FedGraph: A Research Library and Benchmark for Federated Graph Learning

Yuhang Yao[1], **Yuan Li**[1], **Xinyi Fan**[1], **Junhao Li**[1], **Kay Liu**[2], **Weizhao Jin**[3], **Yu Yang**[1], **Srivatsan Ravi**[3], **Philip S. Yu**[2], **Carlee Joe-Wong**[1]
[1]Carnegie Mellon University [2]University of Illinois Chicago [3]University of Southern California
yuhangya@alumni.cmu.edu

Abstract

Federated graph learning is an emerging field with significant practical challenges. While algorithms have been proposed to improve the accuracy of training graph neural networks, such as node classification on federated graphs, the system performance is often overlooked, despite it is crucial for real-world deployment. To bridge this gap, we introduce FedGraph, a research library designed for practical distributed training and comprehensive benchmarking of FGL algorithms. FedGraph supports a range of state-of-the-art graph learning methods and includes a monitoring class that evaluates system performance, with a particular focus on communication and computation costs during training. Unlike existing federated learning platforms, FedGraph natively integrates homomorphic encryption to enhance privacy preservation and supports scalable deployment across multiple physical machines with system-level performance evaluation to guide the system design of future algorithms. To enhance efficiency and privacy, we propose a low-rank communication scheme for algorithms like FedGCN that require pre-training communication, accelerating both the pre-training and training phases. Extensive experiments benchmark FGL algorithms on three major graph learning tasks and demonstrate FedGraph as the first efficient FGL framework to support encrypted low-rank communication and scale to graphs with 100 million nodes. [1] [2]

1 Introduction

Graph neural networks aim to learn representations of graph-structured data that capture features associated with graph nodes and edges between them (Bronstein et al., 2017). Most graph applications can modeled as one of three major graph learning problems: node classification (e.g., classifying nodes representing papers in citation networks based on the research topic), link prediction (e.g., recommending the formation of links that represent friendship between users), or graph classification (e.g. classifying types of proteins in biology, where each protein is represented as a graph). Figure 1 (left) illustrates these graph learning tasks (Benamira et al., 2019; Zhang et al., 2020b).

In practice, graph data is often too large for a single server or naturally distributed across clients. For instance, learning from billions of website visits requires more resources than one server can provide. Even if centralized storage were possible, privacy laws such as GDPR in Europe and PAPG in India restrict cross-border data sharing, and users may be unwilling to share personal data with external servers. To address these challenges, federated learning enables training accurate models on decentralized data while preserving privacy (Zhao et al., 2018).

[1]Code available at `https://github.com/FedGraph/fedgraph`.

[2]Documentation available at `https://docs.fedgraph.org/en/latest/`.

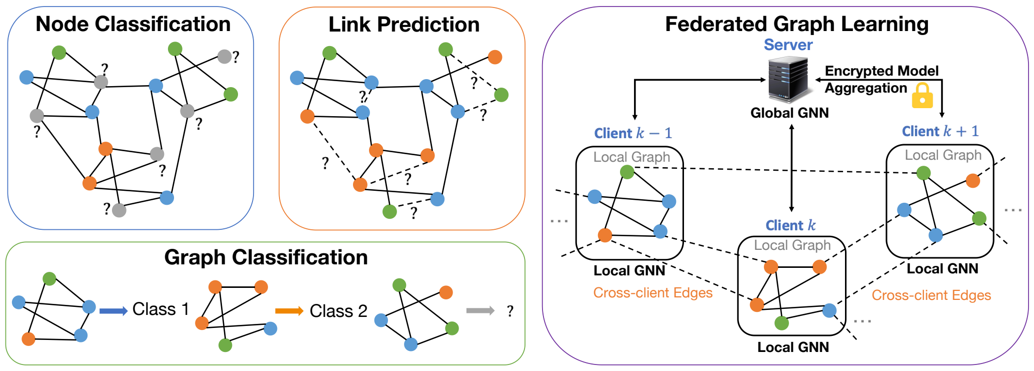


Figure 1: Modeling Applications as Graph Tasks (left) and Federated Graph Training (right). Node classification predicts labels of grey nodes from neighbors, link prediction infers future edges, and graph classification assigns labels to whole graphs. In federated graph learning, nodes span multiple clients with cross-client edges. Each client trains a local GNN on its subgraph and shares encrypted updates with a server, enabling collaborative learning without exposing raw data.

These challenges motivate *Federated Graph Learning* (FGL) as a key research area (Liu et al., 2024). In FGL (Figure 1, right), each client trains a local graph and Graph Neural Network (GNN) model, which are then aggregated at a coordinator server. Different FGL algorithms vary in how they update and aggregate local models, affecting accuracy and system performance (e.g., runtime, communication cost)(Xie et al., 2021; Zhang et al., 2021; Yao et al., 2024). While model accuracy can be tested on open datasets(Hu et al., 2020), evaluating real-world system performance requires benchmarking platforms. Existing FGL benchmarks mostly simulate multiple clients on one machine (Xie et al., 2021; Li et al., 2024), failing to reflect real communication and computation costs. Current libraries also lack mature GNN support: FedScale (Lai et al., 2022) excludes graph models, while FedGraphNN (He et al., 2021) and FederatedScope-GNN (Wang et al., 2022) are outdated and limited to simple methods like FedAvg. Users must implement key FGL features themselves, such as handling cross-client edges, and no platform natively supports advanced techniques like homomorphic encryption (Jin et al., 2023; Yao et al., 2024).

To meet these shortcomings, we introduce *FedGraph*, a research library to easily train GNNs in federated settings. As shown in Table 1, FedGraph supports various federated training methods of graph neural networks under both simulated and real federated environments, as well as encrypted communication between clients and the central server for model update and information aggregation.

Table 1: Comparison with Existing Frameworks. FedGraph supports distributed FGL, cross-client edges, encrypted aggregation, and system-level profiling for large-scale optimization.

	Vanilla FGL	FedScale	FedGraphNN	FederatedScope-GNN	**FedGraph**
Distributed Training	✗	✓	✓	✓	✓
Graph Learning	✓	✗	✓	✓	✓
Multiple FGL Algorithms	✓	✗	✗	○	✓
Cross-Client Edges	✗	✗	✗	✗	✓
Encrypted Aggregation	✗	✗	✗	✗	✓
System Level Profiler	✗	✗	✗	✗	✓
Large Scale ML Optimizations	✗	✗	✗	✗	✓

We summarize the **contributions** of FedGraph as follows.

- FedGraph is the *first* Python library tailored for real-world federated graph learning (FGL), integrating system optimizations for efficiency, scalability, and privacy, along with multiple state-of-the-art algorithms for easy comparison.
- It natively supports *homomorphic encrypted aggregation* for privacy-preserving training and offers a *system-level monitor* to analyze communication and computation overhead.
- A low-rank communication scheme is introduced for methods such as FedGCN, reducing both pre-training and training communication costs.
- Extensive experiments benchmark three FGL tasks and demonstrate scalability to privacy-preserving training on graphs with up to 100 million nodes.

