the graph is a directed graph.

Number of Nodes: The number of nodes N.

Number of Edges: The number of edges M.

Edge Density: The edge density is defined as $\frac{2M}{N(N-1)}$ for undirected graph and $\frac{M}{N(N-1)}$ for directed graph.

Average Degree: The average degree is defined as $\frac{2M}{N}$ for undirected graph and $\frac{M}{N}$ for directed graph.

Edge Reciprocity: The edge reciprocity of a directed graph is defined as $\frac{\overset{\leftrightarrow}{M}}{M}$, where $\overset{\leftrightarrow}{M}$ denotes the number of edges pointing in both directions.

Degree Assortativity: The degree assortativity is defined as the average Pearson correlation coefficient of degree between all pairs of linked nodes.

C.2 Distance

Diameter: The maximum pairwise shortest path distance in the graph.

Pseudo Diameter: The pseudo diameter approximates diameter, which serves as a lower bound of the exact value of diameter.

Average Shortest Path Length: The average of all the pairwise shortest path distance in the graph.

Global Efficiency: The efficiency between a pair of nodes is the multiplicative inverse of the shortest path distance and the global efficiency is the average efficiency of all pairs of nodes in the graph.

C.3 Connectivity

Relative Size of LCC: The relative size of the largest connected component is defined as the ratio between the size of the largest connected component and N.

Relative Size of LSCC: The relative size of the largest strongly connected component is defined as the ratio between the size of the largest strongly connected component and N.

Average Node Connectivity: The local node connectivity for two non-adjacent nodes u and v is the minimum number of nodes that must be removed in order to disconnect them and the average node connectivity is the average local node connectivity of all pairs of two non-adjacent nodes in the graph.

C.4 Clustering

Average Clustering Coefficient: The local clustering coefficient for node u is defined as $\frac{2}{deg(u)(deg(u)-1)}T(u)$ for undirected graph, where $T(u)$ is the number of triangles passing through node u and $deg(u)$ is the degree of node u; and defined as $\frac{2}{deg^{tot}(u)(deg^{tot}(u)-1)-2deg^{\leftrightarrow}(u)}T(u)$ for directed graph, where $T(u)$ is the number of directed triangles through node u, $deg^{tot}(u)$ is the sum of in degree and out degree of node u and $deg^{\leftrightarrow}(u)$ is the reciprocal degree of u and average clustering coefficient is the average local clustering of all the nodes in the graph.

Transitivity: The fraction of all possible triangles present in the graph, which is defined as $3\frac{\#triangles}{\#triads}$, where a $triad$ is a pair of two edges with a shared vertex.

Degeneracy: The least integer k such that every induced subgraph of the graph contains a vertex with k or fewer neighbors.

C.5 Distribution

Power Law Exponent: The exponent parameter of a Power-law distribution that best fits the degree-sequence distribution of the graph.

Pareto Exponent: The exponent parameter of a Pareto distribution that best fits the degree-sequence distribution of the graph.

Gini Coefficient of Degree: The Gini coefficient of the the degree-sequence of the graph.

Gini Coefficient of Coreness: The Gini coefficient of the the coreness-sequence of the graph, where the coreness of a node u indicates the largest integer k of a k-core containing node u.

C.6 Attribute

Edge Homogeneity [17]: The ratio of edges that connect nodes with the same node labels.

