LLMGraph: Automated Graph Neural Networks Designing with LLMs through a Knowledge-Driven Approach

Xiaohan Zheng[1] , **Lanning Wei**[1] , **Yong Li**[1] , **Quanming Yao**[1]
[1]Department of Electronic Engineering, Tsinghua University
zhengxh23@mails.tsinghua.edu.cn, weilanning@163.com, liyong07@tsinghua.edu.cn, qyaoaa@tsinghua.edu.cn

Abstract

Graph Neural Networks (GNNs) are widely used in learning from graph-structured data, but they take efforts to configure and tune. In this demo, we propose LLMGraph, showing how to make GNN automated through Large Language Models. Our system develops a set of agents that construct graph-related knowlege bases and then adopt Retrieve-Agmented Generation to empower the GNN configuration and tuning procedures. These agents, equipped with specialized knowledge bases, extract insights into tasks and graph structures by interacting with the knowledge bases. Empirical results show LLMGraph excels in twelve datasets across three graph learning tasks, validating its effectiveness of GNN model designing.

1 Introduction and Related Work

Graph Neural Networks (GNNs) are extensively applied to learn from graph-structured data across diverse domains, including tasks such as anomaly detection and recommendation systems in social networks [Hamilton *et al.*, 2017], as well as for predicting biomedical molecular properties [Gilmer *et al.*, 2017]. The majority of existing GNNs are designed for diverse graphs under a specific task [Wu *et al.*, 2020], such as capturing graph-level representations [Zhang *et al.*, 2018; Ying *et al.*, 2018]. and learning subgraph patterns in link-level tasks [He *et al.*, 2020; Zhang and Chen, 2018]. However, designing effective GNNs for different graph learning problems is challenging, as it requires substantial knowledge in order to understand the tasks and graphs [Hoffman *et al.*, 1995]. Then, there is a natural question: *How to integrate graph learning knowledge to design effective GNNs?* It is non-trivial to answer this question. Firstly, existing methods have not provided explicit guidelines for utilizing knowledge in designing GNN model architectures. Most GNNs are designed to effectively model graphs for a specific task [Wu *et al.*, 2020; Hamilton *et al.*, 2017; Ying *et al.*, 2018], based on implicit human expertise, which is difficult to explicitly describe and extract.

Therefore, we propose LLMGraph, which automates GNN design using LLMs. Specifically, we have designed a Knowledge Agent to extract graph-related knowledge, building knowledge bases covering advanced graph learning research. Then, we have developed a set of agents that use RAG to interact with knowledge bases, designing GNNs step by step in a knowledge-driven manner. Leveraging LLMs' task analysis, LLMGraph streamlines the designing and refinement of GNN model architectures. Extensive experiments on twelve datasets across three tasks demonstrate LLMGraph's superior performance and efficiency, proving the effectiveness of integrating knowledge for automated GNN design.

2 Method

We introduce LLMGraph, which prepares and utilizes knowledge to design GNN model architectures for diverse graph learning tasks using LLM-based agents. Firstly, we gather graph-related resources and develop a knowledge agent for knowledge extraction and retrieval. Subsequently, the knowledge is then used by several LLM-based agents step by step to design effective GNN model architectures.

2.1 Knowledge Bases Construction and Utilization

Knowledge Bases Construction LLMs face challenges due to outdated knowledge and hallucinations. We address this by creating two knowledge bases, which is currently lacking for designing GNN model architectures. We collect resources and use the Knowledge Agent to manage them.

The Knowledge Agent is tasked with acquiring and integrating specialized knowledge tailored to specific user requirements. This agent mainly manages two types of knowledge bases, as shown in Figure 1: the prior knowledge base and the experiment knowledge base. The prior knowledge base is enriched with task-specific information extracted from sources such as the Open Graph Benchmark (OGB) leaderboards, the PyTorch Geometric (PyG) documentation, and the top-tier conference proceedings that are accessible on Arxiv, ensuring the agent remains at the cutting edge of technology and methodology. The experiment knowledge base archives detailed experimental outcomes such as the benchmark evaluation results, including models setups and their performance on specific datasets, thereby providing insights into their effectiveness and application contexts.

The content of papers and reports often overlaps, with redundant background information and methods that can introduce noise and reduce the informativeness of retrieved knowledge. To address this, we employ a two-level knowledge