In this paper, we first overview the system design in Section 2, followed by highlighting the key system components in Section 3. In Section 4, we present a case study demonstrating how FedGraph facilitates the design and test of low-rank pre-training communication in FGL. We then benchmark the performance on three tasks and evaluate its scalability in Sections 5, and conclude in Appendix 6.

2 FedGraph System Design

In this section, we outline the design principles of the FedGraph library and demonstrate how these principles are implemented in our system design (Figure 2).

2.1 Design Principles

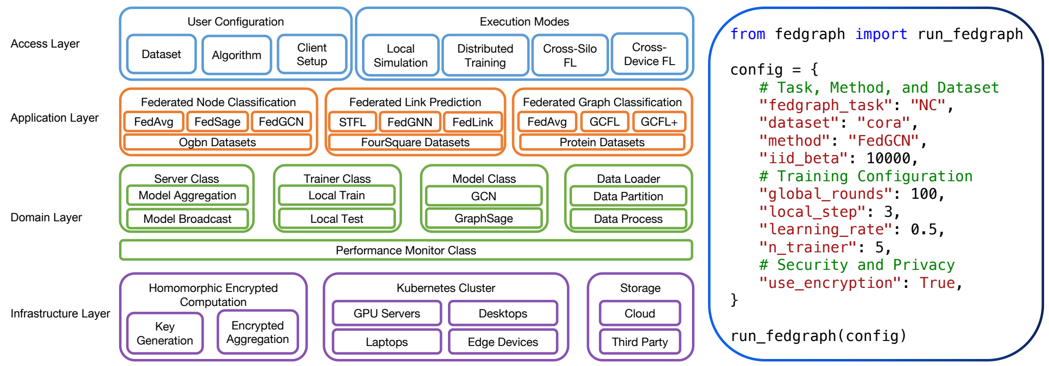
The main focus of FedGraph is providing a scalable and privacy-preserving federated graph learning system with ease of use for federated learning researchers and applied scientists in industry. As illustrated in Figure 2 (left), the system architecture is structured according to four design principles.

Optimized usability: At the access layer, users can configure training with 10-20 lines of code and seamlessly switch between local simulation and federated training, as detailed in Section 2.2. FedGraph abstracts away complexity, offering a unified training and evaluation platform.

Benchmarking methods: At the application layer, FedGraph supports three FGL tasks (node, link, and graph classification) and a wide range of state-of-the-art algorithms, listed in Appendix D.

Extensibility: At the domain layer, modular components (e.g., data loaders, trainer classes) allow easy extension to new datasets and algorithms.

Scalability and privacy: At the infrastructure layer, FedGraph leverages Ray and Kubernetes for distributed training, with optional homomorphic encryption to secure aggregation, supporting large-scale, privacy-preserving FGL.



```
from fedgraph import run_fedgraph

config = {
    # Task, Method, and Dataset
    "fedgraph_task": "NC",
    "dataset": "cora",
    "method": "FedGCN",
    "iid_beta": 10000,
    # Training Configuration
    "global_rounds": 100,
    "local_step": 3,
    "learning_rate": 0.5,
    "n_trainer": 5,
    # Security and Privacy
    "use_encryption": True,
}

run_fedgraph(config)
```

Figure 2: Design Diagram of FedGraph (left) and Quick Start Example (right). The system is organized into four layers: the user access layer, the application layer, the domain layer, and the infrastructure layer. Users only need to focus on the access layer, while the developers can focus on one of the remaining layers based on the domain knowledge.

2.2 FedGraph Use Example

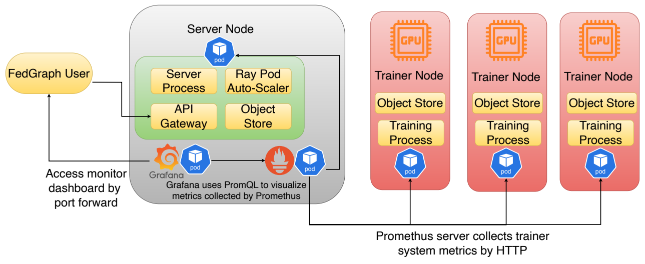
Researchers can install FedGraph with `pip install fedgraph`, configure experiments, and start training federated GNN models. Details on code structure and API are in Appendices B, C, and E. As shown in Figure 2 (right), users set up experiments through a simple configuration: **Task, method, and dataset specification:** define the task, algorithm, dataset, and client distribution, with enforced task-method compatibility for reproducibility. **Training configuration:** specify hyperparameters such as global rounds, local steps, learning rate, and trainer count. **Security and privacy:** enable homomorphic encryption for secure aggregation. **Execution:** run `run_fedgraph(config)` to automatically launch data loading, client initialization, and distributed training.

3 FedGraph System Highlights

In this section, we first introduce FedGraph's monitoring system, which enables usable benchmarking of FGL methods. We then introduce two infrastructure features, FedGraph Homomorphic Encryption for privacy-preserving aggregation and FedGraph Kubernetes for scalable distributed training. Finally, we discuss supported configurations for users to optimize large-scale model training.

3.1 FedGraph Monitoring System

The system(Figure 3) uses a Monitor Class to track runtime, CPU/GPU utilization, memory use, and communication costs between server and clients, with dashboards shown in Figure 9. **Communication cost**: logs transfer rates to identify bottlenecks. **Training time and accuracy**: records duration and model accuracy for comparison. **Resource usage**: tracks CPU/GPU and memory across components (e.g., Server Process, Ray AutoScaler) to reveal inefficiencies or leaks. Section 5 further illustrates resource profiles.



Server Node
FedGraph User
Server Process
Ray Pod Auto-Scaler
API Gateway
Object Store
Grafana
Access monitor dashboard by port forward
Grafana uses PromQL to visualize metrics collected by Promethus
Trainer Node
Object Store
Training Process
Promethus server collects trainer system metrics by HTTP

Figure 3: FedGraph Monitoring System Architecture.

3.2 FedGraph Homomorphic Encryption

FedGraph supports homomorphic encryption (HE) for end-to-end secure computation (Figure 4), making it the first FGL library with native HE support. It secures both (i) feature aggregation before training (Yao et al., 2024) and (ii) model aggregation during training (Zhang et al., 2024; Kim et al., 2025). **Pre-training aggregation**: clients encrypt node features, the server aggregates ciphertexts, and clients decrypt results without exposing raw features. **Training aggregation**: clients encrypt model updates, and the server aggregates them without plaintext access (Zhang et al., 2020a; Jin et al., 2023).