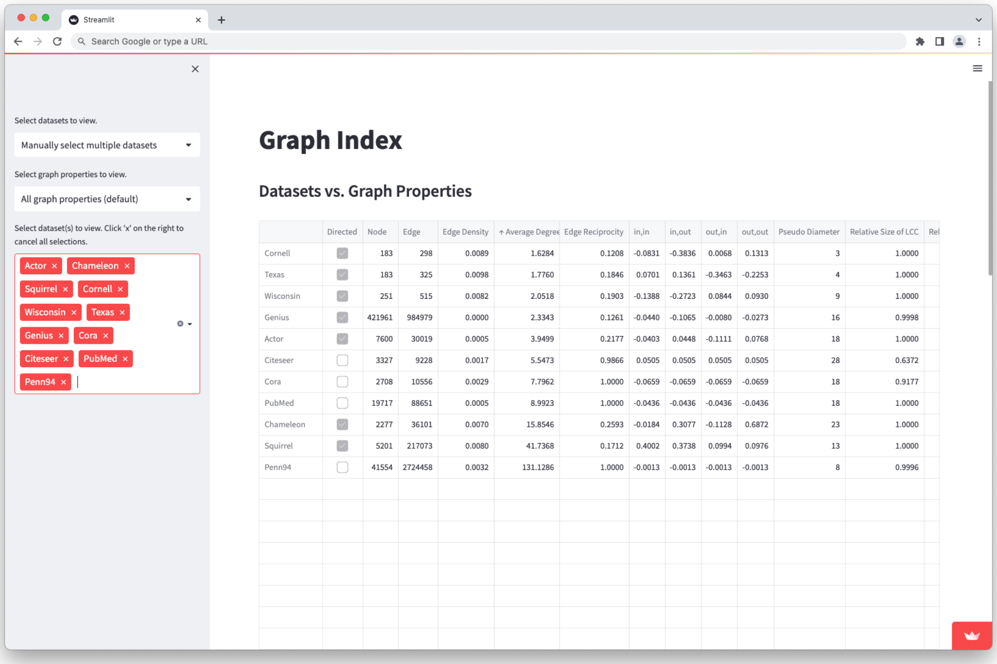
Average Within-Class Feature Angular Similarity [17]: Within-class angular feature similarity is $1 - angular\_distance(X_i, X_j)$ for an edge with its endpoints i and j with the same node labels and average within-class angular feature similarity is the average of all such edges in the graph.

Average Between-Class Feature Angular Similarity [17]: Between-class angular feature similarity is $1 - angular\_distance(X_i, X_j)$ for an edge with its endpoints i and j with different node labels and average between-class angular feature similarity is the average of all such edges in the graph.

Feature Angular SNR [17]: The ratio between average within-class feature angular similarity and average between-class feature angular similarity.

Homophily Measure [19]: The homophily measure is defined as

$$\hat{h} = \frac{1}{C-1}\sum_{k=1}^{C}[h_k - \frac{|C_k|}{N}]_+, \tag{1}$$



Select datasets to view.

Manually select multiple datasets

Select graph properties to view.

All graph properties (default)

Select dataset(s) to view. Click 'x' on the right to cancel all selections.

Actor × Chameleon × Squirrel × Cornell × Wisconsin × Texas × Genius × Cora × Citeseer × PubMed × Penn94 ×

Graph Index

Datasets vs. Graph Properties

	Directed	Node	Edge	Edge Density	↑ Average Degree	Edge Reciprocity	in,in	in,out	out,in	out,out	Pseudo Diameter	Relative Size of LCC	Rel
Cornell	✓	183	298	0.0089	1.6284	0.1208	-0.0831	-0.3836	0.0068	0.1313	3	1.0000	
Texas	✓	183	325	0.0098	1.7760	0.1846	0.0701	0.1361	-0.3463	-0.2253	4	1.0000	
Wisconsin	✓	251	515	0.0082	2.0518	0.1903	-0.1388	-0.2723	0.0844	0.0930	9	1.0000	
Genius	✓	421961	984979	0.0000	2.3343	0.1261	-0.0440	-0.1065	-0.0080	-0.0273	16	0.9998	
Actor	✓	7600	30019	0.0005	3.9499	0.2177	-0.0403	0.0448	-0.1111	0.0768	18	1.0000	
Citeseer	☐	3327	9228	0.0017	5.5473	0.9866	0.0505	0.0505	0.0505	0.0505	28	0.6372	
Cora	☐	2708	10556	0.0029	7.7962	1.0000	-0.0659	-0.0659	-0.0659	-0.0659	18	0.9177	
PubMed	☐	19717	88651	0.0005	8.9923	1.0000	-0.0436	-0.0436	-0.0436	-0.0436	18	1.0000	
Chameleon	✓	2277	36101	0.0070	15.8546	0.2593	-0.0184	0.3077	-0.1128	0.6872	23	1.0000	
Squirrel	✓	5201	217073	0.0080	41.7368	0.1712	0.4002	0.3738	0.0994	0.0976	13	1.0000	
Penn94	☐	41554	2724458	0.0032	131.1286	1.0000	-0.0013	-0.0013	-0.0013	-0.0013	8	0.9996	