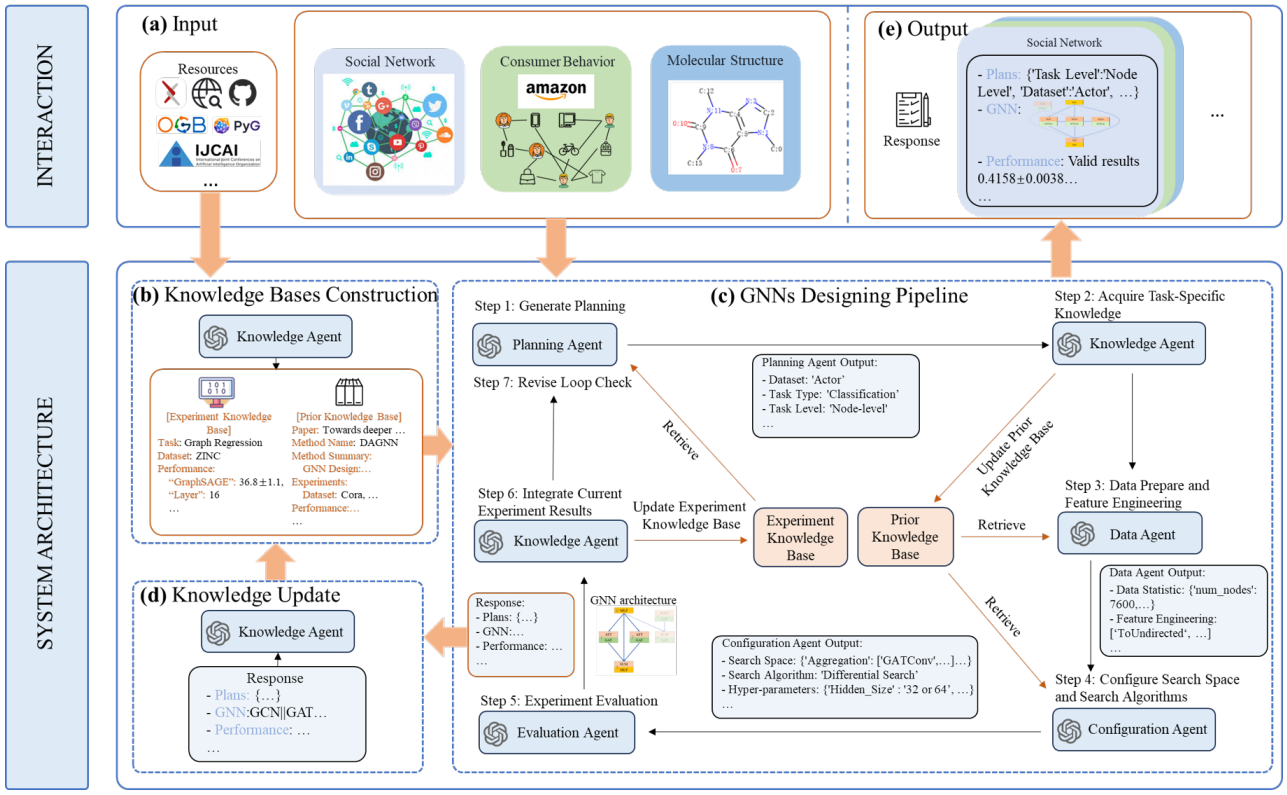

Figure 1: System architecture of LLMGraph. LLMGraph is designed to automate the design of GNN model architectures through a knowledge-driven approach. First, as shown in (b) of the figure,LLMGraph will construct knowledge bases through the Knowledge Agent. These knowledge bases will then be fed into the design pipeline for automated GNN design and finetuning, as shown in (c) of the figure. Finally, as shown in (d) of the figure, the system's response will be returned to the user and updated in the experimental knowledge base.

extraction strategy, first, we start by summarizing inputs to obtain coarse-grained knowledge, then refine this into fine-grained details specific to graph learning tasks, such as architecture design and dataset usage. More details on resource handling and extraction methods are provided in the appendix of extended version [1], please note the link to the repository.

Knowledge Utilization and Update To effectively utilize the constructed knowledge bases, we implement a goal-aware knowledge retrieval mechanism. Utilizing the Retrieve-Agmented Generation (RAG) technique, we enhance the the effectiveness of the designing GNN model architectures by retrieving relevant knowledge. The pre-trained model all-MiniLM-L6-v2 is employed to encode both the extracted knowledge and the queries from other agents. We calculate the cosine similarity in the embedding space to identify the most relevant knowledge. To accommodate the varying goals and resource types in graph learning, we apply a post-ranking strategy. The top-k knowledge items from each resource type are initially retrieved and then re-ranked and selected by the knowledge agent based on the query's context. This refined knowledge is integrated into the graph learning agent's prompt, facilitating the GNN model's design.

LLMGraph also incorporates a dynamically knowledge update mechanism. After the evaluation of a GNN model, the experimental summary, including the task plan, designed GNNs, and results, is stored in memory. The planning agent then compiles a report, which is added to the knowledge base, ensuring that the system's knowledge remains current and applicable for future pipeline runs. This continuous update process allows LLMGraph to adapt and improve over time, enhancing its ability to design effective GNN models.

2.2 Knowledge-Driven GNNs Model Designing

Figure 1 illustrates how each agent engages with knowledge bases to streamline the entire process. The two knowledge bases bridge research and application, they empower agents to make informed decisions.

Planning Agent The Planning Agent generate a task plan based on user instructions, to direct subsequent agent actions, which includes specifications for datasets, task types and evaluation metrics. After all agents completed their tasks, this agent evaluates the experimental results, utilizing insights from the experiment knowledge base to determine whether a revision loop is necessary.

Data Agent The Data Agent utilizes insights from the prior knowledge base to perform feature engineering tailored to specific graphs and tasks, ensuring alignment with expert practices in a knowledge-driven manner.

Configuration Agent The Configuration Agent is responsible for configuring the search space, which includes possible model architecture configurations such as layers and connections, and the search algorithm that explores this space. It

[1] https://github.com/lgssstsp/LLMGraph

Table 1: Performance comparisons of the proposed LLMGraph and baselines on three tasks. We report the test accuracy and the standard deviation for node and graph classification tasks, and use the common Rooted Mean Square Error (RMSE) for the item ranking task. The top-ranked performance in each dataset is highlighted in gray, and the second best one is underlined. The average rank on all datasets is provided in the last column.

	Cora	Photo	Actor	Genius	obgn-arxiv	DD	Proteins	ogbg-molhiv	Amazon-Sports(↓)	Avg. Rank
LLMGraph	87.10(0.36)	96.11(0.33)	40.93(0.35)	90.89(0.11)	72.70(0.54)	78.27(2.57)	75.44(0.93)	74.27(1.54)	0.9298(0.0071)	1
LLMGraph (GL)	86.68(0.40)	95.50(0.21)	39.59(0.39)	90.33(0.15)	72.30(0.54)	77.69(2.24)	74.88(1.16)	73.37(1.23)	0.9622(0.0103)	2.5
GCN	85.68(0.61)	93.13(0.27)	33.98(0.76)	89.10(0.13)	71.74(0.29)	73.59(4.17)	74.84(3.07)	73.89(1.46)	1.0832(0.0077)	5.25
SAGE	86.18(0.35)	94.60(0.25)	39.28(0.18)	89.71(0.09)	71.49(0.27)	76.99(2.74)	73.87(2.42)	73.46(1.69)	0.9900(0.0125)	4.5
AutoML	86.57(0.32)	95.38(0.30)	40.39(0.03)	90.81(0.04)	72.42(0.37)	77.03(2.48)	74.58(2.61)	73.51(3.21)	0.9327(0.0006)	2.63
LLM-GNN	84.64(1.04)	93.73(0.38)	38.92(0.07)	89.31(0.17)	70.83(0.93)	75.12(3.44)	74.47(3.65)	72.93(0.90)	0.9670(0.0079)	5.13

interacts with the prior knowledge base to gain insights on model design, enhancing the effectiveness of search space configuration and algorithm selection.

