Automated Graph Machine Learning

XIN WANG, Department of Computer Science and Technology, BNRist, Tsinghua University, China
HAOYANG LI, Department of Computer Science and Technology, Tsinghua University, China
HAIBO CHEN, Department of Computer Science and Technology, Tsinghua University, China
ZIWEI ZHANG, Department of Computer Science and Technology, Tsinghua University, China
WENWU ZHU*, Department of Computer Science and Technology, BNRist, Tsinghua University, China

Graph machine learning has been extensively studied in both academic and industry. However, as the literature on graph learning booms with a vast number of emerging methods and techniques, it becomes increasingly difficult to manually design the optimal machine learning algorithm for different graph-related tasks. To tackle the challenge, automated graph machine learning, which aims at discovering the best hyperparameter and neural architecture configuration for different graph tasks/data without manual design, is gaining an increasing number of attentions from the research community. In this paper, we extensively discuss automated graph machine learning approaches, covering hyper-parameter optimization (HPO) and neural architecture search (NAS) for graph machine learning. We briefly overview existing libraries designed for either graph machine learning or automated machine learning respectively, and further in depth introduce AutoGL, our dedicated and the world's first open-source library for automated graph machine learning. Also, we describe a tailored benchmark that supports unified, reproducible, and efficient evaluations. Last but not least, we share our insights on future research directions for automated graph machine learning. To the best of our knowledge, this work presents systematic and comprehensive discussions of approaches, libraries, as well as research directions in automated graph machine learning.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Information systems** → **Data mining**.

Additional Key Words and Phrases: Graph Machine Learning, Graph Neural Network, Automated Machine Learning, AutoML, Neural Architecture Search, Hyper-parameter Optimization

ACM Reference Format:
Xin Wang, Haoyang Li, Haibo Chen, Ziwei Zhang, and Wenwu Zhu. 2025. Automated Graph Machine Learning. *J. ACM* 37, 4, Article 111 (August 2025), 49 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Graph data is ubiquitous in our daily life. We can use graphs to model the complex relationships and dependencies between entities ranging from small molecules in proteins and particles in physical simulations to large national-wide power grids and global airlines. Therefore, graph machine learning, i.e., machine learning on graphs, has long been an important research direction for both academics

*Corresponding Author

Authors' Contact Information: Xin Wang, Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China, xin_wang@tsinghua.edu.cn; Haoyang Li, Department of Computer Science and Technology, Tsinghua University, Beijing, China, lihy218@gmail.com; Haibo Chen, Department of Computer Science and Technology, Tsinghua University, Beijing, China, chb24@mails.tsinghua.edu.cn; Ziwei Zhang, Department of Computer Science and Technology, Tsinghua University, Beijing, China, zwzhang@tsinghua.edu.cn; Wenwu Zhu, Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China, wwzhu@tsinghua.edu.cn.

ACM 1557-735X/2025/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

and industry [83, 117]. In particular, network embedding [16, 29, 55, 64] and graph neural networks (GNNs) [26, 110, 168, 169, 205, 215] have drawn increasing attention in the last decade [89, 90]. They are successfully applied to recommendation systems [85, 105, 111, 186], information retrieval [28, 41, 102, 153], fraud detection [1], bioinformatics [144, 220], physical simulation [75], traffic forecasting [99, 192], knowledge representation [156], drug re-purposing [61, 68, 92] and pandemic prediction [73] for Covid-19.

Despite the popularity of graph machine learning algorithms, the existing literature heavily relies on manual hyper-parameter or architecture design to achieve the best performance, resulting in costly human efforts when a vast number of models emerge for various graph tasks. Take GNNs as an example, at least one hundred new general-purpose architectures have been published in top-tier machine learning and data mining conferences in the year of 2021 alone, not to mention cross-disciplinary researches of task-specific designs. More and more human efforts are inevitably needed if we stick to the manual try-and-error paradigm in designing the optimal algorithms for targeted tasks.

On the other hand, automated machine learning (AutoML) has been extensively studied to reduce human efforts in developing and deploying machine learning models [65, 181]. Complete AutoML pipelines have the potential to automate every step of machine learning, including auto data collection and cleaning, auto feature engineering, and auto model selection and optimization, etc. Due to the popularity of deep learning models, hyper-parameter optimization (HPO) [7, 8, 107, 143] and neural architecture search (NAS) [38, 52, 97, 154, 159, 167, 170] are most widely studied. AutoML has achieved or surpassed human-level performance [104, 125, 222] with little human guidance in areas such as computer vision [133, 221].

Automated graph machine learning, combining advantages of AutoML and graph machine learning, naturally serves as a promising research direction to further boost the model performance, which has attracted an increasing number of interests from the community. In this paper, we provide a systematic overview of approaches for automated graph machine learning, with a particular focus on methods from the perspectives of hyperparameter optimization and neural architecture search.[1], introduce related public libraries as well as our AutoGL, a open-source library for automated graph machine learning, describe a tailored benchmark that supports unified, reproducible, and efficient evaluations, and share our insights on challenges and future research directions.

Particularly, we focus on two major topics: HPO and NAS of graph machine learning. For HPO, we focus on how to develop scalable methods. For NAS, we follow the literature and compare different methods from search spaces, search strategies, and performance estimation strategies. We also briefly discuss several recent automated graph learning works that feature in different aspects such as architecture pooling, structure learning, accelerator and joint software-hardware design etc. Besides, how different methods tackle the challenges of AutoML on graphs are discussed along the way as well. Then, we review libraries related to automated graph machine learning and discuss AutoGL, the first dedicated framework and open-source library for automated graph machine learning. We highlight the design principles of AutoGL and briefly introduce its usages, which are all specially designed for AutoML on graphs. Last but not least, we point out the potential research directions for both graph HPO and graph NAS, including but not limited to *Scalability, Explainability, Out-of-distribution generalization, Robustness*, and *Hardware-aware design* etc. We believe this paper will greatly facilitate and further promote the studies and applications of automated graph machine learning in both academia and industry.

[1]We provide a paper collection about automated graph machine learning at https://github.com/THUMNLab/awesome-auto-graph-learning.

To summarize, this article provides a comprehensive and systematic review of the field. Specifically, we make significant contributions in the following aspects:

- We present a comprehensively review of hyper-parameter optimization (HPO) and neural architecture search (NAS) for graph machine learning by incorporating a wide range of recent approaches.
- We broaden the discussion of existing libraries for both graph machine learning and automated machine learning, and give an in-depth introduction to AutoGL, the first open-source library dedicated to automated graph machine learning, covering HPO and NAS.
- We introduce a tailored open-source benchmark that supports unified, reproducible, and efficient evaluation, and conduct further analysis of the results.
- We provide deep insights into future research directions for automated graph machine learning.

The rest of the paper is organized as follows. In Section 2, we introduce the fundamentals and preliminaries for automated graph machine learning by briefly introducing basic formulations of graph machine learning and AutoML. We comprehensively discuss HPO based approaches on graph machine learning in Section 3 and NAS based methods for graph machine learning in Section 4. Then, in Section 5.1, we overview related libraries for graph machine learning and automated machine learning and in depth introduce AutoGL, our dedicated and the world's first open-source library tailored for automated graph machine learning. We discuss the tailored benchmark that enables fair, fully reproducible, and efficient empirical comparisons in Section 6. Last but not least, we outline future research opportunities in Section 7 and conclude the whole paper in Section 9.

2 Fundamentals and Preliminaries of Automated Graph Machine Learning

We briefly present basic problem formulations for graph machine learning, automated machine learning as well as unique characteristics for automated graph machine learning before moving to the next section.

2.1 Graph Machine Learning

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \left\{v_1, v_2, ..., v_{|\mathcal{V}|}\right\}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. The neighborhood of node v_i is denoted as $\mathcal{N}(i) = \left\{v_j : (v_i, v_j) \in \mathcal{E}\right\}$. The nodes can also have features denoted as $\mathbf{F} \in \mathbb{R}^{|\mathcal{V}| \times f}$, where f is the number of features. We use bold uppercases (e.g., $\mathbf{X}$) and bold lowercases (e.g., $\mathbf{x}$) to represent matrices and vectors, respectively.

Most tasks of graph machine learning can be divided into the following two categories:

- Node-level tasks: the tasks are associated with individual nodes or pairs of nodes. Typical examples include node classification and link prediction.
- Graph-level tasks: the tasks are associated with the whole graph, such as graph classification and graph generation.

For node-level tasks, graph machine learning models usually learn a node representation $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d}$ and then adopt a classifier or predictor on the node representation to solve the task. For graph-level tasks, a representation for the whole graph is learned and fed into a classifier/predictor.

GNNs are the current state-of-the-art in learning node and graph representations. The message-passing framework of GNNs [53] is formulated as follows.

$$\mathbf{m}_i^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{a_{ij}^{(l)}\mathbf{W}^{(l)}\mathbf{h}_i^{(l)}, \forall j \in \mathcal{N}(i)\right\}\right) \tag{1}$$

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathrm{COMBINE}^{(l)}\left[\mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)}\right]\right), \tag{2}$$

where $\mathbf{h}_i^{(l)}$ denotes the node representation of node v_i in the l^{th} layer, $\mathbf{m}^{(l)}$ is the message for node v_i, $\text{AGG}^{(l)}(\cdot)$ is the aggregation function, $a_{ij}^{(l)}$ denotes the weights from node v_j to node v_i, $\text{COMBINE}^{(l)}(\cdot)$ is the combining function, $\mathbf{W}^{(l)}$ are learnable weights, and $\sigma(\cdot)$ is an activation function. The node representation is usually initialized as node features $\mathbf{H}^{(0)} = \mathbf{F}$, and the final representation is obtained after L message-passing layers $\mathbf{H} = \mathbf{H}^{(L)}$.