Figure 4: An interactive web page showing the graph properties for each dataset. The left sidebar shows which datasets are selected. The entries are sorted according to the average degree of nodes in this demonstration.

where $[a]_+ = max(a, 0)$, $|C_k|$ is the number of nodes with node label k and h_k is the class-wise homophily metric defined below,

$$h_k = \frac{\sum_{u:Y_u=k} d_u^{(Y_u)}}{\sum_{u:Y_u=k} d_u}, \tag{2}$$

where d_u is the number of neighbors of node u and $d_u^{(Y_u)}$ is the number of neighbors of node u that have the same class label.

Attribute Assortativity: The attribute assortativity is defined as the average Pearson correlation coefficient of the attribute (class labels) between all pairs of linked nodes.

C.7 Visualization

We create a web page to show the aforementioned graph data properties, as shown in Figure 4. We use Streamlit[16] to build and host the website. Users can select multiple datasets and graph properties, and sort by a graph property for a quick comparison.

D Benchmark Experiment Setup

In this section, we describe more details of the experiment setup[17].

We set GCN [55], GAT [56], GraphSAGE [57], MoNet [58] , MLP, and MixHop [59] to have two layers in the benchmark setting. For LINKX [19], we set MLP_A, MLP_X to be a one-layer network and MLP_f to be a two-layers network, following Lim et al. [19].

[16]`https://streamlit.io/cloud`. Unfortunately, due to the anonymity requirement, we are not able to attach the link to the web page.

[17]Please see more details about how to use the benchmark code at `https://anonymous.4open.science/r/gli-updated-3D70/benchmarks/NodeClassification/README.md`.

In order to make a fair comparison, we adopt the same training configuration for all experiments. We use Adam [60] as optimizer for all models except LINKX. AdamW [61] is used with LINKX in order to stay the same with Lim et al. [19]. For all binary classification datasets (penn94, pokec, genius and twitch-gamers), we choose ROC AUC as evaluation metric. For other datasets, test accuracy is used.

Our implementaions of GCN, GAT, GraphSAGE and MoNet are based on DGL [9]. When implementing the models, we reserve default settings in DGL implementation as much as possible. For MixHop and LINKX, we adopt the implementation of Lim et al. [19]. The detailed settings for different models are:

- GAT: Number of heads in multi-head attention = 8. leakyReLU angle of negative slope = 0.2. No residual is applied. Dropout rate on attention weight is the same as overall dropout.
- GraphSAGE: Aggregator type is GCN. No norm is applied.
- MoNet: Number of kernels = 3. Dimension of pseudo-coordinte = 2. Aggregator type = sum.
- MixHop: List of powers of adjacency matrix = $[1, 2, 3]$. No norm is applied.
- LINKX: MLP_A, MLP_X are both one-layer network and MLP_f is a two-layers network. AdamW is used as optimizer. No inner activation.

Hyperparameter tunning. Random search on the following hyperparameter tuning range is performed for every model.

- Hidden size: $\{32, 64\}$
- Learning rate: $\{.001, .005, .01, .1\}$
- Dropout rate: $\{.2, .4, .6, .8\}$
- Weight decay: $\{.0001, .001, .01, .1\}$

We generate 100 random configurations for each model, where each random configuration is run for 5 times on each dataset. The max training epoch number is 10000. We apply early stopping where training is stopped if the validation accuracy does not improve for 50 epochs. When training is finished, we load the weights of models with highest validation accuracy on the dataset. Test accuracy and standard deviation are reported in Table 1.

Gradient Boosting Decision Tree (GBDT) models. We also include two GBDT models, CatBoost [62] and LightGBM [63], which are shown to be strong baselines [64–66]. We train both models for at most 1000 epoches with early stopping if validation accuracy does not improve for 100 epochs. We apply grid search on the following hyperparameters, and we have 5 independent trials for each hyperparameter configuration.