Evaluation Agent The Evaluation Agent is designed to fine-tune the designed GNN and conduct experiments to validate its performance. After completing the experiments, the Evaluation Agent transmits the results to the Knowledge Agent for integration into the experiment knowledge base.

3 Experiments

We evaluate LLMGraph's effectiveness on twelve datasets across three tasks as shown in Table 1, the performance of another three datasets are shown in appendix of extended version. Detailed resource costs and ablation studies are in the appendix of the extended version.

3.1 Experimental settings

Datasets We evaluate twelve widely used datasets across three tasks as shown in Table 1. The detailed introduction of these datasets and the evaluation performance of another three datasets are shown in appendix of extended version.

Baselines In this paper, we provide several kinds of baselines. (1) GNNs with task adaption, including GCN [Kipf and Welling, 2016] and GraphSAGE [Hamilton *et al.*, 2017] with task-specific adaptations. (2) AutoML-based methods. We adopt F2GNN [Wei *et al.*, 2022] / LRGNN [Wei *et al.*, 2023] / Prof-CF [Wang *et al.*, 2022a] for three tasks. (3) LLM-GNN. GNNs generated by LLMs. (4) LLMGraph (GL) operates without external knowledge.

3.2 Performance Comparisons

Table 1 showcases the performance of LLMGraph against baselines on twelve datasets across three tasks. LLMGraph consistently outperforms all baselines, highlighting its ability to design effective GNNs for various graph learning tasks. The enhanced performance of LLMGraph over LLMGraph (GL) underscores the value of incorporating extracted knowledge into the GNN design process. Unlike AutoML methods that operate within a predefined design space, LLMGraph (GL) leverages LLMs to expand this space, achieving comparable performance and validating the agents' problem-solving capabilities. The LLM-GNN baseline, which relies solely on LLM suggestions without knowledge integration, faces challenges in understanding tasks and graphs, resulting in less effective GNN designs. LLMGraph's superior performance highlights the significance of knowledge in designing effective GNNs for graph learning tasks.

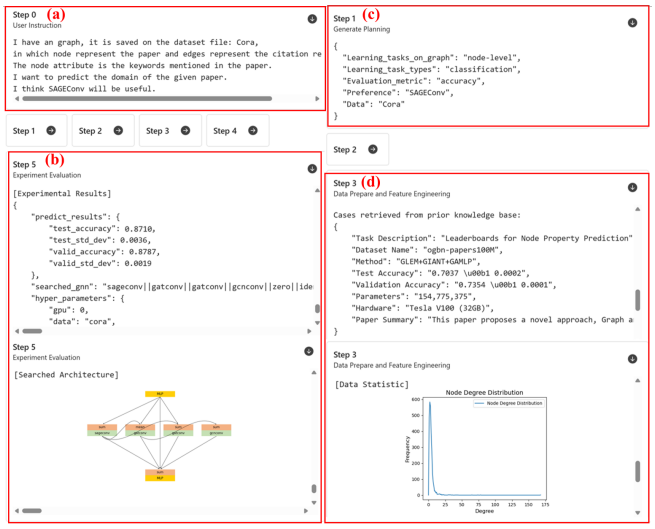

Figure 2: This figure demonstrates the detailed steps and output of LLMGraph.

4 Demonstration

In this section, we demonstrate the use case of LLMGraph on a real-world problem. For example, users aim to predict the category of articles within a citation network.

As shown in Figure 2, **(a)** illustrates the user's input instructions, **(b)** displays the system's experimental results and its designed GNN model, LLMGraph achieves an accuracy of 0.8710 on the Cora dataset, surpassing the GNN-based baselines GCN [Kipf and Welling, 2016] at 0.8568, ACM-GCN [Luan *et al.*, 2022] at 0.8667 (Detailed experiments is in the extended version), and the AutoML-based baseline SANE [Zhao *et al.*, 2021] at 0.8640. **(c)** displays the task plan generated by the Planning Agent, which interprets the user's intention to predict the category of articles within a citation network as a node classification task. **(d)** shows the Data Agent retrieve relevant knowledge from the prior knowledge base. Specifically, the Data Agent extracts methods related to node classification tasks. It also visualizes the statistical characteristics of the graph data, allowing the data agent to gain a deeper understanding of its structure.

This demonstration showcases the effectiveness of LLMGraph in automatically designing GNN model for real-world graph learning problem.

References

[Bergstra *et al.*, 2013] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

[Chien *et al.*, 2021] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *ICLR*, 2021.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272, 2017.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[He and McAuley, 2016] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

[He *et al.*, 2017] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[He *et al.*, 2020] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[Hoffman *et al.*, 1995] Robert R Hoffman, Nigel R Shadbolt, A Mike Burton, and Gary Klein. Eliciting knowledge from experts: A methodological analysis. *Organizational behavior and human decision processes*, 62(2):129–158, 1995.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2016.

[Lim *et al.*, 2021] Derek Lim, Xiuyu Li, Felix Hohne, and Ser-Nam Lim. New benchmarks for learning on non-homophilous graphs. *arXiv preprint arXiv:2104.01404*, 2021.

[Luan *et al.*, 2022] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. In *NeurIPS*, 2022.

[Sen *et al.*, 2008] 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.

[Shchur *et al.*, 2018] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[Tang *et al.*, 2009] 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 data mining*, pages 807–816, 2009.

[Wang *et al.*, 2019] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[Wang *et al.*, 2022a] Xin Wang, Ziwei Zhang, and Wenwu Zhu. Automated graph machine learning: Approaches, libraries and directions. *arXiv preprint arXiv:2201.01288*, 2022.

[Wang *et al.*, 2022b] Zhenyi Wang, Huan Zhao, and Chuan Shi. Profiling the design space for graph neural networks based collaborative filtering. In *WSDM*, pages 1109–1119, 2022.

[Wei *et al.*, 2022] Lanning Wei, Huan Zhao, and Zhiqiang He. Designing the topology of graph neural networks: A novel feature fusion perspective. In *The WebConf*, pages 1381–1391, 2022.