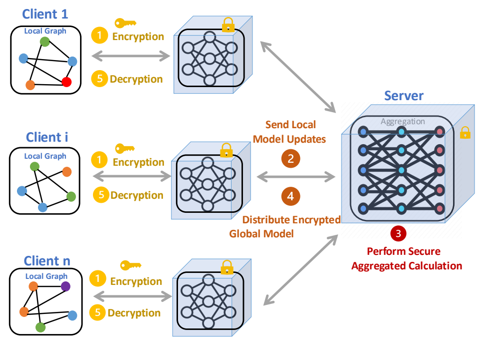


Figure 4: FedGraph Homomorphic Encryption for Secure Aggregation.

Homomorphic Encryption Overhead: Figure 5 shows training time and communication cost for FedGCN with and without HE. As FedGCN involves pre-training and training aggregation, it highlights HE overhead, particularly where feature matrices exceed model parameters. This motivates communication-efficient techniques (Section 4); detailed benchmarks are in Appendix F.

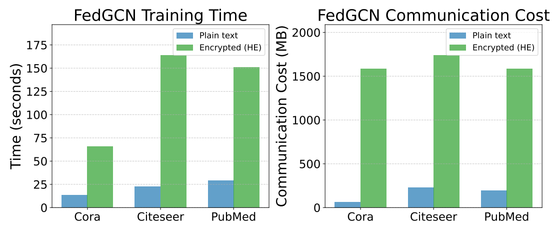


Figure 5: FedGCN performance on plaintext vs. HE.

3.3 FedGraph Kubernetes

FedGraph employs Kubernetes to support diverse workloads and large client populations. Unlike prior work (Table 1), it integrates natively with AWS Elastic Kubernetes Service for flexible resource management. A self-managed cluster runs intensive tasks, with a master node handling orchestration and worker nodes executing distributed training. The Kubernetes Cluster Autoscaler dynamically adjusts resources, ensuring efficiency under varying loads.

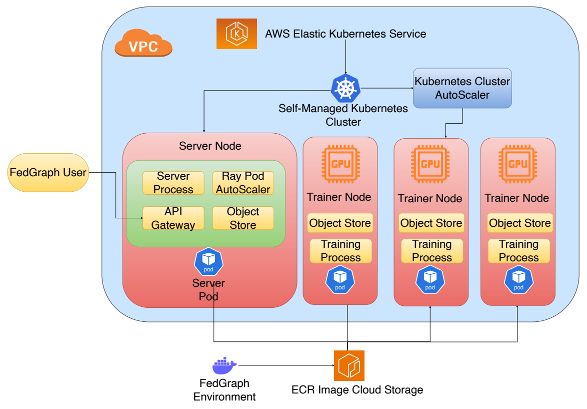


Figure 6: Kubernetes Service for Cluster Management.

3.4 Optimizations for Scalability

FedGraph scales to graphs with millions of nodes through several optimizations. **Client selection**: selectively engages a fraction of clients per round (Li et al., 2019), reducing communication and client overhead. **Minibatch training**: processes graph subsets to lower computation and memory use, enabling participation by resource-limited devices. **Communication and resource optimization**: minimizes data transfer and leverages Kubernetes to dynamically manage pods.

4 Case Study: Communication- and Computation-Efficient Federated Node Classification with Low-Rank Feature Compression

Communication is a major challenge in federated graph learning, especially with large graphs or privacy-preserving methods like homomorphic encryption (HE). To address this, FedGraph integrates low-rank feature compression, reducing overhead while maintaining accuracy.

Architecture Support in FedGraph: FedGraph's modular design separates pre-training feature aggregation and model training, enabling tailored optimizations. In this case study, HE is applied in both phases, while low-rank compression is used during pre-training. The HE interface naturally supports low-rank encrypted aggregation due to its additive structure.

Low-Rank Pre-Training Feature Aggregation: In FedGCN, pre-training communication aggregates cross-client node features, often encrypted for privacy (Yao et al., 2024). We compress these features via random projection: the server generates a matrix $\mathbf{P} \in \mathbb{R}^{d\times k}$ ($k \ll d$) and distributes it to clients. Each client computes $\hat{\mathbf{X}}_i = \mathbf{X}_i\mathbf{P}$, sends it to the server, which aggregates $\hat{\mathbf{X}}_{agg} = \sum_{i=1}^{m} \hat{\mathbf{X}}_i$, and redistributes the result. To prevent inversion attacks, $\mathbf{P}$ may itself be encrypted. This method cuts communication in both directions while preserving privacy.

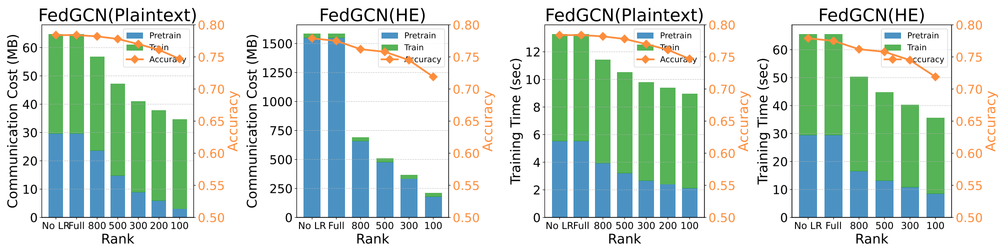


Figure 7: Comparison of communication cost (first and second plots) and training time (third and fourth plots) for FedGCN with different low-rank compression settings. We show results with plaintext and with HE. Each bar represents the total cost divided into pre-training (blue) and training (green) phases, with accuracy plotted as an orange line to show performance trade-offs.

Performance Evaluation: We evaluate low-rank compression in FedGCN on the Cora dataset, varying rank from the full 1433 features down to 100, achieving up to 93% reduction. Figure 7 compares communication cost, training time, and accuracy. HE alone introduces large overhead, especially in pre-training, which is substantially reduced with low-rank projection. Accuracy remains stable across ranks, showing that low-rank compression effectively balances efficiency and privacy.

5 Benchmarking FedGraph on Graph Learning Tasks and Scalability

We benchmark FedGraph on graph classification, node classification, and link prediction, and then evaluate its scalability on larger datasets. Additional results are in Appendix G.

5.1 Benchmarking Federated Graph Learning Tasks

We benchmark FedGraph on three tasks-graph classification, node classification, and link prediction-evaluating accuracy, training time, and communication cost. **Federated Graph Classification**: On five datasets (IMDB-BINARY, IMDB-MULTI, MUTAG, BZR, COX2), GCFL+ and GCFL+dWs achieve the highest accuracy but at much higher training and communication costs, especially on

IMDB. FedAvg is fastest and most communication-efficient, making it suitable for constrained settings (Figure 8a). **Federated Node Classification**: On Cora, Citeseer, and PubMed, FedGCN outperforms FedAvg in accuracy but incurs large pre-training communication overhead from feature aggregation. These costs align with theory and motivate the low-rank method in Section 4 (Figure 8b). **Federated Link Prediction**: Using the Foursquare dataset (Yang et al., 2016), FedLink and STFL reach the highest AUC but with the greatest time and communication costs. 4D-FED-GNN+ trains fastest, while StaticGNN is most network-efficient (Figure 8c).