For the graph-level representation, pooling methods (also called readout) are applied to the node representations

$$\mathbf{h}_{\mathcal{G}} = \text{POOL}(\mathbf{H}), \tag{3}$$

i.e., $\mathbf{h}_{\mathcal{G}}$ is the representation of $\mathcal{G}$.

2.2 Automated Machine Learning (AutoML)

Many AutoML algorithms such as HPO and NAS can be formulated as the following bi-level optimization problem:

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \mathcal{L}_{val}(\mathbf{W}^*(\alpha), \alpha) \\ \text{s.t.} \quad &\mathbf{W}^*(\alpha) = \arg\min_{\mathbf{W}} (\mathcal{L}_{train}(\mathbf{W}, \alpha)), \end{aligned} \tag{4}$$

where α is the optimization objective of the AutoML algorithm, e.g., hyper-parameters in HPO and neural architectures in NAS, $\mathcal{A}$ is the feasible space for the objective, and $\mathbf{W}(\alpha)$ are trainable weights in the graph machine learning models. Essentially, we aim to optimize the objective in the feasible space so that the model achieves the best results in terms of a validation function, and $\mathbf{W}^*$ indicates that the weights are fully optimized in terms of a training function. Different AutoML methods differ in how the feasible space is designed and how the objective functions are instantiated and optimized since directly optimizing Eq. (4) requires enumerating and training every feasible objective, which is prohibitive in practice.

Typical formulations of automated graph machine learning need to properly integrate the above formulations in Section 2.1 and Section 2.2 to form a new optimization problem.

2.3 Automated Graph Machine Learning

Automated graph machine learning, which non-trivially combines the strength of AutoML and graph machine learning, faces the following challenges.

- **The uniqueness of graph machine learning:** Unlike audio, image, or text, which has a grid structure, graph data lies in a non-Euclidean space [14]. Thus, graph machine learning usually has unique architectures and designs. For example, typical NAS methods focus on the search space for convolution and recurrent operations, which is distinct from the building blocks of GNNs [48].
- **Complexity and diversity of graph tasks:** As aforementioned, graph tasks per se are complex and diverse, ranging from node-level to graph-level problems, and with different settings, objectives, and constraints [66]. How to impose proper *inductive bias* and integrate *domain knowledge* into a graph AutoML method is indispensable.
- **Scalability:** Many real graphs such as social networks or the Web are incredibly large-scale with billions of nodes and edges [197]. Besides, the nodes in the graph are interconnected and cannot be treated as independent samples. Designing scalable AutoML algorithms for graphs poses significant challenges since both graph machine learning and AutoML are already notorious for being compute-intensive.

Approaches with HPO or NAS for graph machine learning reviewed in later sections target at handling at least one of these three challenges. As such, we will discuss approaches for automated

graph machine learning from two aspects: i) HPO for graph machine learning and ii) NAS for graph machine learning.

3 HPO for Graph Machine Learning

In this section, we review HPO for graph machine learning. The main challenge here is scalability, i.e., a real graph can have billions of nodes and edges, and each trial on the graph is computationally expensive. Next, we elaborate on how different methods tackle the efficiency challenge. Notice that we omit some straightforward HPO methods such as random search and grid search [8] since they are applied to graphs without any modification.

Tu *et al.* [150] propose AutoNE, the first HPO method specially designed to tackle the efficiency problem of graphs, to facilitate the graph hyper-parameter optimization for large-scale graph representation learning. AutoNE proposes a transfer paradigm that samples subgraphs as proxies for the large graph. Specifically, AutoNE has three modules: the sampling module, the signature extraction module, and the meta-learning module. In the sampling module, multiple representative subgraphs are sampled from the large graph using a multi-start random walk strategy. Each subgraph learns a representation by the signature extraction module. Then, AutoNE conducts HPO on the sampled subgraphs using Bayesian optimization [143] and records the results. Finally, using the HPO results and representation of subgraphs to extract meta-knowledges, AutoNE fine-tunes hyper-parameters on the large graph using the meta-learning module. In this way, AutoNE achieves satisfactory results while maintaining scalability since the knowledge of multiple HPO trials on the sampled subgraphs and a few HPO trails on the large graph are properly integrated.

Wang *et al.* [158] propose e-AutoGR to further increase the explainability of hyper-parameter optimization for automated graph representation learning, with the help of hyper-parameter importance decorrelation. e-AutoGR employs six fully explainable graph features, i.e., *number of nodes, number of edges, number of triangles, global clustering coefficient, maximum total degree value* and *number of components*, as measures for similarity between different graphs. A hyper-parameter decorrelation algorithm (HyperDeco) is proposed to decorrelate the mixed relations among different hyper-parameters given various graph features so that more accurate importance of different hyper-parameters towards model performances can be estimated through any regression approaches. The authors theoretically validate the correctness of the proposed hyper-parameter decorrelation algorithm and empirically discover that *first-order proximity* is most important for AROPE [203], *number of walks* together with *window size* is of great importance for DeepWalk [124], and *dropout* is particularly important for GCN [76].

Guo *et al.* [59] propose ITuNE to replace the sampling process of AutoNE with graph coarsening to generate a hierarchical graph synopsis. A similarity measurement module is also proposed to ensure that the coarsened graph shares sufficient similarity with the large graph. Compared with sampling, such graph synopsis can better preserve graph structural information. Therefore, JITuNE argues that the best hyper-parameters in the graph synopsis can be directly transferred to the large graph. Besides, since the graph synopsis is generated in a hierarchy, the granularity can be more easily adjusted to meet the time constraints of downstream tasks.

Yuan *et al.* [194] propose HESGA as another strategy to improve efficiency using a hierarchical evaluation strategy together with evolutionary algorithms. Specifically, HESGA proposes to evaluate the potential of hyper-parameters by interrupting training after a few epochs and calculating the performance gap with respect to the initial performance with random model weights. This gap is used as a fast score to filter out unpromising hyper-parameters. Then, the standard full evaluation, i.e., training until convergence, is adopted as the final assessor to select the best hyper-parameters to be stored in the population of the evolutionary algorithm.

Besides efficiency, Yoon *et al.* [188] propose AutoGM to focuses on studying a unified framework for various graph machine learning algorithms. Specifically, AutoGM finds that many popular GNNs and PageRank can be characterized in a framework similar to Eq. (1) with five hyper-parameters: the number of message-passing neighbors, the number of message-passing steps, the aggregation function, the dimensionality, and the non-linearity. AutoGM also adopts Bayesian optimization to optimize these hyper-parameters.

Yuan *et al.* [196] focus on the impact of selecting two types of GNN hyper-parameters (i.e., graph-related layers and task-specific layers) on the performance of GNN for molecular property prediction. They employed CMA-ES for HPO, which is a derivative-free and evolutionary black-box optimization method. The results reveal that optimizing the two types of hyper-parameters separately can result in improvement on GNN performance, and removing any of the two types of hyper-parameters may result in deteriorated performance. Even doing this means a larger search space, which seems to be more challenging given the same number of trials (limited computational resources), such a strategy can surprisingly achieve better performance. Meanwhile, their study further confirms the importance of HPO for GNNs in molecular property prediction problems.

Many molecular datasets are far smaller than other datasets in typical deep learning applications. Most HPO methods have not been explored in terms of their performances on these small datasets in molecular domain. Yuan et al. [195] conduct a theoretical analysis of common and specific features for two state-of-the-art HPO algorithms: i.e., TPE and CMA-ES, and they compare them with random search (RS). Experimental studies are carried out on several benchmarks in MoleculeNet, from different perspectives to investigate the impact of RS, TPE, and CMA-ES on HPO of GNNs for molecular property prediction. Their experimental results indicate that TPE is the most suited HPO method for GNN under molecular property prediction problems with limited computational resources. Meanwhile, RS is the simplest method capable of achieving comparable performance with TPE and CMA-ES.

GCN models are sensitive to the choice of hyper-parameters such as dropout rate and learning weight decay [158], especially for deep GCN models. Zhu et al. [219] therefore target at automating the training of GCN models through hyper-parameter optimization. To be specific, they propose a self-tuning GCN (ST-GCN) approach by incorporating *hypernets* in each graph convolutional layer, enabling the joint optimization over GCN model parameters and hyper-parameters. They further extend the approach through incorporating the population based training scheme and adopt a population based training framework to self-tuning GCN, thus alleviating local minima problem via exploring hyper-parameter space globally. Experimental results on three benchmark datasets demonstrate the effectiveness of their approaches in terms of optimizing multi-layer GCNs.

Bu et al. [15] analyze the performance of different evolutionary algorithms on automated graph machine learning through experimental study. The experimental results show that evolutionary algorithms can serve as an effective alternative to the traditional hyper-parameter optimization algorithms such as random search, grid search and Bayesian Optimization for GNN.

Sun et al. [145] propose AutoGRL, an automated graph representation learning framework for node classification task. AutoGRL consists of an appropriate search space with four components: data augmentation, feature engineering, hyper-parameter optimization, and architecture search. Given graph data, AutoGRL searches for the best graph representation learning model in the search space using an efficient searching algorithm. Extensive experiments are conducted on four real-world node classification datasets to demonstrate that AutoGRL can automatically find competitive graph representation learning models on specific graph data effectively and efficiently.