[Wei *et al.*, 2023] Lanning Wei, Zhiqiang He, Huan Zhao, and Quanming Yao. Search to capture long-range dependency with stacking gnns for graph classification. In *Proceedings of the ACM Web Conference 2023*, pages 588–598, 2023.

[Wu *et al.*, 2020] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2020.

[Ying *et al.*, 2018] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, pages 4800–4810, 2018.

[Zhang and Chen, 2018] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[Zhang *et al.*, 2018] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[Zhao *et al.*, 2021] Huan Zhao, Quanming Yao, and Weiwei Tu. Search to aggregate neighborhood for graph neural network. In *ICDE*, 2021.

A Detailed Introduction of Agents in LLMGraph

In this section, we provide the implementation details of each agent within the LLMGraph and how they collaborate to achieve the automated process of generating graph learning models from user instructions.

A.1 Instructions

As shown in Table 2, we provide five keys when constructing the instructions, which contain the descriptions of data, learning task, evaluation metric and user preferences. It should be noted that more complex instructions can be constructed by introducing error information or more personalized keys, which is out-of-scope of this paper.

Table 2: The keywords in user instruction.

Keywords	Content
Data $\mathcal{D}$	The introduction of graph or the used dataset.
Task $\mathcal{T}$	The learning target on the graph in natural language, e.g., recommending items to particular users or predicting the properties of specific molecular graphs.,
Evaluation metric $\mathcal{M}$	The evaluations of the learning target. In general, it would be the test performance or the response time.
Preference $\mathcal{P}$	The prior knowledge of users in setting up the graph learning procedures, either on the feature engineering, architecture design or hyper-parameter selections.,
Constrains $\mathcal{C}$	The constrains on model, device, or training strategy.

A.2 Knowledge Agent

The Knowledge Agent is tasked with acquiring and integrating specialized knowledge tailored to specific user requirements and agent functions. The specific prompt and function design is illustrated in Table 6. The agent operates in phases: acquiring knowledge from diverse sources, processing through vector-based semantic retrieval to prioritize knowledge, and integration of insights into the graph learning process. Those operations ensure the agent accesses relevant information and transforms it into insights for the design of graph learning models. To support these operations, the Knowledge Agent is equipped with tools like search engines and natural language processing models. This allows it to acquire advanced information and handle large volumes of data efficiently necessary for knowledge extraction and synthesis.

A.3 Planning Agent

The planning agent serves as the starting point of our proposed LLMGraph and is responsible for parsing the user's natural language instructions to extract key information about the graph learning task. To achieve this goal, we have adopted a structured prompt template named PromptTemplate. The design of this template aims to guide LLMs in analyzing user requests and generating responses in a predefined format. The specific prompt and function design is illustrated in Table 3. The expected LLM output and its content are detailed in Table 2. Based on the above prompt, the planning agent will invoke LLM to extract relevant information from the user request and store the information in key-value pairs for use by subsequent agents.

A.4 Data Agent

The Data Agent plays a crucial role to identify and select the most appropriate data pre-processing methods and feature engineering techniques in line with the specific requirements of the task. To better select feature engineering techniques from the PyTorch Geometric (PyG) document that are more suitable for the current task and the statistical characteristics of data, the Data Agent interacts with the Knowledge Agent. This interaction involves conveying its requirements to the Knowledge Agent, after which the Data Agent makes decisions based on the knowledge returned by the Knowledge Agent. This step ensures that the data is provided in the most suitable form for model learning and the prompt and function design is shown in Table 4.

A.5 Configuration Agent

In the Configuration Agent, we first select suitable modules and operations from the PyTorch Geometric (PyG) document based on the knowledge retrieved from knowledge base to configure the search space. Then, using the configured search space along with the formatted instructions, we choose the appropriate search algorithm. The prompt and function design is shown in Table 5. The configured search space and search algorithm are also utilized to enhance the knowledge base by documenting the performance outcomes associated with these configurations. This process not only demonstrates current efficiencies but also enriches the knowledge base, thereby supporting the continuous improvement and advancement of the LLMGraph framework towards more effective solutions.

A.6 Evaluation Agent

The Evaluation Agent is designed to conduct experiments to validate the proposed solutions. The prompt and function design of the Evaluation Agent is shown in Table 7. Initially, it employs selected search algorithms to navigate through the defined search space. It executes the code, tailored to the search space and algorithms, to facilitate Neural Architecture Search (NAS), thereby identifying optimal models. Subsequently, it refines the hyperparameters to derive the final experimental outcomes. In this process, we utilize HyperOPT [Bergstra *et al.*, 2013] to enhance the fine-tuning phase. Finally, the experiment agent constructs a comprehensive experimental result, informed by the experimental data.

This experimental result encompasses the searched models, the hyper-parameters, the prediction performance and the resource consumption as detailed in Fig. 1.

Table 3: The prompt and function design of the Planning Agent.

The Prompt and Function Design of Planning Agent
Profile You are a Graph Learning Specialist with exceptional abilities in interpreting complex user instructions and translating them into structured, actionable data formats. Furthermore, once the evaluation agent has completed the assessment, the planning agent will determine whether to restart the experiments, depending on the outcomes of the evaluation agent and the experimental results from the knowledge base. # Objective Your task is to deconstruct user instructions related to graph learning into detailed, executable plans, ensuring all elements of the request are accurately captured and clearly categorized. # Functions 1. **Task Plan Function** - **Purpose**: To meticulously parse and interpret user instructions, identifying key information pertinent to graph learning tasks. - **Input**: User instructions. - **Output**: A Python Dictionary of the task highlighting its level (node, link, or graph), type of learning task (classification, regression, etc.), the name of dataset, and evaluation metrics. 2. **Revision Plan Function** - **Input**: The complete output of the current experiment, previous results for comparison and the experimental results from the knowledge base. - **Output**: A Python Dictionary of the revision plan, which contains the reason for the revision, direction of revision and the expected performance. # Human Expertise Human expertise in this context involves a deep understanding of how to interpret complex user requests related to graph learning. The expertise includes: 1. **Task Categorization**: Identifying the level of graph learning tasks (node, link, or graph-level) based on the user's description. 2. **Task Type Determination**: Distinguishing between different types of learning tasks, such as classification or regression, based on the details provided in the user's request. 3. **Evaluation Metric Selection**: Selecting appropriate evaluation metrics. 4. **Preference Identification**: Noting any specific operational preferences mentioned by the user. 5. **Dataset Identification**: Accurately identifying the relevant datasets mentioned in the user's request.