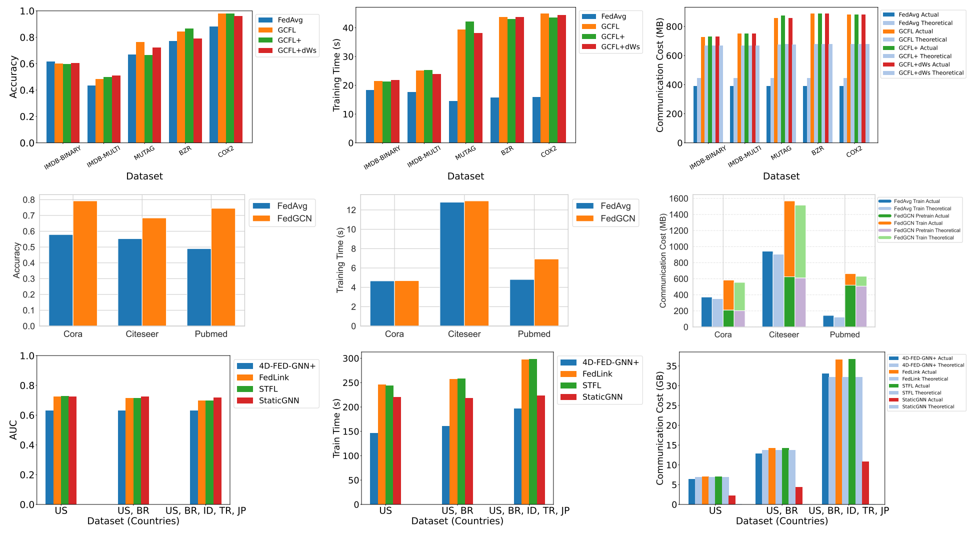
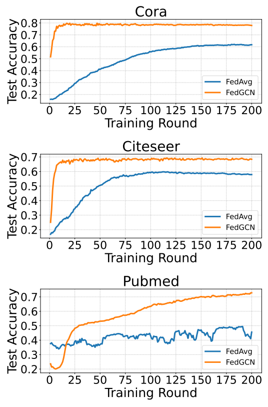
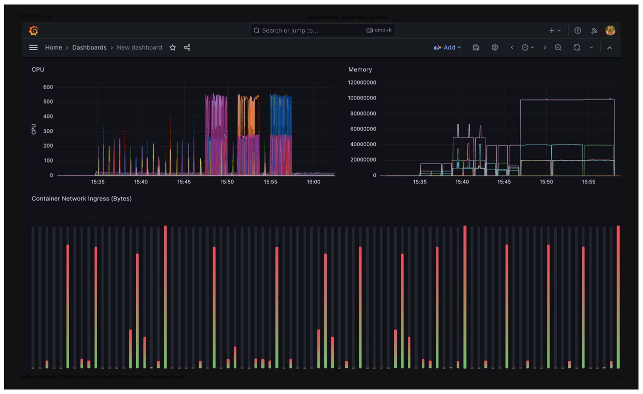


Figure 8: Benchmark results on three FGL tasks (10 clients): accuracy (left), training time (middle), and communication cost (right). (a) Graph classification, (b) Node classification, (c) Link prediction.

5.2 System Performance Monitoring and Resource Utilization

Using Grafana with Prometheus, we monitor 10 training nodes and one server. Figure 9 shows that FedGCN converges faster and achieves higher accuracy than FedAvg. Resource usage scales with dataset size: Cora and Citeseer yield lighter loads, while Pubmed and Ogbn-Arxiv cause higher sustained CPU and memory usage, with spikes aligned to training rounds.





Search or jump to...
Home › Dashboards › New dashboard
Add
CPU
Memory
Container Network Ingress (Bytes)

Figure 9: Test accuracy across training rounds (left: Cora, Citeseer, Pubmed) and Grafana dashboard showing CPU, memory, and network usage (right).

5.3 Real-World Dataset with Realistic Client Data Distribution

We finally evaluate FedGraph at scale on the Ogbn-Papers100M dataset (over 50GB), one of the largest publicly available graph benchmarks. We use Hugging Face for dataset storage and partitioning, with the Trainer class managing local data loading for each client. The data is distributed across 195 clients, with node counts assigned following a power law distribution based on country population sizes, mimicking realistic federated environments where larger clients hold more data.

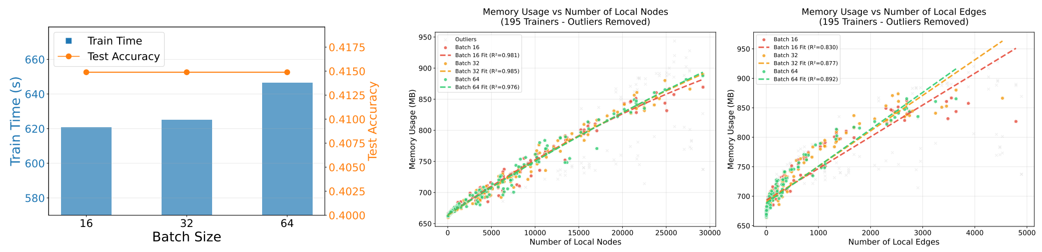


Figure 10: Training Time, Test Accuracy, and Memory Usage for each client on Ogbn-Papers100M under Different Batch Sizes (800 rounds).

As shown in Figure 10, we assess the effect of varying batch sizes (16, 32, and 64) on training time and test accuracy over 800 training rounds. Training time increases modestly with batch size due to additional per-round computation. Test accuracy remains nearly unchanged, with a slight gain from batch size 16 to 32 and a plateau at 64, suggesting flexibility in batch size selection without compromising model performance. Memory usage remains stable at approximately 17.5 GB, demonstrating FedGraph's scalability and efficient resource handling in large-scale settings.

5.4 Benchmarking Scalability under Increasing Clients

We next evaluate the scalability of FedGraph by varying the number of clients in Table 2. As the number of clients increases, the overall communication cost grows substantially, eventually becoming the primary bottleneck. In contrast, the training time per client decreases since each client processes a smaller subgraph.

Table 2: Training and communication time (seconds) for datasets under varying client numbers.