Yang et al. [179] address the underexplored issue of obtaining reliable and trustworthy predictions using automated Graph Neural Networks (GNNs). It integrates uncertainty estimation into the Hyperparameter Optimization (HPO) problem through a bilevel formulation in a novel model

named HyperU-GCN. The upper-level problem focuses on reasoning uncertainties by developing a probabilistic hypernetwork through a variational Bayesian approach. The lower-level problem targets how the weights in the Graph Convolutional Network (GCN) respond to a distribution of hyperparameters. By incorporating model uncertainty into the hyperparameter space, HyperU-GCN is able to achieve calibrated predictions, similar to Bayesian model averaging over hyperparameters. Experimental results on six public datasets indicate that this approach outperforms several state-of-the-art methods in terms of node classification accuracy and expected calibration error (ECE).

Lloyd et al. [108] focus on the challenges of embedding knowledge graphs into low-dimensional spaces, a process that is computationally expensive largely due to hyperparameter optimization. They introduce a novel approach using Sobol sensitivity analysis to evaluate the significance of different hyperparameters in affecting the quality of the embeddings. Through thousands of trials and subsequent regression analysis, they identify considerable variability in the importance of different hyperparameters across various knowledge graphs. This variability is attributed to differences in dataset characteristics. Additionally, the paper makes a unique contribution by identifying data leakage issues in the UMLS knowledge graph and presenting a leakage-robust variant, termed UMLS-43.

Yoon et al. [189] address the challenge of selecting the most suitable graph algorithm for specific real-world applications due to the proliferation of algorithms with different problem formulations, computational times, and memory footprints. To resolve this, they propose AutoGM, an automated system for graph mining algorithm development. The paper introduces a unified framework, UNIFIEDGM, which simplifies the search space for graph algorithms by requiring only five parameters for algorithm determination. AutoGM then uses Bayesian Optimization to find the optimal parameter set for UNIFIEDGM. To assess algorithmic efficacy within a given computational budget, the authors introduce a novel budget-aware objective function. Tests on various real-world datasets show that AutoGM generates novel graph algorithms that offer the best speed-accuracy trade-off compared to existing models.

Zhang et al. [202] address the issue of inefficient hyper-parameter (HP) tuning in the context of knowledge graph (KG) learning. The authors first conduct a thorough analysis of different hyper-parameters and their transferability from smaller subgraphs to full graphs. Based on these insights, they introduce a two-stage search algorithm called KGTuner. In the first stage, the algorithm efficiently explores hyper-parameter configurations using small subgraphs. In the second stage, the best-performing configurations are fine-tuned on the full, large-scale graph. Experimental results demonstrate that KGTuner outperforms baseline algorithms, achieving an average relative improvement of 9.1% across four different embedding models when applied to large-scale KGs in the open graph benchmark.

Yang et al. [178] present a systematic analysis of the impact of hyperparameters on both factorization-based and graph-sampling-based graph embedding techniques for homogeneous graphs. The authors design generalized techniques that include a wide range of hyperparameters and conduct an exhaustive experimental study with over 3,000 trained embedding models per dataset. The findings reveal that optimal hyperparameter settings, rather than the complexity of the embedding models, largely account for performance gains. The study shows that well-tuned hyperparameters can outperform a collection of 18 state-of-the-art graph embedding models by a margin of 0.30-35.41% across various tasks. Importantly, the paper notes that there is no universal set of hyperparameters that are optimal for all tasks, but offers task-specific recommendations for hyperparameter settings, which can serve as valuable guidelines for future research in embedding-based graph analyses.

Table 1. A summary of different graph neural architecture search (GNAS) methods for automated graph machine learning.

Method	Search space				Tasks		Search Strategy	Performance Estimation	Other Characteristics
	Micro	Macro	Pooling	HP	Layers	Node	Graph			
GraphNAS [48]	✓	✓	✗	✗	Fixed	✓	✗	RNN controller + RL	-	-
AGNN [216]	✓	✗	✗	✗	Fixed	✓	✗	Self-designed controller + RL	Inherit Weights	-
SNAG [211]	✓	✓	✗	✗	Fixed	✓	✗	RNN controller + RL	Inherit Weights	Simplify the micro search space
PDNAS [213]	✓	✓	✓	✗	Fixed	✓	✗	Differentiable	One-shot	-
NAS-GNN [119]	✓	✗	✓	✗	Fixed	✓	✗	Evolutionary algorithm	-	-
AutoGraph [101]	✓	✓	✗	✓	Various	✓	✗	Evolutionary algorithm	-	-
GeneticGNN [141]	✓	✗	✓	✗	Fixed	✓	✗	Evolutionary algorithm	-	-
EGAN [210]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Sample small graphs for efficiency
NAS-GCN [70]	✓	✓	✓	✗	Fixed	✗	✓	Evolutionary algorithm	-	Handle edge features
LPGNAS [214]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Search for quantization options
GraphGym [191]	✓	✓	✗	✓	Various	✓	✓	Random search	-	Transfer across datasets and tasks
SGAS [81]	✓	✗	✗	✗	Fixed	✓	✓	Self-designed algorithm	-	-
Peng *et al.* [123]	✓	✗	✗	✗	Fixed	✗	✓	CEM-RL [126]	-	Search spatial-temporal modules
GNAS [21]	✓	✓	✗	✗	Various	✓	✓	Differentiable	One-shot	-
AutoSTG [121]	✗	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot+meta learning	Search spatial-temporal modules
DSS [100]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Dynamically update search space
SANE [212]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	-
AutoAttend [58]	✓	✓	✗	✗	Fixed	✓	✓	Evolutionary algorithm	One-shot	Cross-layer attention
DiffMG [34]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Support heterogeneous graphs
DeepGNAS [42]	✓	✓	✗	✗	Various	✓	✗	Controller +RL	-	Alleviate over-smoothing
LLC [165]	✗	✓	✗	✗	Various	✓	✗	Differentiable	One-shot	-
FL-AGCNS [152]	✓	✓	✗	✗	Fixed	✓	✗	Evolutionary algorithm	One-shot	Federated learning setting
G-Cos [201]	✓	✗	✗	✗	Fixed	✓	✗	Evolutionary algorithm	One-shot	Accelerator search
PAS [166]	✓	✓	✓	✗	Fixed	✗	✓	Differentiable	One-shot	-
FGNAS [109]	✓	✗	✗	✗	Fixed	✓	✗	RNN controller +RL	-	Software-hardware co-design
GraphPAS [25]	✓	✗	✗	✗	Fixed	✓	✗	Evolutionary algorithm	Sharing population	Parallel search
ALGNN [19]	✓	✓	✗	✓	Various	✓	✗	MOPSO [27]	-	Consider consumption cost
EGNAS [20]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Handle edge features
AutoGEL [164]	✓	✓	✓	✗	Fixed	✓	✓	SNAS [173]	One-shot	Handle edge features
GASSO [130]	✓	✗	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Graph structure learning
G-RNA [171]	✓	✓	✗	✗	Fixed	✓	✗	Evolutionary algorithm	One-shot	Search with robustness metrics
GRACES [129]	✓	✗	✓	✗	Fixed	✗	✓	Differentiable	One-shot	Handle distribution shifts
GAUSS [57]	✓	✗	✗	✗	Fixed	✓	✗	Evolutionary algorithm	One-shot	Handle large-scale graphs
PasCa [200]	✓	✗	✗	✗	Various	✓	✗	Bayesian Optimization	-	Decouple neural message passing
PMMM [79]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Search meta paths
DHGAS [209]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Search spatial-temporal modules
MTGC3 [128]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Handle multiple tasks
OMG-NAS [18]	✓	✓	✗	✗	Fixed	✓	✓	RNN controller +RL	One-shot	Handle distribution shifts
DCGAS [183]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Handle distribution shifts
GASIM [208]	✓	✓	✓	✗	Fixed	✓	✗	Differentiable	One-shot	Handle distribution shifts
BiGNAS [52]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Handle joint tasks
CARNAS [97]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Handle distribution shifts
AutoGFM [24]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Search graph foundation model

4 NAS for Graph Machine Learning

NAS methods can be compared in three aspects [38]: search space, search strategy, and performance estimation strategy. Next, we review NAS methods for graph machine learning from these three aspects and discuss some designs uniquely for graphs. We mainly review NAS for GNNs fitting

Table 2. The typical search space of different types of aggregation weights a_{ij}. We omit the layer superscript for brevity.

Type	Formulation
CONST	$a_{ij}^{\text{const}} = 1$
GCN	$a_{ij}^{\text{gcn}} = \frac{1}{\sqrt{\lvert\mathcal{N}(i)\rvert\lvert\mathcal{N}(j)\rvert}}$
GAT	$a_{ij}^{\text{gat}} = \text{LeakyReLU}\left(\text{ATT}\left(\mathbf{W}_a\left[\mathbf{h}_i, \mathbf{h}_j\right]\right)\right)$
SYM-GAT	$a_{ij}^{\text{sym}} = a_{ij}^{\text{gat}} + a_{ji}^{\text{gat}}$
COS	$a_{ij}^{\text{cos}} = \cos\left(\mathbf{W}_a\mathbf{h}_i, \mathbf{W}_a\mathbf{h}_j\right)$
LINEAR	$a_{ij}^{\text{lin}} = \tanh\left(\text{sum}\left(\mathbf{W}_a\mathbf{h}_i + \mathbf{W}_a\mathbf{h}_j\right)\right)$
GENE-LINEAR	$a_{ij}^{\text{gene}} = \tanh\left(\text{sum}\left(\mathbf{W}_a\mathbf{h}_i + \mathbf{W}_a\mathbf{h}_j\right)\right)\mathbf{W}'_a$

Eq. (1), which is the focus of the literature. We summarize the characteristics of different methods in Table 1.