Hyperparameters for CatBoost:

- learning rate: $\{.01, .1\}$;
- depth: $\{4, 6\}$.

Hyperparameters for LightGBM:

- learning rate: $\{.01, .1\}$;
- number of leaves $\{15, 63\}$.

E Package Maintenance

This section outlines the designs of GLI that aim to ensure long-term viability and usability as an open-source project.

E.1 Open Source License

GLI adopts the MIT License, aligning with our principle to favor broader application and trustworthiness of graph learning. By using MIT License, we only "require preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code."[18]

[18] `https://choosealicense.com/licenses/mit/`

E.2 Package Indexing

Currently, GLI provides a `setup.py` to facilitate the installation from the source. Moreover, we divide the package dependencies into three categories: `default`, `test`, and `doc` to meet different needs of users and potential contributors. We tested and successfully installed GLI on popular operating systems (Windows 11, MacOS with M1, Ubuntu, and CentOS). As a part of future work, we will use package indexing tools, including PyPI and Anaconda, to package the GLI project.

E.3 Documentation

Automatic deployment. GLI uses `sphinx`[19] and `autosummary`[20] to generate API references automatically from docstrings in source codes. The docstrings are written in NumPy format[21] for better readability, in comparison to the common Sphinx format.

Structure. Figure 5 shows the main page of GLI's documents. The web page has three main sections: "Get Started", "Modules", and "Data" as shown on the left toctree. The "Get Started" section contains an instruction on installation, and a tutorial for examples of basic usages and contributor guidelines. The "Modules" section contains the API references to core modules in GLI. The "Data" section illustrates GLI's file-based storage API.

E.4 Contributor Guidelines

We position contributor guidelines in two places: `CONTRIBUTING.md` in GitHub repository and "Contributor Guidelines" section in the aforementioned online document page. The contributor guidelines include the installation of the development environment, the steps to make contribution, and the development practices to follow. In particular, we distinguish three kinds of contribution: new dataset, new feature, and bug fix and ask contributors to follow different guidelines correspondingly.

E.5 Tutorial

To flatten the learning curve for new users and potential contributors, we prepared a brief tutorial for their reference. The tutorial starts with an explanation of GLI's architecture, and follows with examples of data-loading and downstream tasks. For example, to train a GCN on Cora node classification task.