Clients	Cora		CiteSeer		PubMed		OGBN-arXiv	
	Train	Comm	Train	Comm	Train	Comm	Train	Comm
5	1.39	1.69	1.58	2.78	2.08	1.55	127.71	4.48
10	1.36	2.78	1.79	6.55	1.77	2.60	45.82	5.95
15	1.56	3.99	2.40	9.58	1.57	3.84	21.77	7.77
20	1.49	4.87	2.07	13.62	1.83	4.63	17.89	9.24

6 Conclusion

In this paper, we presented FedGraph, a Python library designed for benchmarking federated graph learning algorithms. Unlike general federated learning platforms, FedGraph supports a diverse set of algorithms and enables systematic comparisons across algorithms, datasets, and system configurations. It features fully distributed training, homomorphic encryption for privacy-preserving scenarios, and a built-in system profiler to measure communication and computation overhead. The modular API allows easy integration of custom datasets and algorithms. Through extensive experiments, including low-rank compression and large-scale training on graphs with up to 100 million nodes, we demonstrate that FedGraph is a practical and scalable tool for real-world FGL evaluation.

While this work focuses on enabling privacy-preserving federated graph learning, future efforts are needed to explore more robust privacy risk assessments, additional optimization strategies, and the inclusion of a broader range of FGL algorithms to expand benchmark coverage and better support industrial deployment.

References

Adrien Benamira, Benjamin Devillers, Etienne Lesot, Ayush K Ray, Manal Saadi, and Fragkiskos D Malliaros. 2019. Semi-supervised learning and graph neural networks for fake news detection. In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 568–569.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

Fabrianne Effendi and Anupam Chattopadhyay. 2024. Privacy-preserving graph-based machine learning with fully homomorphic encryption for collaborative anti-money laundering. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 80–105.

Xingbo Fu, Binchi Zhang, Yushun Dong, Chen Chen, and Jundong Li. 2022. Federated graph machine learning: A survey of concepts, techniques, and applications. *ACM SIGKDD Explorations Newsletter* 24, 2 (2022), 32–47.

Zeynep Gürler and Islem Rekik. 2022. Federated Brain Graph Evolution Prediction using Decentralized Connectivity Datasets with Temporally-varying Acquisitions. *IEEE Transactions on Medical Imaging* (2022).

Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Carl Yang, Han Xie, Lichao Sun, Lifang He, Liangwei Yang, Philip S Yu, Yu Rong, et al. 2021. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145* (2021).

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

Nam Hyeon-Woo, Moon Ye-Bin, and Tae-Hyun Oh. 2021. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098* (2021).

Weizhao Jin, Yuhang Yao, Shanshan Han, Carlee Joe-Wong, Srivatsan Ravi, Salman Avestimehr, and Chaoyang He. 2023. FedML-HE: An efficient homomorphic-encryption-based privacy-preserving federated learning system. *arXiv preprint arXiv:2303.10837* (2023).

Mohammad Mahdi Kamani, Yuhang Yao, Hanjia Lyu, Zhongwei Cheng, Lin Chen, Liangju Li, Carlee Joe-Wong, and Jiebo Luo. 2024. WYZE rule: federated rule dataset for rule recommendation benchmarking. *Advances in Neural Information Processing Systems* 36 (2024).

Sungwon Kim, Yoonho Lee, Yunhak Oh, Namkyeong Lee, Sukwon Yun, Junseok Lee, Sein Kim, Carl Yang, and Chanyoung Park. 2025. Subgraph federated learning for local generalization. *arXiv preprint arXiv:2503.03995* (2025).

Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International conference on machine learning*. PMLR, 11814–11827.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems* 2 (2020), 429–450.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189* (2019).

Xunkai Li, Yinlin Zhu, Boyang Pang, Guochen Yan, Yeyu Yan, Zening Li, Zhengyu Wu, Wentao Zhang, Rong-Hua Li, and Guoren Wang. 2024. Openfgl: A comprehensive benchmarks for federated graph learning. *arXiv preprint arXiv:2408.16288* (2024).

Rui Liu, Pengwei Xing, Zichao Deng, Anran Li, Cuntai Guan, and Han Yu. 2024. Federated graph neural networks: Overview, techniques, and challenges. *IEEE transactions on neural networks and learning systems* (2024).

Guannan Lou, Yuze Liu, Tiehua Zhang, and Xi Zheng. 2021. STFL: A temporal-spatial federated learning framework for graph neural networks. *arXiv preprint arXiv:2111.06750* (2021).

Xiang Ni, Xiaolong Xu, Lingjuan Lyu, Changhua Meng, and Weiqiang Wang. 2021. A vertical federated learning framework for graph convolutional network. *arXiv preprint arXiv:2106.11593* (2021).

Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. *Proceedings of Machine Learning and Systems* 4 (2022), 673–693.

Zhen Wang, Weirui Kuang, Yuexiang Xie, Liuyi Yao, Yaliang Li, Bolin Ding, and Jingren Zhou. 2022. Federatedscope-gnn: Towards a unified, comprehensive and efficient package for federated graph learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4110–4120.

Han Xie, Jing Ma, Li Xiong, and Carl Yang. 2021. Federated graph classification over non-iid graphs. *Advances in neural information processing systems* 34 (2021), 18839–18852.

Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudre-Mauroux. 2019. Revisiting user mobility and social relationships in lbsns: a hypergraph embedding approach. In *The world wide web conference*. 2147–2157.

Dingqi Yang, Daqing Zhang, and Bingqing Qu. 2016. Participatory cultural mapping based on collective behavior data in location-based social networks. *ACM Transactions on Intelligent Systems and Technology (TIST)* 7, 3 (2016), 1–23.

Yuhang Yao, Weizhao Jin, Srivatsan Ravi, and Carlee Joe-Wong. 2024. FedGCN: Convergence-Communication Tradeoffs in Federated Training of Graph Convolutional Networks. *Advances in Neural Information Processing Systems* 36 (2024).

Yuhang Yao, Mohammad Mahdi Kamani, Zhongwei Cheng, Lin Chen, Carlee Joe-Wong, and Tianqiang Liu. 2023. FedRule: Federated rule recommendation system with graph neural networks. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*. 197–208.

Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. 2020a. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*. 493–506.

Ke Zhang, Lichao Sun, Bolin Ding, Siu Ming Yiu, and Carl Yang. 2024. Deep efficient private neighbor generation for subgraph federated learning. In *Proceedings of the 2024 SIAM International Conference on Data Mining (SDM)*. SIAM, 806–814.

Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu Ming Yiu. 2021. Subgraph federated learning with missing neighbor generation. *Advances in Neural Information Processing Systems* 34 (2021).

Weijia Zhang, Hao Liu, Yanchi Liu, Jingbo Zhou, and Hui Xiong. 2020b. Semi-supervised hierarchical recurrent graph neural network for city-wide parking availability prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1186–1193.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).