4.1 Search Space

The first challenge of NAS on graphs is the search space design since the building blocks of graph machine learning are usually distinct from other deep learning models such as CNNs or RNNs. For GNNs, the search space can be divided into the following five categories.

Micro search space

Following the message-passing framework in Eq. (1), the micro search space defines how nodes exchange messages with others in each layer. Commonly adopted micro search spaces [48, 216] compose the following components:

- Aggregation function AGG$(\cdot)$: SUM, MEAN, MAX, and MLP.
- Aggregation weights a_{ij}: common choices are listed in Table 2.
- Number of heads when using attentions: 1, 2, 4, 6, 8, 16, etc.
- Combining function COMBINE$(\cdot)$: CONCAT, ADD, and MLP.
- Dimensionality of $\mathbf{h}^l$: 8, 16, 32, 64, 128, 256, 512, etc.
- Non-linear activation function $\sigma(\cdot)$: Sigmoid, Tanh, ReLU, Identity, Softplus, Leaky ReLU, ReLU6, and ELU.

However, directly searching all these components results in thousands of possible choices in a single message-passing layer. Thus, it may be beneficial to prune the space to focus on a few crucial components depending on applications and domain knowledge [211].

4.1.1 Macro search space. Similar to residual connections and dense connections in CNNs, node representations in one layer of GNNs do not necessarily solely depend on the immediate previous layer [80, 175]. These connectivity patterns between layers form the macro search space. Formally, such designs are formulated as

$$\mathbf{H}^{(l)} = \sum\nolimits_{j<l} \mathcal{F}_{jl}\left(\mathbf{H}^{(j)}\right), \tag{5}$$

where $\mathcal{F}_{jl}(\cdot)$ can be the message-passing layer in Eq. (1), ZERO (i.e., not connecting), IDENTITY (e.g., residual connections), or an MLP. Since the dimensionality of $\mathbf{H}^{(j)}$ can vary, IDENTITY can only be adopted if the dimensionality of each layer matches.

4.1.2 Pooling methods. To handle graph-level tasks, information from all the nodes are aggregated to form graph-level representations using the pooling operation in Eq. (3). Jiang et al. [70]

propose a pooling search space including row- or column-wise sum, mean, or maximum, attention pooling, attention sum, and flatten. More advanced methods such as hierarchical pooling [187] could also be added to the search space with careful designs. For example, PAS [166] further proposes to search for adaptive pooling architectures. Firstly they design a unified framework consisting of four modules: *Aggregation*, *Pooling*, *Read out* and *Merge*, which can cover existing human-designed pooling methods (global and hierarchical) for graph classification. Based on this framework, a novel search space is designed by incorporating popular operations in human-designed architectures. To further enable efficient search, a coarsening strategy is proposed to continuously relax the search space, with the utilization of differentiable search methods. Extensive experiments on six real-world datasets from three domains are conducted, and the results demonstrate the effectiveness and efficiency of the proposed framework.

4.1.3 Hyper-parameters. Besides architectures, other training hyper-parameters can be incorporated into the search space, i.e., similar to jointly conducting NAS and HPO. Typical hyper-parameters include the learning rate, the number of epochs, the batch size, the optimizer, the dropout rate, and the regularization strengths such as the weight decay. These hyper-parameters can be jointly optimized with architectures or separately optimized after the best-architectures are found. HPO methods in Section 3 can also be combined here.

4.1.4 Layers. Another critical model choice not incorporated in the above four categories is the number of message-passing layers. Unlike CNNs, most currently successful GNNs are shallow, e.g., with no more than three layers, possibly due to the over-smoothing problem [80, 98]. Limited by this problem, the existing NAS methods for GNNs preset the number of layers as a fixed small number. Except for a recent attempt DeepGNAS [42], how to automatically design deep GNNs while integrating techniques to alleviate over-smoothing remains mostly unexplored.

4.2 Search Strategy

The search strategy can be broadly divided into three categories: architecture controllers trained with reinforcement learning (RL), differentiable methods, and evolutionary algorithms.

4.2.1 Controller + RL. A widely adopted NAS search strategy is to use a controller to generate the neural architecture descriptions and train the controller with reinforcement learning to maximize the model performance as rewards. For example, if we consider neural architecture descriptions as a sequence, we can use RNNs as the controller [222]. Such methods can be directly applied to GNNs with a suitable search space and performance evaluation strategy.

4.2.2 Differentiable. Differentiable NAS methods such as DARTS [104] and SNAS [173] have gained popularity in recent years. Instead of optimizing different operations separately, differentiable methods construct a single super-network (known as the *one-shot model*) containing all possible operations. Formally, we denote

$$\mathbf{y} = o^{(x,y)}(\mathbf{x}) = \sum\nolimits_{o \in O} \frac{\exp(\mathbf{z}_o^{(x,y)})}{\sum_{o' \in O} \exp(\mathbf{z}_{o'}^{(x,y)})} o(\mathbf{x}), \tag{6}$$

where $o^{(x,y)}(\mathbf{x})$ is an operation in the GNN with input $\mathbf{x}$ and output $\mathbf{y}$, O are all candidate operations, and $\mathbf{z}^{(x,y)}$ are learnable vectors to control which operation is selected. Briefly speaking, each operation is regarded as a probability distribution of all possible operations. In this way, the architecture and model weights can be jointly optimized via gradient-based algorithms. The main challenges lie in making the NAS algorithm differentiable, where several techniques such as Gumbel-softmax [69] and concrete distribution [112] are resorted to. When applied to GNNs, slight modification may be

needed to incorporate the specific operations defined in the search space, but the general idea of differentiable methods remains unchanged.

4.2.3 Evolutionary Algorithms. Evolutionary algorithms are a class of optimization algorithms inspired by biological evolution. For NAS, randomly generated architectures are considered initial individuals in a population. Then, new architectures are generated using mutations and crossover operations based on the population. The architectures are evaluated and selected to form the new population, and the same process is repeated. The best architectures are recorded while updating the population, and the final solutions are obtained after sufficient updating steps.

For GNNs, regularized evolution (RE) NAS [133] has been widely adopted. RE's core idea is an aging mechanism, i.e., in the selection process, the oldest individuals in the population are removed. Genetic-GNN [140] also proposes an evolution process to alternatively update the GNN architecture and the learning hyper-parameters to find the best fit of each other.

4.2.4 Combinations. It is also feasible to combine these three types of search strategies mentioned above. For example, AGNN [216] proposes a reinforced conservative search strategy by adopting both RNNs and evolutionary algorithms in the controller and train the controller with RL. By only generating slightly different architectures, the controller can find well-performing GNNs more efficiently. Peng et al. [123] adopt CEM-RL [126], which combines evolutionary and differentiable methods.

4.3 Performance Estimation Strategy

Due to the large number of possible architectures, it is infeasible to fully train each architecture independently. Next, we review some performance estimation strategies.

A commonly adopted "trick" to speed up performance estimation is to reduce fidelity [38], e.g., by reducing the number of epochs or the number of data points. This strategy can be directly generalized to GNNs.

Another strategy successfully applied to CNNs is sharing weights among different models, known as parameter sharing or weight sharing [125]. For differentiable NAS with a large one-shot model, parameter sharing is naturally achieved since the architectures and weights are jointly trained. However, training the one-shot model may be difficult since it contains all possible operations. To further speed up the training process, single-path one-shot model [60] has been proposed where only one operation between an input and output pair is activated during each pass.

For NAS without a one-shot model, sharing weights among different architecture is more difficult but not entirely impossible. For example, since it is known that some convolutional filters are common feature extractors, inheriting weights from previous architectures is feasible and reasonable in CNNs [132]. However, since there is still a lack of understandings of what weights in GNNs represent, we need to be more cautious about inheriting weights [211]. AGNN [216] proposes three constraints for parameter inheritance: same weight shapes, same attention and activation functions, and no parameter sharing in batch normalization and skip connections.

4.4 Recent Featured Works

In this section, we discuss several recent advances in automated graph machine learning that feature in taking topological structure learning, robustness and generalization, scalability, data heterogeneity, efficient architecture search or software-hardware co-design into considerations.

4.4.1 Architecture Search with Graph Structure Learning. Qin *et al.* [130] investigate the important question that *how NAS is able to select the desired GNN architectures* by conducting a measurement study with experiments, which discovers that gradient based NAS methods tend to

select proper architectures based on the usefulness of different types of information with respect to the target task. The explorations further show that gradient based NAS also suffers from noises hidden in the graph, resulting in searching suboptimal GNN architectures. Based on these findings, they propose a Graph differentiable Architecture Search model with Structure Optimization (GASSO), which allows differentiable search of the architecture with gradient descent and is able to discover graph neural architectures with better performance through employing graph structure learning as a denoising process in the search procedure. The proposed GASSO model is capable of simultaneously searching the optimal architecture and adaptively adjusting graph structure by jointly optimizing graph architecture search and graph structure denoising. Extensive experiments on real-world graph datasets demonstrate that the proposed GASSO model is able to achieve state-of-the-art performance compared with existing baselines.

4.4.2 Robust and Generalizable Graph NAS. G-RNA [171] enhances the robustness of graph NAS methods against adversarial attacks. G-RNA designs a robust search space for the message-passing mechanism by incorporating graph structure mask operations. The graph structure mask operations cover important robust essences of graph structure and could recover various existing defense methods as well. The framework also defines a robustness metric to guide the search process and filter robust architectures. Specifically, G-RNA uses an attack proxy to produce several adversarial samples based on the clean graph, and it searches robust GNNs using the robustness metric with clean and generated adversarial samples.