Table 4: The prompt and function design of the Data Agent.

The Prompt and Function Design of Data Agent.
Profile You are a Graph Learning Specialist with specialized skills in navigating and utilizing structured knowledge for optimizing machine learning workflows. Your expertise includes leveraging external knowledge sources and extracting and interpreting complex data from document formats to assist in feature engineering. # Objective Your task is to analyze the PyG documentation and user requests to identify and select appropriate feature engineering techniques that are most effective for the specified task plan. This involves extracting relevant techniques from the documentation that directly align with the user's objectives and requirements, thereby enhancing the model's performance. # Functions 1. **Feature Engineering Selection Function** - **Purpose**: To determine the best feature engineering techniques from the provided documentation that align with the user's request, the task requirements and the statistical characteristics of data. - **Input**: User request $'user\_req'$, task plan details $'task\_plan'$, specific documentation content $'content'$, and the statistical characteristics of data $'graph\_structural\_analysis'$. - **Output**: A list of up to three selected feature engineering techniques, formatted as a Python Dictionary ensuring they are directly applicable and beneficial for enhancing the model's performance. # Human Expertise Human experts are crucial in the process of selecting effective feature engineering techniques from PyG documentation. Their expertise involves analyzing the task requirements, understanding the types of datasets involved, and comprehending detailed descriptions of feature engineering. This knowledge enables them to choose the most relevant and beneficial feature engineering functions that align with the specific needs of the task and enhance the overall performance of the model. These decisions are based on a deep understanding of how different techniques can affect the efficiency and effectiveness of graph neural network models in various contexts.

Table 5: The prompt and function design of the Configuration Agent.

The Prompt and Function Design of Configuration Agent
Profile You are a Graph Learning Specialist specialized in navigating and utilizing configuration options based structured knowledge. Your expertise allows you to effectively absorb relevant knowledge, parse documentation, extract operational data, and apply those information to configure search space and search algorithm. # Objective Your primary objective is to orchestrate the configuration process of graph neural networks by selecting appropriate modules, preparing operation candidates, evaluating these candidates, and finally selecting an optimal configuration that enhances the model's effectiveness and efficiency. # Functions 1. **Module Selection Function**: Aimed at identifying the best modules for inclusion in the graph neural network based on the task's specifics. - **Input**: Task requirements and available module options. - **Output**: List of modules deemed most suitable for the task. 2. **Operation Preparation Function**: Prepares the detailed list of operations from specific documentation content $'content'$ that can be performed by the selected modules. - **Input**: Selected modules and specific documentation content. - **Output**: Detailed operations capable of being executed by these modules. 3. **Candidate Selection Function**: Evaluates the prepared operations and selects the most promising candidates for final deployment. - **Input**: List of prepared operations. - **Output**: Shortlist of candidate operations for search space. 4. **Construct Search Space Function**: Constructs a comprehensive search space where different configurations can be tested and evaluated. - **Input**: Candidate operations. - **Output**: A structured search space. 5. **Algorithm Selection Function**: Selects the most suitable algorithm through based on the identified requirements. - **Input**: The constructed search space, the task plan and the selected modules. - **Output**: The most effective search algorithm for finding the network architecture. # Human Expertise The configuration of graph neural networks involves a sequential process. Initially, human experts select modules that align with the specific demands of the task. Following this, they outline potential operations for these modules and narrow down the choices to the most effective ones for candidate selection. The next step is constructing a search space based on the selected modules and selected candidates. Finally, human experts select an algorithm that best navigates this space to find the optimal architecture of the network. Throughout this process, human expertise ensures that each step is tailored to meet the task-specific goals and technical requirements efficiently.

Table 6: The prompt and function design of the Knowledge Agent.

The Prompt Design of Knowledge Agent
Profile As a Graph Learning Specialist, you are tasked with extracting and retrieving knowledge to enhance graph learning frameworks using advanced capabilities like RAG techniques. You are equipped with tools to read various file types and summarize relevant information to support GNN design based on specific learning requirements. # Objective Your primary task is to identify knowledge requirements, extract pertinent information from diverse resources, and retrieve specific knowledge to aid different graph learning procedures. # Functions 1. **Knowledge Extraction** - **Purpose**: To gather and classify knowledge from multiple resources to provide both coarse-grained and fine-grained insights into graph learning. - **Input**: Diverse resources including academic papers, technical reports, and benchmark results. - **Output**: Summarized knowledge in structured formats, saved on hardware devices for later retrieval. 2. **Knowledge Retrieval** - **Purpose**: To dynamically retrieve relevant knowledge based on specific requirements of graph learning agents using encoded embedding vectors. - **Input**: Encoded queries reflecting the needs of various graph learning agents. - **Output**: Retrieved knowledge that is semantically aligned with the input requirements, enhancing the graph learning processes. 3. **Query Construction** - **Purpose**: To construct effective retrieval queries by integrating specific keywords and agent requirements to align with learning objectives. - **Input**: Agent-specific information and keywords based on the graph learning tasks. - **Output**: Constructed queries that guide the retrieval of relevant knowledge. # Human Expertise Human experts critically assess the relevance and accuracy of extracted knowledge, fine-tune retrieval queries, and ensure the effective use of extracted and retrieved knowledge in enhancing understanding of tasks and graph structures.

Table 7: The prompt and function design of the Evaluation Agent.