A Frequently Asked Questions

A.1 Client Selection

FedGraph supports two client selection methods that randomly or uniformly select clients at each round, as in server_class.py. Server-side algorithm components can also be added by modifying the server class.

```python
assert 0 < sample_ratio <= 1, "Sample ratio must be between 0 and 1"

num_samples = int(self.num_of_trainers *ample_ratio)

if sampling_type == "random":
    selected_trainers_indices = random.sample(
        range(self.num_of_trainers), num_samples
    )
elif sampling_type == "uniform":
    selected_trainers_indices = [
        (
            i
            + int(self.num_of_trainers * sample_ratio)
            * current_global_epoch
        )
        % self.num_of_trainers
        for i in range(num_samples)
    ]

else:
    raise ValueError("sampling_type must be either 'random' or '
       uniform'")
```

A.2 Easy Integration with New Baselines

We agree that adding more FGL methods to FedGraph could make it more useful to users. Indeed, our main goal is to provide a library for benchmarking the real system performance of federated graph learning methods. Though we believe we have covered most state-of-the-art methods, we acknowledge that some methods from the literature are missing, so we also make the library easy to add new methods for researchers. Researchers can create a new training class based on an existing training method in the library (e.g., FedAvg, FedGCN).

A.3 Communication Overhead after HE with Low Rank

Homomorphic Encryption (HE) has sometimes significant drawbacks, in particular, high computing and communication loads. We include an HE implementation in the FedGraph framework as some prior work on federated graph learning has proposed HE as a way to preserve privacy, e.g., Effendi and Chattopadhyay (2024); Fu et al. (2022); Ni et al. (2021). Thus, researchers in the field may wish to evaluate the effects of HE on various graph learning algorithms, or even evaluate new ways to combine HE with federated graph learning. As a benchmark platform, we do not advocate for or against such ideas; our goal is simply to facilitate the evaluation of federated graph learning algorithms that require HE implementations. For example, researchers may wish to quantitatively compare the training and communication time of their algorithms with and without HE in order to precisely measure the overhead induced by using HE. We will clarify this point in the revised manuscript.

Our work on low-rank HE is meant to serve as an example case study of the types of research that FedGraph enables with HE. We fully agree that it does not reduce the overhead of communication during training. However, FedGraph also facilitates implementing low-rank HE schemes for the aggregation process, such as FedPara Hyeon-Woo et al. (2021).

A.4 Comparison with Other Framework

The FedGraphnn and FederatedScopeGNN libraries have not been maintained since 2023. When running FedAvg, due to using the same graph training library, PyG, and the distributed setup, FedGraph has a similar run time as FedGraphnn and FederatedScopeGNN. As shown in Table 1, the main advantage is supporting new algorithms, homomorphic encryption, system-level profilers, and large-scale optimization like low rank, client selection, mini-batch, etc. Given the optimization methods (e.g., low rank in Figure 7), FedGraph can run much faster than these frameworks.

A.5 Support Differential Privacy and Homomorphic Encryption

FedGraph supports differential privacy(DP) for aggregation as an option in configuration. Our implementation of DP achieves comparable performance to both the plaintext version and HE without having an accuracy loss. Table 3 provides a comparison of different matrix running FedGCN with Cora using plaintext, HE, and DP. Results are averaged over 5 runs. Both HE and DP protect the pre-training or training communication without exposing the raw data at the server in different approaches. This meets our goal of presenting FedGraph to provide user with the flexibility of using and choosing the privacy mechanism that best fits their specific needs.

Framework	Pre-train Communication (MB)	Pre-train Time (s)	Total Time (s)	Accuracy
Plaintext	56.61	4.91	12.08	0.793
HE	1208.87	17.49	40.91	0.791
DP	57.69	5.60	13.09	0.792

Table 3: Comparison of privacy preservation methods in terms of pre-train communication cost, pre-train time, total time, and accuracy.

A.6 Why HE Affects Model Accuracy

HE performance is highly sensitive to hyperparameter settings. For example, in Table 4 and Table 5, we vary the Polynomial Modulus Degree to balance security and computational capacity. A higher degree supports more complex computations but incurs greater overhead, while a lower degree restricts the allowable computation depth before noise dominates, leading to truncated or simplified operations that degrade model accuracy. In some cases, decryption itself becomes inaccurate, with the error magnitude determined by both the chosen hyperparameters and the encrypted values. Since the decrypted outputs are used for graph model training, such errors accumulate and ultimately reduce training accuracy.

A.7 Run on Edge Devices

In this paper, we focus on AWS cloud experiments as many of our envisioned use cases for federated graph learning–including learning on medical record data stored at different hospitals, or user-product consumption data stored in different countries–falls into the cross-silo federated learning paradigm, where clients are likely large servers or computers (hospitals or countries respectively, in the two examples above). FedGraph could also be run on Linux-based mobile or Internet of Things devices, e.g., Jetson Nanos, as might be appropriate for mobile applications like wearable health sensing.

B FedGraph Code Structure

Fedgraph library can be separated into six modules:

Data Process Module: data_process.py. This module is responsible for data generation and processing. It should be called before calling the runner, so it is only applicable for node classification and graph classification in the latest version. (In link prediction, the dataset is generated and processed inside the runner.)

Runner Module: federated_methods.py. Task-based runners are defined in this module. For node classification and link prediction, there's a shared runner that could call inner modules to perform federated graph learning. For graph classification, it will further assign the program to an algorithm-based runner.

Server Classes Module: server_class.py. It defines different task-based server classes including server_NC, server_GC, and server_LP.

Trainer Classes Module: trainer_class.py. It defines different task-based trainer(client) classes including trainer_NC, trainer_GC, and trainer_LP.

Backbone Models Module: gnn_models.py. It defines different backbone model classes. Generally, it is task-based (i.e., each task corresponds to one backbone model), but for some models, their variants are also included, and the user could also switch the backbone model or define a new one themselves.

Utility Functions: utils_nc.py, utils_gc.py, utils_lp.py. In the current version, the utility functions for different tasks are located in separate Python modules, which is convenient for development. In the later versions, it might be better to use the shared 'utils.py', and use different markers like "NC", "GC", and "LP" to distinguish them.

C FedGraph API and Runners

When calling the FedGraph API, as shown in Figure 11, Users can simply specify the name of the dataset and algorithm. The API will then call the corresponding data loader class to generate the required data, which is then automatically fed into the appropriate algorithm runner for tasks like Node Classification, Graph Classification, or Link Prediction. Additionally, users can seamlessly add their own datasets or federated graph learning algorithms if needed, as long as they satisfy the form requirements for the specified task.