Besides the robustness, GRACES [129] addresses the limitations of existing graph neural architecture search methods in dealing with distribution shifts between training and test graphs. GRACES tailors a customized GNN architecture suitable for each graph instance to handle the distribution shifts. GRACES uses a self-supervised disentangled graph encoder [86, 87, 91, 93] to characterize invariant factors hidden in diverse graph structures. It further proposes a prototype based architecture self-customization strategy to tailor specialized GNN architectures for graphs based on the similarities of their representations with operation prototypes vectors in the latent space. Finally, the customized supernet provides differentiable weights on the mixture of different operations. GRACES model can be easily optimized in an end-to-end fashion through gradient based methods

Cai *et al.* [18] propose OMG-NAS, an Out-of-distribution Generalized Multimodal Graph Neural Architecture Search framework that aims to enhance the robustness of multimodal graph neural networks (MGNNs) under distribution shifts. Conventional MGNAS methods often overfit the training distribution and capture spurious correlations, leading to suboptimal architectures when tested on out-of-distribution (OOD) data. OMG-NAS addresses these challenges through two key innovations. First, it introduces a Multimodal Graph Feature Decorrelation (MGFD) strategy, which disentangles multimodal features into unimodal components and iteratively reweights samples using random Fourier features to suppress spurious dependencies while preserving invariant predictive features. Second, it employs a Global Multimodal Sample Weight Estimator (GMSWE) that transfers optimal sample weights across different candidate architectures, enabling more stable and efficient searches. The architecture search is conducted via a reinforcement learning-based controller that optimizes MGNNs for both predictive accuracy and OOD generalization [148]. Evaluations on three real-world multimodal graph datasets under both multi-modal [163] and single-modal OOD scenarios demonstrate that OMG-NAS consistently outperforms state-of-the-art baselines, achieving higher accuracy. Ablation studies further confirm the individual contributions of MGFD and GMSWE, underscoring the framework's effectiveness in mitigating spurious correlations and enhancing generalization.

Yao *et al.* [183] propose Data-augmented Curriculum Graph Neural Architecture Search (DCGAS), a framework designed to enhance the generalization of graph NAS under distribution shifts. Existing methods customize architectures for individual graphs but neglect the role of data augmentation and

the varying importance of training samples, limiting their ability to generalize to unseen distributions. DCGAS addresses these issues through two key innovations. First, it introduces an embedding-guided data generator based on a discrete graph diffusion model, which synthesizes new graphs structurally similar to given inputs, enriching the training distribution and improving the architecture customizer's adaptability. Second, it incorporates a two-factor uncertainty-based curriculum weighting strategy that measures both architecture-level and data-level uncertainties, assigning higher training weights to more informative graphs. This approach ensures that the model prioritizes samples critical for improving out-of-distribution performance. Experiments on synthetic datasets with controlled distribution shifts (Spurious-Motif) and real-world molecular property prediction tasks demonstrate that DCGAS consistently outperforms state-of-the-art baselines, including GRACES, particularly in low-data regimes, confirming its effectiveness in learning robust mappings from graph data to architectures.

Zhang *et al.* [208] propose the Disentangled Continual Graph Neural Architecture Search with Invariant Modular Supernet (GASIM) to address the largely unexplored problem of continually searching optimal GNN architectures for sequentially arriving graph tasks without forgetting past knowledge. Existing GNAS methods assume static tasks and entangle architecture factors, leading to catastrophic forgetting in continual settings due to architecture conflicts, where the optimal design for a new task may harm performance on previous ones. GASIM mitigates this by introducing a modular graph architecture super-network, where each module specializes in tasks with similar latent graph factors, enabling disentangled task–architecture relationships. A factor-based task–module router predicts latent graph factors via distribution matching and routes incoming tasks to the most suitable module, reducing interference across tasks. An invariant architecture search mechanism further captures shared knowledge within each module by enforcing an invariance loss across tasks routed to the same module, balancing adaptation to new tasks with retention of past performance. Extensive experiments on CoraFull, Arxiv, and Reddit under the challenging class-incremental setting show that GASIM consistently outperforms both manually designed GNNs and existing GNAS baselines, improving performance while alleviating forgetting effects.

Li *et al.* [97] propose Causal-aware Graph Neural Architecture Search (CARNAS), targeting the challenge of distribution shifts in Graph NAS by uncovering stable causal relationships between graph structures and architectures. Unlike prior methods that rely solely on graph–architecture correlations—which may be spurious and unstable under shifting distributions—CARNAS introduces three modules to explicitly model causality: (1) *Disentangled Causal Subgraph Identification*, which uses disentangled GNN layers to extract causal subgraphs with stable predictive power across environments; (2) *Graph Embedding Intervention*, which performs interventions in the latent space by combining causal subgraph embeddings with non-causal ones to simulate diverse environments, aided by a supervised loss to preserve essential features; and (3) *Invariant Architecture Customization*, which constructs GNN architectures from causal subgraph representations and enforces causal invariance via variance regularization across intervention environments. This design aims to ensure that the learned architecture depends solely on causal subgraphs, enhancing out-of-distribution generalization. Experiments on synthetic (Spurious-Motif) and real-world (OGBG-Mol*) datasets demonstrate that CARNAS consistently surpasses both manually designed GNNs and existing NAS baselines, achieving superior performance under significant distribution shifts while maintaining computational efficiency.

4.4.3 Large-scale Graph NAS. Guan *et al.* [57] presents the graph architecture search at scale (GAUSS) method, designed to address the limitations of existing graph NAS approaches in handling large-scale graphs. Traditional graph NAS methods are computationally intensive and suffer from the consistency collapse issues, making them unsuitable for large graphs. GAUSS tackles these

problems by introducing an efficient light-weight supernet and a joint architecture-graph sampling technique. Specifically, it proposes a graph sampling-based single-path one-shot supernet to minimize computational load. To handle consistency issues, the method incorporates a unique architecture peer learning mechanism on sampled sub-graphs, as well as an architecture importance sampling algorithm. These innovations aim to smooth the highly non-convex optimization objective and stabilize the architecture sampling process. Theoretical analyses and empirical tests on five different datasets, ranging from 10^4 to 10^8 vertices, show that GAUSS outperforms existing GNAS methods, marking it as the first framework capable of efficiently handling large-scale graphs with billions of edges within a single GPU day.

Zhang *et al.* [200] further introduces PasCa, a new system aimed at systematically exploring the design space for scalable graph neural networks. It addresses the limitations of current GNNs that are not well-suited for scalability. Through the deconstruction of the neural message-passing mechanism, the authors propose a novel scalable graph neural architecture paradigm (SGAP) that includes a design space with up to 150,000 different architectures. To navigate this expansive design space, it also presents an auto-search engine capable of multi-objective optimization to find GNN architectures that are both efficient and accurate. Empirical studies across ten benchmark datasets reveal that the architectures discovered by PasCa, specifically PasCa-V3, not only offer competitive predictive accuracy but also achieve up to 28.3 times faster training speeds compared to state-of-the-art methods like JK-Net [175].

4.4.4 Heterogeneous Graph NAS. Heterogeneous information networks (HINs) are used to describe real-world data with intricate entities and relationships. Several recent works [35, 50, 79] study the neural architecture search on HINs to handle the heterogeneous node types and relationships. Specifically, Li *et al.* [79] proposes a method called Partial Message Meta Multigraph search (PMMM) for automatically optimizing the neural architecture design on Heterogeneous Information Networks (HINs), aiming at better stability and flexibility. It adopts an efficient differentiable framework to search for a meaningful meta multigraph, which captures more flexible and complex semantic relations than a meta graph. The authors also propose a stable algorithm called partial message search to ensure that the searched meta multigraph consistently outperforms manually designed meta-structures. Experimental results on six benchmark datasets for node classification and recommendation tasks demonstrate the effectiveness of PMMM. The proposed method outperforms state-of-the-art heterogeneous GNNs, discovers meaningful meta multigraphs, and exhibits significantly improved stability.

Zhang *et al.* [209] first consider the temporal information on heterogeneous graphs, and propose a method called Dynamic Heterogeneous Graph Attention Search (DHGAS) for automating the design of Dynamic Heterogeneous Graph Neural Networks (DHGNNs). The existing DHGNNs are manually designed and lack adaptability to diverse dynamic heterogeneous graph scenarios. To overcome this limitation, the authors introduce a search space that considers spatial-temporal dependencies and heterogeneous interactions in graphs and develop an efficient search algorithm. The proposed DHGAS method can automatically discover optimal DHGNN architectures without human guidance. The authors introduce a unified dynamic heterogeneous graph attention (DHGA) framework that enables nodes to attend to their heterogeneous and dynamic neighbors. They also design a localization space to determine attention application and a parameter.

4.4.5 Graph Transformer & Foundation Model Architecture Search. Besides searching the architecture of graph neural networks, AutoGT [207] is proposed to search the architecture of graph transformer, which is another type of strong graph encoder [185]. However, unlike their applications in text and image data, using Transformers for graph data involves additional complexities due to the non-euclidean nature of graphs. The authors identify two main challenges. The first challenge is

how to creat a unified search space for graph Transformers. The second challenge is how to handle the complex relationship between the architecture of each Transformer layer and graph encoding strategies. To address these challenges, they introduce Automated Graph Transformer (AutoGT), a neural architecture search framework designed for graphs. AutoGT uses a unified graph Transformer formulation and includes a comprehensive search space that considers both architectural and encoding options. To manage the coupling between architecture and graph encodings, the authors propose an encoding-aware performance estimation strategy. Through rigorous experiments, they demonstrate that AutoGT outperforms state-of-the-art hand-crafted models, establishing its effectiveness and broad applicability.