The Prompt and Function Design of Evaluation Agent
Profile You are a Graph Learning Specialist skilled in automating neural architecture search (NAS) and tuning for graph neural networks. Your capabilities extend to managing entire lifecycle processes from architecture search, fine-tuning, to summarizing outcomes into actionable insights. # Objective Your primary task is to automate the search and tuning of neural architectures, streamline execution processes, and generate detailed, structured summaries of the outcomes. # Functions 1. **Experimental Validation** - **Purpose**: Integrates and automates the complete process of neural architecture search and fine-tuning. - **Input**: User-defined parameters including search space, feature engineering functions, GPU preferences. - **Output**: Execution scripts, search logs, and tuning logs from both NAS and fine-tuning processes. 2. **Summary Generation** - **Purpose**: Analyzes logs and data from both the NAS and fine-tuning processes to synthesize comprehensive summaries that incorporate prediction outcomes, architecture details, optimized hyperparameters, and resource usage. - **Input**: Logs and outputs generated during the NAS and fine-tuning phases. - **Output**: Python dictionary with detailed summaries highlighting key results, strategic insights, and areas for optimization. # Human Expertise Human experts play a crucial role in setting up and configuring the execution codes to align with the selected feature engineering functions and the search space. Human experts also critically analyze the results to ensure detailed and actionable summaries are generated, thus providing a deeper understanding of the effectiveness of the tested network architectures.

B Experiments

B.1 Datasets

In the node classification task, Cora [Sen *et al.*, 2008] is the citation network where each node represents a paper, and each edge represents the citation relation between two papers; Computers and Photo [Shchur *et al.*, 2018] are the Amazon co-purchase graphs where nodes represent goods that are linked by an edge if these goods are frequently bought together; Physics [Shchur *et al.*, 2018] is a co-authorship graph where nodes are authors who are connected by an edge if they co-author a paper. Actor [Tang *et al.*, 2009] co-occurrence network is the actor-only induced subgraph of the film-directoractor-writer network where nodes correspond to an actor, and the edge between two nodes denotes co-occurrence on the same Wikipedia page. The statistics of these datasets are provided in Table 10.

In the graph classification task, NCI1 and NCI109 are datasets of chemical compounds; DD and PROTEINS are datasets both protein graphs. Ogbg-molhiv is the dataset of molecular graphs. The statistics of these datasets are provided in Table 8.

In the item ranking task, two datasets are adopted. Epinions is a graph of consumer reviews, in which node represent the users and the edges represent trust relationship between users. Amazon-Sports is a e-commerce dataset which contains the subset of sports category from [He and McAuley, 2016]. The statistics of these datasets are provided in Table 9.

B.2 Baselines

We use three types of baselines in this paper to evaluate the versatility of LLMGraph, i.e., the human-designed widely used baselines and SOTA methods used in recent two years; the AutoML-based method; and the baselines that suggested by LLMs directly. In the following, we introduce the baselines used in each task, and then analyze the construction of LLM-GNN methods in each dataset.

Node-level. We adopt the (1) human-designed method. GCN, GIN and SAGE baseline; GPR-GNN [Chien *et al.*, 2021] that learns the weights of each layer based on generalized PageRank, and ACM-GCN [Luan *et al.*, 2022] that adaptive mixing the channel information from low-/high-/full-frequency, and the configuration of these two methods are followed the original paper. (2) NAS-based method SANE that learns the connections based on JKNet and F2GNN that designs the network topology from the feature fusion perspective. We employ the official code and then search on the target dataset.

Graph-level. We adopt the (1) the human-designed global pooling method. GCN, GIN and SAGE baseline with global add readout operations; DGCNNthat learn the graph representation based on the selected top-ranked nodes; and the hierarchical pooling methods DiffPool [Ying *et al.*, 2018] to learn the hierarchy nature in the graphs; (2) NAS-based method. LRGNNthat design the network topology to capture the long-range dependency. The experiments are conducted based on the official code provided by this paper.

Link-level. In the item ranking task, we adopt (1) the human-designed method. NCF [He *et al.*, 2017] leverages neural networks to learn a more expressive interaction functions; NGCF [Wang *et al.*, 2019] and LightGCN [He *et al.*, 2020] are the representative GNN-based collaborative filtering methods, where the former adopts hadamard product and the latter uses identity function in the message function. (2) NAS-based method. Prof-CF [Wang *et al.*, 2022b] aim to design an expressive search space by pruning operations. We conduct the experiments following the settings and using the codes provided by this paper.

The construction of LLM-GNN baseline. Apart from that, we provide the LLM-GNN baseline, for which the GNN and hyper-parameters are suggested by LLM (GPT-4o) given the dataset description and statistics, the details can be found in Table 11. They are obtained based on the following prompts:

```
Your are an expert on graph learning.
Firstly, could you please describe
the dataset {D} used in the task {T}.
Then, you can suggest one GNN that
could achieve better performance on
this dataset.  Here are the design
dimensions you can refer to:  the
aggregation operation (message-passing
layer), the activation function, the
layer numbers of the designed GNN,
the skip-connections beyond layer,
the function that integrating the
features from different layers, the
hyper-parameters, and etc.
```

For the non-homophilous dataset genius, we further provide the dataset introduction and experimental observations as shown in the following:

```
This is a non-homophilous dataset
that connected nodes may have different
labels.  For this dataset, using MLPs
may have better performance than general
GNNs.
```

For the item ranking task, we provide one additional sentence in the prompt:

```
Multiple GNNs can be added when
extracting the results, and then you
can choose the numbers of GNNs and the
combination function.
```

B.3 Performance Comparisons

In this section, we show the performance comparisons with the baselines to evaluate the effectiveness of the proposed method and the implemented instance in graph learning over different tasks. As indicated in Table 13, our method outperforms all others across four node classification datasets, even when compared to NAS-based methods that focus on either architecture topology or aggregation operation. For LLM-GNN, where the GNN and hyper-parameters are directly suggested by LLM, the performance is subpar compared to the SAGE baseline due to inappropriate hyper-parameters. They fail to outperform well-designed baselines provided by research, such as GPRGNN and ACM-GNN used in this table. This highlights the challenge LLMs face in keeping up-to-date with the latest knowledge in graph learning. In contrast, our method, which designs the flexible and versatile graph

Table 8: The statistics of four graph classification datasets.

Dataset	# Graphs	# Feature	# Classes	Avg.# of Nodes	Avg.# of Edges
NCI1	4,110	89	2	29.87	32.3
NCI109	4,127	38	2	29.69	32.13
DD	1,178	89	2	384.3	715.7
PROTEINS	1,113	3	2	39.1	72.8
ogbg-molhiv	41,127	9	2	25.5	27.5

Table 9: The statistics of two datasets used in item ranking task.