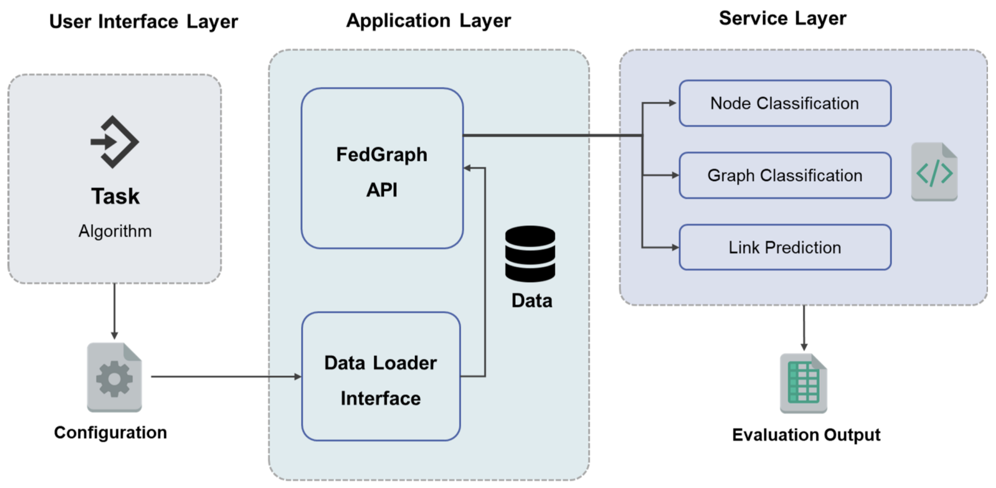


Figure 11: High-level Demonstration of FedGraph API Design

The function run_fedgraph is a general runner that receives the dataset and the configurations. It will further assign the program to a task-specified runner run_NC , run_GC , or run_LP based on the user's specification on Task.

```
def run_fedgraph(args, data):
    if args.fedgraph_task == "NC":
        run_NC(args, data)
    elif args.fedgraph_task == "GC":
        run_GC(args, data)
    elif args.fedgraph_task == "LP":
        run_LP(args)
```

D Supported Algorithms and Datasets

For federated node classification, link prediction, and graph classification, we integrated different datasets and algorithms for each task, shown in Table 4 and Table 5. Researchers can also easily implement new algorithms and add their datasets.

Task	Dataset
Node Classification	Cora, Citeseer, Pubmed, Ogbn-Arxiv, Ogbn-Products, Ogbn-MAG Hu et al. (2020)
Graph Classification	MUTAG, BZR, COX2, DHFR, PTC-MR, AIDS, NCI1, ENZYMES, DD, PROTEINS, COLLAB, IMDB-BINARY, IMDB-MULTI Xie et al. (2021)
Link Prediction	FourSquare Yang et al. (2019), WyzeRule Kamani et al. (2024)

Table 4: Supported datasets of node classification, graph classification, and link prediction in federated learning.

Task	Algorithm	Backbone
Node Classification	FedAvg Li et al. (2019)	GCN
	Distributed GCN	GCN
	BNS-GCN Wan et al. (2022)	GCN
	FedSage+ Zhang et al. (2021)	GraphSage
	FedGCN Yao et al. (2024)	GCN,GraphSage
Graph Classification	SelfTrain	GIN
	FedAvg Li et al. (2019)	GIN
	FedProx Li et al. (2020)	GIN
	GCFL Xie et al. (2021)	GIN
	GCFL+ Xie et al. (2021)	GIN
	GCFL+dWs Xie et al. (2021)	GIN
Link Prediction	FedAvg Li et al. (2019)	GCN
	STFL Lou et al. (2021)	GCN
	FedGNN+ Gürler and Rekik (2022)	GCN
	FedLink	GCN
	FedRule Yao et al. (2023)	GCN

Table 5: Supported algorithms of node classification, graph classification, and link prediction in federated learning.

E Runner Workflow

E.1 Runner Workflow for Node Classification Task

For the node classification task, the user specifies the dataset name in the configuration file. The dataset is preprocessed and partitioned across clients based on the chosen federated setting. The data is accessed directly from local storage or via an API, and then a time window may be generated to support temporal learning tasks.

Dataset extraction and preprocessing are managed by the function `dataloader_NC` in the module `data_process.py`, which handles both temporal and static graph datasets. Each client holds a subgraph or node features, and training proceeds in a federated manner.

The core execution is handled by `run_NC`, which directs the process to an algorithm-specific runner `run_NC_{algorithm}`. Each algorithm may require different configurations, so separate *.yaml* files are used. In each global round, clients perform local training, exchange model updates with the server, and participate in global validation. Results are recorded at the end of each round. The workflow is illustrated in Figure 12a.

E.2 Runner Workflow for Graph Classification Task

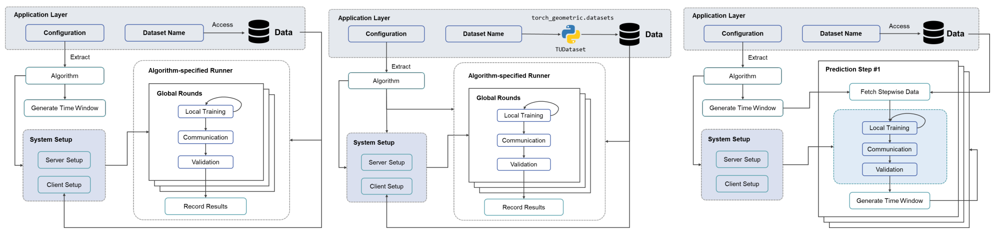
In the graph classification task, the dataset name can be specified by the user, and it could be either a single dataset or multiple datasets. For single dataset GC, the graphs will be assigned to a designated number of clients; for multiple datasets GC, each dataset will correspond to a client. Dataset generation and preparation are controlled by the function `dataloader_GC` in the module `data_process.py`. It will further assign the data process task to the function `data_loader_GC_single` or `data_loader_GC_multiple`. All the provided datasets are built-in TUDatasets in the Python library `torch_geometric`.

For the graph classification task, different algorithms require different sets of arguments. Therefore, we divide the configurations into separate .yaml files, each corresponding to one algorithm. In `run_GC`, the program will be further assigned to an algorithm-based runner `run_GC_{algorithm}`. The whole workflow is demonstrated in Figure 12b.

E.3 Runner Workflow for Link Prediction Task

For the link prediction task, we provide a common dataset. The user only needs to specify the country codes. The original datasets are stored in Google Drive, so the user does not need to prepare the dataset by itself. The API will automatically check whether the dataset already exists and download the corresponding one if not.

There are also multiple available algorithms for the link prediction task. However, they share the same set of arguments so that we don't need to create separate *.yaml* files. The user could conveniently select different algorithms by directly changing the algorithm field in the configuration file. All the algorithms will share the same runner. The whole workflow is demonstrated in Figure 12c.



(a) Node classification. (b) Graph classification. (c) Link prediction.

Figure 12: Federated graph training workflows for (a) node classification, (b) graph classification, and (c) link prediction tasks.