Chen *et al.* [24] propose AutoGFM, an Automated Graph Foundation Model with Adaptive Architecture Customization, to address the limitation of existing graph foundation models (GFMs) that rely on fixed, hand-crafted graph neural network (GNN) architectures. Such fixed designs often lead to suboptimal performance when transferring across diverse domains and tasks, due to the phenomenon of *architecture inconsistency*—the optimal GNN architecture can vary significantly between datasets. AutoGFM mitigates this by discovering an invariant graph–architecture relationship and dynamically customizing architectures for each dataset while still enabling knowledge sharing via a weight-sharing supernet. The framework comprises three key components: (i) a *disentangled contrastive graph encoder* that extracts invariant and variant patterns from graph data; (ii) an *invariant-guided architecture customization* strategy that uses invariant patterns to predict architectures while shielding them from spurious variant influences; and (iii) a *curriculum architecture customization mechanism* to prevent early-stage bias where certain datasets dominate the architecture search process. Extensive experiments on eight datasets spanning node-, edge-, and graph-level tasks demonstrate that AutoGFM achieves state-of-the-art results, outperforming both manually designed GNNs and existing graph neural architecture search (GNAS) baselines in pre-training, fine-tuning, and few-shot learning settings.

4.4.6 Multi-task Learning. Qin *et al.* [128] propose MTGC3, a multi-task graph neural architecture search framework with task-aware collaboration and curriculum learning, aiming to address the previously unexplored challenge of automatically discovering optimal GNN architectures for multiple tasks simultaneously. Unlike conventional GNAS methods that focus on single-task scenarios, MTGC3 introduces a structurally diverse supernet to manage multiple architectures and graph structures in a unified framework, enabling customized architectures for each task. To facilitate information sharing without compromising architectural independence, it incorporates a soft task-collaborative module that adaptively learns transferability patterns among tasks. Furthermore, a task-wise curriculum training strategy is developed to balance task contributions during optimization by adjusting the influence of tasks according to their relative difficulties, thereby stabilizing the search process. Experiments on synthetic datasets and real-world benchmarks such as OGBG-Tox21, OGBG-ToxCast, and OGBG-Sider demonstrate that MTGC3 consistently outperforms both manually designed GNNs and state-of-the-art GNAS methods, effectively capturing inter-task collaboration while tailoring architectures to diverse task requirements.

Ge *et al.* [52] propose Behavior Importance-Aware Graph Neural Architecture Search (BiGNAS), a unified framework for cross-domain recommendation that jointly optimizes GNN architecture and behavior data importance. Existing GNN-based cross-domain recommendation (CDR) methods often rely on fixed, manually designed architectures and fail to effectively filter noisy source-domain behaviors, leading to limited adaptability and negative transfer. To address these issues, BiGNAS introduces two key components: a Cross-Domain Customized Supernetwork and a Graph-Based Behavior Importance Perceptron. The supernetwork adopts a one-shot, retrain-free paradigm to automatically search domain-specific optimal GNN architectures within a recommendation-oriented

search space, enabling continuous architecture optimization without retraining. The perceptron leverages auxiliary learning to dynamically evaluate the importance of user behaviors in the source domain at domain, item, and user levels, guiding the supernetwork toward more effective information transfer. These modules are integrated in a bi-level optimization framework using implicit gradients, allowing end-to-end training. Experiments on benchmark Amazon CDR datasets and a large-scale industrial advertising dataset demonstrate that BiGNAS achieves consistent and significant performance gains over state-of-the-art baselines, with up to 10.57

4.4.7 Search of Efficient Architectures. G-Cos [201] is a GNN and accelerator co-search framework that can automatically search for matched GNN structures and accelerators to maximize both task accuracy and acceleration efficiency. Specifically, G-CoS integrates two major components: i) a generic GNN accelerator search space which is applicable to various GNN structures and ii) a one-shot GNN and accelerator co-search algorithm that enables simultaneous and efficient search for optimal GNN structures as well as their matched accelerators. Extensive experiments and ablation studies show that the GNNs together with accelerators generated by G-CoS consistently outperforms state-of-the-art GNNs and GNN accelerators in terms of both task accuracy and hardware efficiency, while only requiring a few hours for the end-to-end generation of the best matched GNNs and their accelerators. Similarly, LPGNAS [214] jointly searches for architectures and quantisation choices so that both model and buffer sizes can be greatly reduced while keeping similar accuracy as other methods. Their empirical results show that 4-bit weights, 8-bit activations quantisation strategy might be the key for GNNs. ALGNN [19] considers the computation cost and complexity of the searched model using a multi-objective optimization method.

4.4.8 Co-design of Software and Hardware. Lu *et al.* [109] propose FGNAS as the first software-hardware co-design framework for automating the search and deployment of GNNs. Using FPGA as the target platform, the FGNAS framework is able to perform the FPGA-aware graph neural architecture search. FPGA is employed as the vehicle for illustration and implementation of the methods. Specific hardware constraints are considered so that quantization is adopted to compress the model. Under specific hardware constraints, they show the FGNAS framework can successfully identify a solution of higher accuracy while using shorter time than random search and the traditional two-step tuning. To evaluate the design, they conduct experiments on benchmark datasets, i.e., Cora, CiteSeer and PubMed, and the results show that the proposed FGNAS framework has better capability in optimizing the accuracy of GNNs when the hardware implementation is specifically constrained.

4.4.9 A Unifying Invariance Perspective on GNAS. Recent GNAS research spans structure learning, robustness, out-of-distribution (OOD) generalization, efficiency on large-scale graphs, heterogeneous and multimodal settings, and hardware–algorithm co-design. While these directions may appear disparate, they can be viewed through a unified lens of *invariance*: effective architectures discovered by NAS should remain stable under variations in graph structure, environments, computational resources, and data modalities. From this perspective, prior work can be organized along four complementary invariance dimensions: (i)Structural invariance, where methods such as GASSO [130] and structure-aware NAS approaches seek architectures robust to perturbations in topology or features by incorporating noise injection, denoising, or structure-refinement mechanisms; (ii)Distributional and Environmental invariance, encompassing GRACES [129], OMG-NAS [18], DCGAS [183], GASIM [208], CARNAS [97], and related OOD-focused methods that modify search objectives or search spaces to avoid spurious correlations and improve generalization across distribution shifts, sampling bias, temporal drift, or latent causal factors; (iii)Resource invariance, including scalable and hardware-aware approaches such as GAUSS [57], PasCa [200], G-CoS [201], and FGNAS [109],

which emphasize architectures whose performance remains stable across varying compute budgets, memory constraints, or accelerator characteristics; and (iv)Modality and heterogeneity invariance, where works such as PMMM [79], DHGAS [209], AutoGT [207], and AutoGFM [24] design type-aware or modality-specific modules to ensure robustness across heterogeneous information networks and multimodal graph settings. This invariance-oriented view provides a coherent framework that links search space design, training objectives, and benchmark construction, highlighting the need for future evaluations that incorporate robustness, OOD generalization, and resource-aware testbeds aligned with these invariance principles.

4.4.10 Supported GNAS methods in AutoGL. The preceding sections from 4.4.1 to 4.4.9 introduce recent featured works, detailing the latest advancements in Automated Graph Machine Learning, particularly in GNAS. These advances cover crucial areas like robustness, generalization, scalability, heterogeneous data handling, and Graph Transformers. Table 3 below serves as a reference for users and potential contributors by mapping recent featured works to their current support status within the AutoGL library. It provides a direct assessment of AutoGL's coverage of automated graph machine learning, indicating which advanced concepts are fully integrated and which are not.

Table 3. Support status of featured NAS algorithms in AutoGL.

Status	Works
Supported	Random [96]; RL [222]; Evolution [106]; DARTS [104]; SPOS [60]; GraphNAS [48]; ENAS [125]; GASSO [130]; GAUSS [57]; AutoGT [207]; GRACES [129]; G-RNA [171]
Unsupported	DCGAS [183]; GASIM [208]; CARNAS [97]; MTGC3 [128]; BiGNAS [52]; G-Cos [201]; FGNAS [109]

4.5 Discussions

In this section, we will discuss other unique NAS designs for graphs in terms of search space, transferability and scalability.

4.5.1 The search space. Besides the basic search space presented in Section 4.1, different graph tasks may require other search spaces. For example, meta-paths are critical for heterogeneous graphs [34], edge features are essential in modeling molecular graphs [70] and many graph tasks [20, 164], and spatial-temporal modules are needed in skeleton-based recognition [123] and traffic forecasting [121]. A suitable search space usually requires careful designs and domain knowledge.

4.5.2 Transferability. It is non-trivial to transfer GNN architectures across different datasets and tasks due to the complexity and diversity of graph tasks. GraphGym [190] propose to adopt a fixed set of GNNs as anchors and rank the performance of these GNNs on different tasks and datasets. Then, the rank correlation serves as a metric to measure the similarities between different datasets and tasks. The best-performing GNNs of the most similar tasks are transferred to solve the target tasks.

4.5.3 The scalability challenge of large-scale graphs. Similar to AutoNE introduced in Section 3, EGAN [210] proposes to sample small graphs as proxies and conduct NAS on the sampled subgraphs to improve the efficiency of NAS. While achieving some progress, more advanced and principle approaches are further needed to handle billion-scale graphs.