	Epinions	Amazon-Sports
# Users	40,163	11,435
# Items	139,738	5, 405
# Interactions	664, 824	108, 004
# Rating Scale	[1,5]	[1, 5]
Density	0.012%	0.17%

learning approach with the assistance of external knowledge, rather than solely relying on LLMs, exhibits superior performance. Similar results are observed in the graph classification datasets in Table 14. LLM-GNN struggles to outperform existing methods, whether they overlook hierarchical information or use models with insufficient depth.

For the item ranking task, we evaluate the RMSE performance on two datasets as shown in Table 15. It can be observed that LLMGraph achieves higher performance than human-designed methods, which demonstrates its effectiveness. When compared to Prod-CF [Wang *et al.*, 2022b], it neglects multiple components in GNNs, such embedding dim and layer combination, resulting in inferior performance. Furthermore, when we use LLM to directly suggest GNNs, we mention multiple components in the prompt, but only Epinions considers this design dimension when designing GNNs. These findings empirically demonstrate the outdated information maintained in LLMs and their limited ability to design more powerful GNNs based on the latest prior knowledge.

The evaluations conducted on three tasks demonstrate the feasibility of LLMGraph in handling diverse user requirements that may arise in real-world scenarios. It has the potential to achieve superior or comparable performance to human-designed methods. Given its low requirements for coding ability and domain knowledge of graph learning, LLMGraph is particularly accessible to non-expert users, including those unfamiliar with graph learning. From this perspective, it serves as a versatile approach for designing effective graph learning solutions.

B.4 Resource Cost

In this paper, we use GPT-4o as the core of our agent to design GNNs for different tasks and graphs. Then, we empirically evaluate the resource cost on the Actor dataset. For simplicity, the cost of knowledge retrieval is grouped into each graph learning agent. As shown in Table 12, the Planning Agent, Data Agent and Configuration Agent take less than one minute to generate the outputs, which can be ignored compared with the long time required for experimental evaluations. Additionally, it costs 0.187 USD to designing GNN for given problem with the assistant of relevant knowledge, and it is expected to have lower cost considering the downward trend in API pricing. These results indicate that LLM-Graph effectively design GNNs for different graph learning problems in an efficient and economic manner.

B.5 The statistics, task distributions, and examples of the extracted knowledge.

As shown in Table 16, we provide the statistics of the extracted knowledge from four types of resource types, including academic papers, OGB technical reports, PyG methods, and benchmark results. The table includes the quantity of knowledge entries and the distribution of different types of knowledge across three levels of graph learning: Graph-level, Node-level, and Link-level. Additionally, examples of these knowledge entries are also provided.

Table 10: The statistics of four node classification datasets.

Datasets	#Nodes	#Edges	#Features	#Classes
Cora [Sen *et al.*, 2008]	2,708	5,278	1,433	7
Computers [Shchur *et al.*, 2018]	13,381	245,778	767	10
Photo [Shchur *et al.*, 2018]	7,487	238,162	745	8
Physics [Shchur *et al.*, 2018]	34,493	495,924	8,415	5
Genius [Lim *et al.*, 2021]	421,961	984, 979	12	2
Actor [Tang *et al.*, 2009]	7600	33544	931	5

Table 11: The suggested solutions used in LLM-GNN baseline.

Data	Suggestions
Cora	GNN: Two-layer stakcing-based GCN. Hyper-parameters: { hidden size:16, dropout ratio:0, learning rate:0.01, weight decay:5e-4 }
Computer	GNN: Two-layer stakcing-based GCN. Hyper-parameters: { 'hidden':256, 'dropout':0.1, 'lr':0.01, 'l2':5e-4,' }
Photo	GNN: Two-layer GCN in which the aggregation results are concatenated to predict the node labels. Hyper-parameters: { 'hidden':128, 'dropout':0.1, 'lr':0.005, 'l2':5e-4, }
Physics	GNN: Two-layer GCN in which the aggregation results are concatenated to predict the node labels. Hyper-parameters: { 'hidden':128, 'dropout':0.1, 'lr':0.005, 'l2':5e-4, }
Genius	GNN: Two-layer GraphSAGE with mean aggregator. Hyper-parameters: {'activation': ReLU, hidden size: 128, lr: 0.01, 'l2':5e-4}.
Actor	GNN: GraphSAGE with two layers and skip connections, applying a two-layer MLP for post-processing. Hyper-parameters: {'hidden':64}.
DD	GNN: Three-layer GIN in which the aggregation results are added, and graph representation vector is obtained with Global sum readout operation. Hyper-parameters: { 'hidden':128, 'dropout':0.1, 'lr':0.01, 'l2':5e-4 }.
PROTEINS	GNN: Two-layer stakcing-based GIN, and graph representation vector is obtained with Global sum readout operation. Hyper-parameters: { 'hidden':32, 'dropout':0, 'lr':0.01, 'l2':5e-4 }.
NCI1	GNN: Three-layer GIN in which the aggregation results are added, and graph representation vector is obtained with Global sum readout operation. Hyper-parameters: { 'hidden':64, 'dropout':0.0, 'lr':0.01, 'l2':5e-4 }.
NCI109	GNN: Three-layer GIN in which the aggregation results are added, and graph representation vector is obtained with Global sum readout operation. Hyper-parameters: { 'hidden':128, 'dropout':0.1, 'lr':0.01, 'l2':5e-4 }
Epsonion	Three components are used in the GNN. In each component, two-layer stakcing-based GCN are employed, and these components are concatenated to obtain the node representations. The user and item rep- resentations are first concatenated or summed up, and then fed into an MLP for prediction
Amazon-Sports	GNN: Two-layer GAT in which the aggregation results are concatenated to formulate the final node representations. The Hadamard operation is adopted when calculating the messages. The final prediction is obtained based on the dot product of the user and item representations.

Table 12: The resource cost of LLMGraph on the Actor dataset.

Agents	Time (s)	Token (K)	Money (USD$)
Planning	5.94	8.1	0.024
Data	39.11	14.1	0.043
Configuration	24.67	24.6	0.105
Evaluation	755.42	5.1	0.015

Table 13: The performance comparisons on the node classification task.