F Benchmark Homomorphic Encryption

We provide a comprehensive guide for the configuration of Homomorphic Encryption in FedGraph to provide security guarantees corresponding to the graph structure and size.

F.1 Parameter Configuration

In Table 6, we present the key parameters for configuring CKKS homomorphic encryption. The selection of the parameters is based on the dataset size and desired security level. Different parameter combinations create tradeoffs between computational overhead, communication cost, and precision. We present the default setting and parameter selection range to guide the user in selecting an appropriate combination that achieves the balance between objectives.

F.2 Microbenchmark

We then provide the microbenchmark of HE on federated graph training in Table 7. The experiments are conducted on a 2-layer FedGCN for node classification tasks, running 100 global rounds with default settings in Cora. For CKKS parameters, we evaluate different polynomial modulus degrees (Poly_mod), coefficient modulus sizes (Coeff_mod), and precision levels. Time(s) for the encrypted

Parameter	Default Value	Description	Range
scheme	CKKS	Encryption scheme type	N/A
polynomial modulus degree	16384	Maximum degree of polynomials used to represent encrypted data ($N \geq 2\times$ max(nodes, features))	4096, 8192, 16384, 32768
coefficient modulus bit size	[60, 40, 40, 40, 60]	Bit size for coefficient modulus that controls precision	Array of integers in the range $[20, 60]$
global scale	2^{40}	Global scale factor for encoding precision	2^{30}, 2^{40}, 2^{50}, etc.
security level	128	Bit security level	128, 192, 256

Table 6: TenSEAL Homomorphic Encryption Configuration Parameters. This table shows the key parameters for configuring the CKKS encryption scheme in the FedGraph library, including their default values, descriptions, and available ranges.

version show pre-train/training/total times, respectively. Communication costs (Comm_cost) include both pre-training and training rounds.

Dynamic Precision: We adjust encryption parameters based on graph sizes and the numerical precision needed. For graphs like Cora, a polynomial modulus degree of 16384 with precision 2^{40} satisfies the ideal accuracy, while an increased value provides more precise security protection.
Communication Cost Optimization: We employ several strategies to manage the communication overhead inherent in HE operation. The selection of coefficient modulus chain, [60,40,40,40,60], etc., enables efficient depth management for multiple HE operations. Depending on specific dataset characteristics (sparse matrix, larger datasets, etc.), we also employ efficient encryption methods to optimize communication cost and balance the performance.

When comparing among datasets, we observe that HE maintains equivalent accuracy across different parameter selections, as long as they satisfy the modulus requirement. If a smaller-than-required parameter size is used, the accuracy drops sharply, which indicates invalid encryption.

Method	Poly _mod	Coeff_mod	Precision	Dataset	Time(s)	Comm_cost (MB)	Accuracy
FedGCN (plaintext)	N/A	N/A	N/A	Cora	13.29	59.21	0.783 ± 0.07
FedGCN (HE)	16384	[60,40,40,40,60]	2^{40}	Cora	27.71/23.22/56.05	3279.15	0.779 ± 0.08
FedGCN (HE)	32768	[60,40,40,40,60]	2^{50}	Cora	29.44/36.17/71.76	4434.58	0.781 ± 0.08
FedGCN (plaintext)	N/A	N/A	N/A	Citeseer	20.39	187.99	0.658 ± 0.06
FedGCN (HE)	8192	[60,40,40,60]	2^{40}	Citeseer	79.35/30.63/113.08	5791.42	0.660 ± 0.07
FedGCN (HE)	16384	[60,40,40,40,60]	2^{40}	Citeseer	123.22/45.8/173.40	8084.50	0.652 ± 0.06
FedGCN (plaintext)	N/A	N/A	N/A	PubMed	25.78	150.43	0.774 ± 0.12
FedGCN (HE)	8192	[60,40,40,60]	2^{40}	PubMed	70.28/19.73/93.18	3612.60	0.757 ± 0.19
FedGCN (HE)	16384	[60,40,40,40,60]	2^{40}	PubMed	123.22/45.8/173.40	8084.50	0.769 ± 0.13

Table 7: Microbenchmark of FedGCN under Homomorphic Encryption with different CKKS scheme parameters. The experiments are conducted on a 2-layer FedGCN for node classification tasks, running 100 global rounds with default settings across three datasets (Cora, Citeseer, PubMed). For CKKS parameters, we evaluate different polynomial modulus degree (Poly_mod), coefficient modulus sizes (Coeff_mod), and precision levels. Time(s) for the encrypted version show pre-train/training/total times, respectively. Communication costs (Comm_cost) include both pre-training and training rounds. Plain-text FedGCN serves as the baseline for comparison. Communication costs are measured for pre-training communication and training rounds separately.

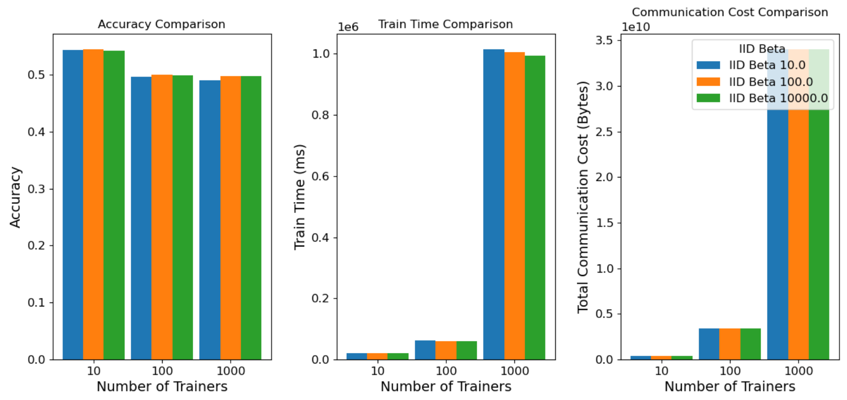


Figure 13: Training Time, Communication Cost, Test Accuracy on Ogbn-Arxiv in a Large Number of Clients. All experiments run on 10 AWS instances. 1000 trainers take a long time since it runs sequentially on 10 instances.

G Additional Experiments

G.1 Increasing the Number of Clients with Fixed Computation Resources

To better test scalability and fit real-world data, we increase the number of clients to 1000. In Figure 13, we observe that as the number of clients increases, the overall training time grows significantly due to sequential running on 10 instances, added communication overhead, and the need for increased synchronization among clients.

As we scale from 10 to 1000 clients under a fixed IID Beta value, there is a small decline in accuracy, likely due to the increased data heterogeneity each client possesses. The communication cost also escalates notably with more clients, highlighting the trade-off between parallelism and efficiency in federated settings. This experiment shows the system's ability to handle large-scale client distributions while revealing the resources required to maintain accuracy and efficiency.