5 Libraries for Automated Graph Machine Learning

Although there have been quite a few libraries for both graph machine learning and automated machine learning, there is no but one library for automated graph machine learning. Therefore, we will briefly overview libraries for graph machine learning and automated machine learning, followed by the in-depth introduction of the world's first dedicated open-source automated graph machine learning library, AutoGL.

5.1 Libraries for Graph Machine Learning and Automated Machine Learning

Publicly available libraries are important to facilitate and advance the research and applications of AutoML on graphs. First, we briefly list libraries for graph machine learning and automated machine learning, respectively.

Graph Machine Learning Libraries Popular libraries for graph machine learning include PyTorch Geometric [45], Deep Graph Library [155], GraphNets [6], AliGraph [218], Euler [2], PBG [78], PGL [118], TF-GNN [142], Stellar Graph [30], Spektral [56], CodDL [77], OpenNE [4], OpenHGNN [3], GEM [54], Karateclub [135] and classical NetworkX [62]. However, these libraries do not support AutoML.

Automated Machine Learning Libraries On the other hand, AutoML libraries such as NNI [199], AutoKeras [72], AutoSklearn [44], Hyperopt [9], TPOT [120], AutoGluon [39], MLBox [31], and MLJAR [115] are widely adopted. Unfortunately, because of the uniqueness and complexity of graph tasks, they cannot be directly applied to automate graph machine learning.

Despite their successes, integrating these libraries to fully support automated graph machine learning is non-trivial. This motivates us to design a specific library tailored for automated graph machine learning.

5.2 AutoGL: A Library for Automated Graph Machine Learning

We present Automated Graph Learning (AutoGL)[2], covering hyper-parameter optimization (HPO) and neural architecture search (NAS) for graph machine learning. The overall framework of AutoGL is shown in Figure 1. We summarize and abstract the pipeline of AutoML on graphs into five modules: auto feature engineering, neural architecture search, model training, hyper-parameter optimization, and auto ensemble. For each module, we provide plenty of state-of-the-art algorithms, standardized base classes, and high-level APIs for easy and flexible customization. The AutoGL library is built upon PyTorch Geometric (PyG) [45], a widely adopted graph machine learning library. AutoGL has the following key characteristics:

- **Open source:** The code[3] and detailed documentation[4] are available online.
- **Easy to use:** AutoGL is designed to be user-friendly. Users can conduct quick AutoGL experiments with less than ten lines of code.
- **Flexible to be extended:** The modular design, high-level base class APIs, and extensive documentation of AutoGL allow flexible and easy customized extensions.

5.3 Detailed Designs of AutoGL

In this section, we introduce AutoGL designs in detail. AutoGL is designed in a modular and object-oriented fashion to enable clear logic flows, easy usages, and flexible extensions. All the APIs exposed to users are abstracted in a high-level fashion to avoid redundant re-implementation

[2]This manuscript is based on AutoGL v0.2.0-pre released on 11st, July 2021. Please visit the website for the most up-to-date version.

[3]https://github.com/THUMNLab/AutoGL/

[4]https://autogl.readthedocs.io/

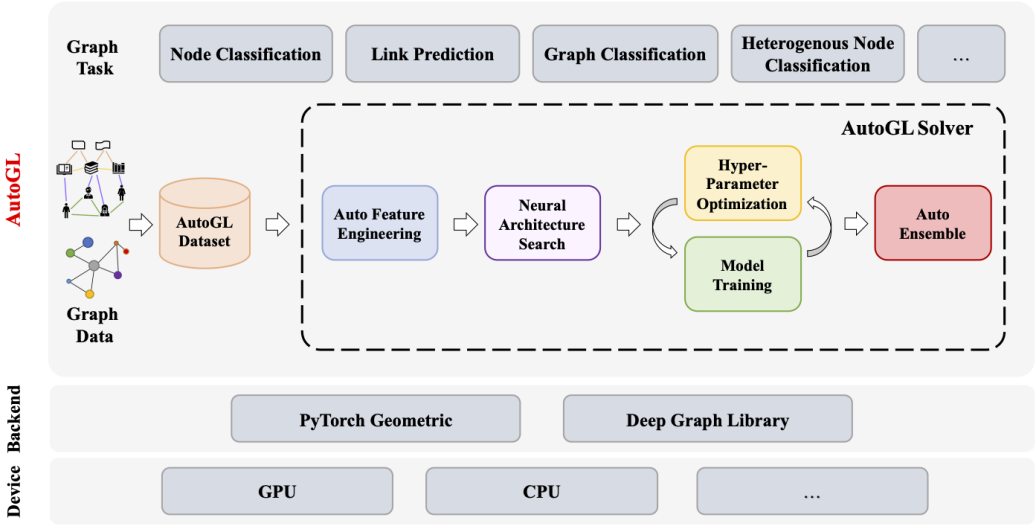


Fig. 1. The overall framework of AutoGL.

of models, algorithms, and train/evaluation protocols. All the five main modules, i.e., auto feature engineering, neural architecture search, model training, hyper-parameter optimization and auto ensemble, have taken into account the unique characteristics of graph machine learning. Next, we elaborate on the detailed designs for each module.

5.3.1 AutoGL Dataset. We first briefly introduce our dataset management. AutoGL Dataset is currently based on `Dataset` from PyTorch Geometric and supports common benchmarks for node and graph classification, including the recent Open Graph Benchmark [66]. We present the complete list of datasets in Table 4, and users can also easily customize datasets following our documentation.

Specifically, we provide widely adopted node classification datasets including Cora, CiteSeer, PubMed [136], Amazon Computers, Amazon Photo, Coauthor CS, Coauthor Physics [137], Reddit [63], and graph classification datasets such as MUTAG [32], PROTEINS [11], IMDB-B, IMDB-M, COLLAB [177], etc. Datasets from Open Graph Benchmark [66] are also supported.

5.3.2 Auto Feature Engineering. The graph data is first processed by the auto feature engineering module, where various nodes, edges, and graph-level features can be automatically added, compressed, or deleted to help boost the graph learning process after. Graph topological features can also be extracted to utilize graph structures better.

Currently, we support 24 feature engineering operations abstracted into three categories: generators, selectors, and graph features. The generators aim to create new node and edge features based on the current node features and graph structures. The selectors automatically filter out and compress features to ensure they are compact and informative. Graph features focus on generating graph-level features.

We summarize the supported generators in Table 5, including Graphlets [114], EigenGNN [204], PageRank [12], local degree profile, normalization, one-hot degrees, and one-hot node IDs. For selectors, GBDT [74] and FilterConstant are supported. An automated feature engineering method DeepGL [134] is also supported, functioning as both a generator and a selector. For graph feature,

Table 4. The statistics of the supported datasets. For datasets with more than one graph, #Nodes and #Edges are the average numbers of all the graphs. #Features correspond to node features by default, and edge features are specified.

Dataset	Task	#Graphs	#Nodes	#Edges	#Features	#Classes
Cora	Node	1	2,708	5,429	1,433	7
CiteSeer	Node	1	3,327	4,732	3,703	6
PubMed	Node	1	19,717	44,338	500	3
Reddit	Node	1	232,965	11,606,919	602	41
Amazon Computers	Node	1	13,381	245,778	767	10
Amazon Photo	Node	1	7,487	119,043	745	8
Coauthor CS	Node	1	18,333	81,894	6,805	15
Coauthor Physics	Node	1	34,493	247,962	8,415	5
ogbn-products	Node	1	2,449,029	61,859,140	100	47
ogbn-proteins	Node	1	132,534	39,561,252	8(edge)	112
ogbn-arxiv	Node	1	169,343	1,166,243	128	40
ogbn-papers100M	Node	1	111,059,956	1,615,685,872	128	172
Mutag	Graph	188	17.9	19.8	7s	2
PTC	Graph	344	14.3	14.7	18	2
ENZYMES	Graph	600	32.6	62.1	3	6
PROTEINS	Graph	1,113	39.1	72.8	3	2
NCI1	Graph	4,110	29.8	32.3	37	2
COLLAB	Graph	5,000	74.5	2,457.8	-	3
IMDB-B	Graph	1,000	19.8	96.5	-	2
IMDB-M	Graph	1,500	13.0	65.9	-	3
REDDIT-B	Graph	2,000	429.6	497.8	-	2
REDDIT-MULTI5K	Graph	5,000	508.5	594.9	-	5
REDDIT-MULTI12K	Graph	11,929	391.4	456.9	-	11
ogbg-molhiv	Graph	41,127	25.5	27.5	9, 3(edge)	2
ogbg-molpcba	Graph	437,929	26.0	28.1	9, 3(edge)	128
ogbg-ppa	Graph	158,100	243.4	2,266.1	7(edge)	37

Netlsd [149] and a set of graph feature extractors implemented in NetworkX [62] are wrapped, e.g., `NxTransitivity`, `NxAverageClustering`, etc.

We also provide convenient wrappers that support feature engineering operations in PyTorch Geometric [45] and NetworkX [62]. Users can easily customize feature engineering methods by inheriting from the class `BaseGenerator`, `BaseSelector`, and `BaseGraph`, or `BaseFeatureEngineer` if the methods do not fit in our categorization.

5.3.3 Neural Architecture Search. In AutoGL, Neural Architecture Search (NAS) aims to automate the construction of Graph Neural Networks. The best GNN model will be searched using various NAS methods to fit the current datasets. In Neural Architecture Search module, Algorithm, GNNSpace, and Estimator submodule is developed to further solve the search problem. GNNSpace defines the whole search range where we explore the best models. Algorithms are used to determine which architectures should be evaluated next, and the Estimators are used for deriving the performances of target architectures.