	Cora	Photo	Computer	Actor	Genius	Avg. Rank
LLMGraph (ours)	87.10(0.36)	96.11(0.33)	93.00(0.18)	40.93(0.35)	90.89(0.11)	1.4
LLMGraph (GL)	86.68(0.40)	95.50(0.21)	92.47(0.20)	39.59(0.39)	90.33(0.15)	3
GCN	85.68(0.61)	93.13(0.27)	90.52(0.42)	33.98(0.76)	89.10(0.13)	7
GIN	85.23(0.56)	92.67(0.57)	57.18(0.21)	32.81(1.05)	88.33(0.38)	8.6
SAGE	86.18(0.35)	94.60(0.25)	90.53(0.31)	39.28(0.18)	89.71(0.09)	5.2
GPRGNN	87.62(0.48)	91.93(0.26)	88.90(0.37)	40.74(0.53)	90.05(0.31)	5
ACM-GCN	86.67(0.14)	94.35(0.65)	88.58(0.39)	41.84(1.15)	90.08(0.05)	4.4
F2GNN	86.57(0.32)	95.38(0.30)	91.42(0.26)	40.39(0.03)	90.81(0.04)	3.4
LLM-GNN	84.64(1.04)	93.73(0.38)	89.20(1.16)	38.92(0.07)	89.31(0.17)	7

Table 14: The performance comparisons on the graph classification task.

	NCI1	DD	Proteins	ogbg-molhiv	NCI109	Avg. Rank
LLMGraph (ours)	81.48(1.78)	78.27(2.57)	75.44(0.93)	74.27(1.54)	81.55(1.91)	1.4
LLMGraph (GL)	81.12(1.98)	77.69(2.24)	74.88(1.16)	73.37(1.23)	81.25(1.53)	3.8
GCN	76.96(3.07)	73.59(4.17)	74.84(3.07)	73.89(1.46)	75.70(4.03)	5
SAGE	64.12(1.07)	76.99(2.74)	73.87(2.42)	73.46(1.69)	75.53(1.64)	6.8
GIN	77.95(1.95)	74.62(2.74)	74.50(4.10)	74.22(1.53)	73.25(2.67)	5.4
DGCNN	76.08(1.03)	61.63(5.33)	73.95(3.04)	68.91(2.24)	74.58(3.99)	7.6
DiffPool	75.04(1.98)	77.85(3.53)	75.11(2.14)	74.59(0.75)	71.48(2.46)	4.2
LRGNN	81.59(1.45)	77.03(2.48)	74.58(2.61)	73.51(3.21)	81.39(1.92)	3.4
LLM-GNN	71.70(2.58)	75.12(3.44)	74.47(3.65)	72.93(0.90)	73.04(3.65)	7.4

Table 15: The performance comparisons on the item ranking task.

	Epinions	Amazon-Sports	Avg. Rank
LLMGraph (ours)	0.8823(0.0083)	0.9298(0.0071)	1.5
LLMGraph (GL)	0.9088(0.0093)	0.9622(0.0103)	3.5
GCN	1.0753(0.0098)	1.0832(0.0077)	8.5
SAGE	0.9416(0.0081)	0.9900(0.0125)	5.5
NCF	1.0070(0.0055)	0.9342(0.0008)	5
NGCF	1.1437(0.0240)	1.0668(0.0038)	8.5
LightGCN	0.9926(0.0001)	0.9705(0.0003)	6
Prof-CF	0.8729(0.0014)	0.9327(0.0006)	1.5
LLM-GNN	0.9737(0.0101)	0.9670(0.0079)	5

Table 16: The statistics, task distributions, and examples of the extracted knowledge.

Resource Type	Number of Entry	Distributions	Example
Academic Paper	225	Graph-level:73 , Node-level:122, Link-level:30.	{ 'Paper Name': 'Towards Deeper Graph Neural Networks', 'Method Name': 'Deep Adaptive Graph Neural Network (DAGNN)', 'Method Summary': { 'GNN Design': { 'Agg Ops': 'Graph convolutions are decoupled into two processes: · · · .', · · · 'Activation': 'The model uses a sigmoid activation function for · · · .' } }, 'Experiment Summary': { 'Datasets': ['Cora', 'Amazon Photo', 'CiteSeer', · · ·], 'Dataset Summary': { 'Cora': {'Num Classes': 7,· · · }, · · · 'CiteSeer': {'Num Classes': 6,· · · } }, 'Baseline': ['MLP', 'ChebNet', 'GCN', 'GAT', 'APPNP', 'SGC'], 'Performance Comparisons': { 'Cora': {'GCN': '0.8492',· · · }, · · · 'CiteSeer': {'GCN': '0.7115',· · · } } } }
OGB	227	Graph-level:53, Node-level:79, Link-level:95.	{ "Task Description": "Node Property Prediction", "Dataset Name": "ogbn-papers100M", "Method": "SGC", · · · . "Hardware": "Xeon E7-8890x (1.5TB CPU)", "Paper Summary": "The paper 'Simplifying Graph Convolutional Networks' introduces · · · ." }
PYG	66	Graph-level:26 , Node-level:30, Link-level:10.	{ "Operation Name": "FAConv", "Description": "The Frequency Adaptive Graph Convolution operator · · · .", "Model design and experimental setup":{ "GNN Design": { "Agg Ops": "Low-pass and high-pass filters to · · · .", · · · . "Activation": "Uses a tanh function within the self-gating mechanism." }, "Experimental Setup": { "Datasets": "Cora, Citeseer, Pubmed, Chameleon, Squirrel, and Actor networks.", · · · . "Performance Comparisons": "FAGCN showed superior performance across multiple datasets · · · ." } }, "Paper Summary": "The paper discusses the design of a novel graph convolutional network model · · · " }
Benchmark	10	Graph-level:5, Node-level:2, Link-level:3.	{ "Task Description": "Graph Regression with ZINC dataset · · · ", "Dataset Description": "The ZINC dataset includes 12K molecular graphs · · · ", "Performance Summary": { "Detailed Performance": { "GraphSage": { "MAE": "$0.368 \pm 0.011(Test), 0.128 \pm 0.009(Train)$", · · · . "Layers": "16" }, · · · }, } }