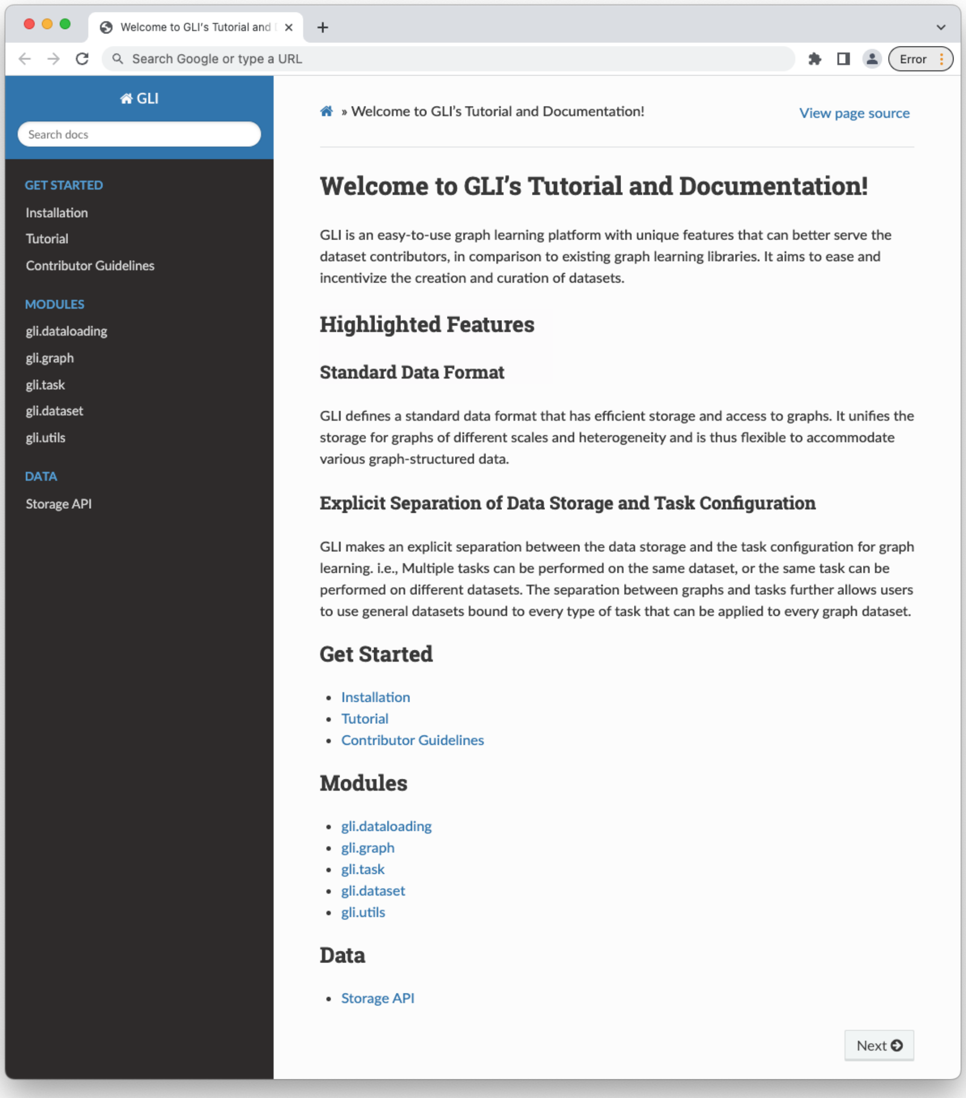
E.6 Code Quality

GLI uses `pylint`, `pycodestyle`, and `pydocstyle` to ensure the code quality. Specifically, we have followed Google Python Style Guide[22] to configure the automatic linting and style checking tools. Moreover, they are enforced through both pre-commit hooks locally and continuous integration remotely. Besides, GLI uses `pytest` to help developers test whether a new patch violate the correctness of the codes. The testing is designed to cover all the core modules of GLI, including `gli.graph`, `gli.task`, `gli.dataset` and `gli.dataloading`. Users can run testing locally before they open a pull request. We also set up the continuous integration to run testing on GitHub, and enforce that a pull request must pass all tests before merging.

E.7 Others

GLI uses `Makefile` to facilitate the development. Contributors can run `make test` to run the aforementioned tests locally to test the core modules on all datasets. Alternatively, one can specify a single dataset to test by `make pytest DATASET=<dataset name>`, which is a common scenario for dataset contribution.

[19] `https://www.sphinx-doc.org/en/master/`
[20] `https://www.sphinx-doc.org/en/master/usage/extensions/autosummary.html`
[21] `https://numpydoc.readthedocs.io/en/latest/format.html`
[22] `https://google.github.io/styleguide/pyguide.html`



Welcome to GLI's Tutorial and Documentation!

GLI is an easy-to-use graph learning platform with unique features that can better serve the dataset contributors, in comparison to existing graph learning libraries. It aims to ease and incentivize the creation and curation of datasets.

Highlighted Features

Standard Data Format

GLI defines a standard data format that has efficient storage and access to graphs. It unifies the storage for graphs of different scales and heterogeneity and is thus flexible to accommodate various graph-structured data.

Explicit Separation of Data Storage and Task Configuration

GLI makes an explicit separation between the data storage and the task configuration for graph learning. i.e., Multiple tasks can be performed on the same dataset, or the same task can be performed on different datasets. The separation between graphs and tasks further allows users to use general datasets bound to every type of task that can be applied to every graph dataset.

Get Started

- Installation
- Tutorial
- Contributor Guidelines

Modules

- gli.dataloading
- gli.graph
- gli.task
- gli.dataset
- gli.utils

Data

- Storage API

Figure 5: Preview of GLI document page.