Table 5. Supported generators in the auto feature engineering module.

Name	Description
`graphlet`	Local graphlet numbers
`eigen`	EigenGNN features.
`pagerank`	PageRank scores.
`PYGLocalDegreeProfile`	Local Degree Profile features
`PYGNormalizeFeatures`	Row-normalize all node features
`PYGOneHotDegree`	One-hot encoding of node degrees.
`onehot`	One-hot encoding of node IDs

We have supported various GNAS models, including algorithms specified for graph data like GASSO [130] and G-RNA [171], as well as general NAS algorithms such as Random Search [96], reinforcement learning (RL) [222], and evolutionary algorithm (EA) [106]. Users can easily customize GNAS algorithms by inheriting from the `BaseNAS` class.

5.3.4 Model Training. This module handles the training and evaluation process of graph machine learning tasks with two functional sub-modules: Model and Trainer. Model handles the construction of graph machine learning models, e.g., GNNs, by defining learnable parameters and the forward pass. Trainer controls the optimization process for the given model. Common optimization methods are packaged as high-level APIs to provide neat and clean interfaces. More advanced training controls and regularization methods in graph tasks like early stopping and weight decay are also supported.

The model training module supports both node-level and graph-level tasks, e.g., node classification and graph classification. Commonly used models for node classification such as GCN [76], GAT [151], and GraphSAGE [63], GIN [176], and pooling methods such as Top-K Pooling [47] are supported. Users can quickly implement their own graph models by inheriting from the `BaseModel` class and add customized tasks or optimization methods by inheriting from `BaseTrainer`.

5.3.5 Hyper-Parameter Optimization. The Hyper-Parameter Optimization (HPO) module aims to automatically search for the best hyper-parameters of a specified model and training process, including but not limited to architecture hyper-parameters such as the number of layers, the dimensionality of node representations, the dropout rate, the activation function, and training hyper-parameters such as the learning rate, the weight decay, the number of epochs. The hyper-parameters, their types (e.g., integer, numerical, or categorical), and feasible ranges can be easily set.

We have supported various HPO algorithms, including algorithms specified for graph data like AutoNE [150] and AutoGR [158] and general-purpose algorithms like random search [8], Tree Parzen Estimator [7], etc. Users can customize HPO algorithms by inheriting from the `BaseHPOptimizer` class.

5.3.6 Auto Ensemble. This module can automatically integrate the optimized individual models to form a more powerful final model. Currently, we have adopted two kinds of ensemble methods: voting and stacking. Voting is a simple yet powerful ensemble method that directly averages the output of individual models. Stacking trains another meta-model to combine the output of models. We have supported general linear models (GLM) and gradient boosting machines (GBM) as meta-models.

5.3.7 AutoGL Solver. On top of the modules mentioned above, we provide another high-level API Solver to control the overall pipeline. In Solver, the five modules are integrated systematically to form the final model. Solver receives the feature engineering module, a model list, the HPO module,

and the ensemble module as initialization arguments to build an Auto Graph Learning pipeline. Given a dataset and a task, Solver first perform auto feature engineering to clean and augment the input data, then optimize all the given models using the model training and HPO module. At last, the optimized best models will be combined by the Auto Ensemble module to form the final model.

Solver also provides global controls of the AutoGL pipeline. For example, the time budget can be explicitly set to restrict the maximum time cost, and the training/evaluation protocols can be selected from plain dataset splits or cross-validation.

5.4 Evaluation of AutoGL

In this section, we provide experimental results. Note that we mainly want to showcase the usage of AutoGL and its main functional modules rather than aiming to achieve the new state-of-the-art on benchmarks or compare different algorithms. For node classification, we use Cora, CiteSeer, and PubMed with the standard dataset splits from [76], and additionally include the large-scale ogbn-arXiv dataset to evaluate scalability. For link prediction, we use the same node-level datasets with a 60/20/20 split to further demonstrate the applicability of AutoGL to edge-level tasks. For graph classification, we follow the setting in [40] and report the average accuracy of 10-fold cross-validation on MUTAG, PROTEINS, and IMDB-B.

5.4.1 AutoGL Results. We turn on all the functional modules in AutoGL except NAS, and report the fully automated results on node classification and link prediction in Table 6 and Table 7. To comprehensively validate the effectiveness and adaptability of the AutoGL pipeline, our comparison baselines cover classic GNN models: GCN [76], GAT [151], and GraphSAGE [63], as well as two representative advanced GNN variants: GATv2 [13] and GraphConv [116]. For graph classification tasks (results in Table 8), we use the best single model under the cross-validation setting. We observe that in all the benchmark datasets, AutoGL achieves better results than other models, demonstrating the importance of AutoML on graphs and the effectiveness of the proposed pipeline in the released library.

Table 6. The results of node classification (Accuracy).

Model	Cora	CiteSeer	PubMed	ogbn-arXiv
GCN	80.9 ± 0.7	70.9 ± 0.7	78.7 ± 0.6	52.7 ± 0.5
GAT	82.3 ± 0.7	71.9 ± 0.6	77.9 ± 0.4	53.5 ± 0.7
GraphSAGE	74.5 ± 1.8	67.2 ± 0.9	76.8 ± 0.6	53.8 ± 1.3
GATv2	82.2 ± 0.7	68.9 ± 1.3	78.0 ± 0.6	53.6 ± 0.3
GraphConv	80.0 ± 0.7	66.7 ± 0.6	78.1 ± 0.6	47.5 ± 1.3
AutoGL	**83.2 ± 0.6**	**72.4 ± 0.6**	**79.3 ± 0.4**	**54.5 ± 0.4**

5.4.2 Hyper-Parameter Optimization. Table 9 reports the results of two implemented HPO methods, i.e., random search and TPE [7], for the semi-supervised node classification task. As shown in the table, as the number of trials increases, both HPO methods tend to achieve better results. Besides, both methods outperform vanilla models without HPO. Note that a larger number of trials do not guarantee better results because of the potential overfitting problem. We further test these HPO methods with ten trials for the graph classification task and report the results in Table 10. The results generally show improvements over the default hand-picked parameters on all datasets.

Table 7. The results of link prediction (AUC).

Model	Cora	CiteSeer	PubMed	ogbn-arXiv
GCN	93.9 ± 0.7	93.9 ± 1.1	93.4 ± 2.2	94.7 ± 0.7
GAT	92.9 ± 0.7	94.4 ± 0.6	90.3 ± 1.5	92.4 ± 1.5
GraphSAGE	91.1 ± 1.2	90.3 ± 2.3	89.3 ± 4.7	92.4 ± 1.9
GATv2	92.3 ± 0.5	91.4 ± 0.8	94.0 ± 0.7	92.3 ± 1.4
GraphConv	82.3 ± 1.6	82.4 ± 1.6	91.7 ± 2.4	88.4 ± 1.1
AutoGL	**94.2 ± 0.3**	**94.8 ± 0.8**	**97.5 ± 0.1**	**96.4 ± 0.4**

Table 8. The results of graph classification (Accuracy).

Model	MUTAG	PROTEINS	IMDB-B
Top-K Pooling	80.8 ± 7.1	69.5 ± 4.4	71.0 ± 5.5
GIN	82.7 ± 6.9	66.5 ± 3.9	69.1 ± 3.7
AutoGL	**87.6 ± 6.0**	**73.3 ± 4.4**	**72.1 ± 5.0**

Table 9. The results of different HPO methods for node classification (Accuracy).

		Cora		CiteSeer		PubMed	
Method	Trials	GCN	GAT	GCN	GAT	GCN	GAT
None		80.9 ± 0.7	82.3 ± 0.7	70.9 ± 0.7	71.9 ± 0.6	78.7 ± 0.6	77.9 ± 0.4
random	1	81.0 ± 0.6	81.4 ± 1.1	70.4 ± 0.7	70.1 ± 1.1	78.3 ± 0.8	76.9 ± 0.8
	10	82.0 ± 0.6	82.5 ± 0.7	71.5 ± 0.6	**72.2 ± 0.7**	79.1 ± 0.3	78.2 ± 0.3
	50	81.8 ± 1.1	**83.2 ± 0.7**	71.1 ± 1.0	72.1 ± 1.0	**79.2 ± 0.4**	78.2 ± 0.4
TPE	1	81.8 ± 0.6	81.9 ± 1.0	70.1 ± 1.2	71.0 ± 1.2	78.7 ± 0.6	77.7 ± 0.6
	10	82.0 ± 0.7	82.3 ± 1.2	71.2 ± 0.6	72.1 ± 0.7	79.0 ± 0.4	**78.3 ± 0.4**
	50	**82.1 ± 1.0**	83.2 ± 0.8	**72.4 ± 0.6**	71.6 ± 0.8	79.1 ± 0.6	78.1 ± 0.4

Table 10. The results of different HPO methods for graph classification (Accuracy).

HPO	MUTAG		PROTEINS		IMDB-B	
	Top-K Pooling	GIN	Top-K Pooling	GIN	Top-K Pooling	GIN
None	76.3 ± 7.5	82.7 ± 6.9	69.5 ± 4.4	66.5 ± 3.9	71.0 ± 5.5	69.1 ± 3.7
random	82.7 ± 6.8	**87.6 ± 6.0**	**73.3 ± 4.4**	**71.0 ± 5.9**	71.5 ± 4.1	**71.3 ± 4.0**
TPE	**83.9 ± 10.1**	86.7 ± 6.2	72.3 ± 5.5	71.0 ± 7.2	**71.6 ± 2.5**	70.2 ± 3.7

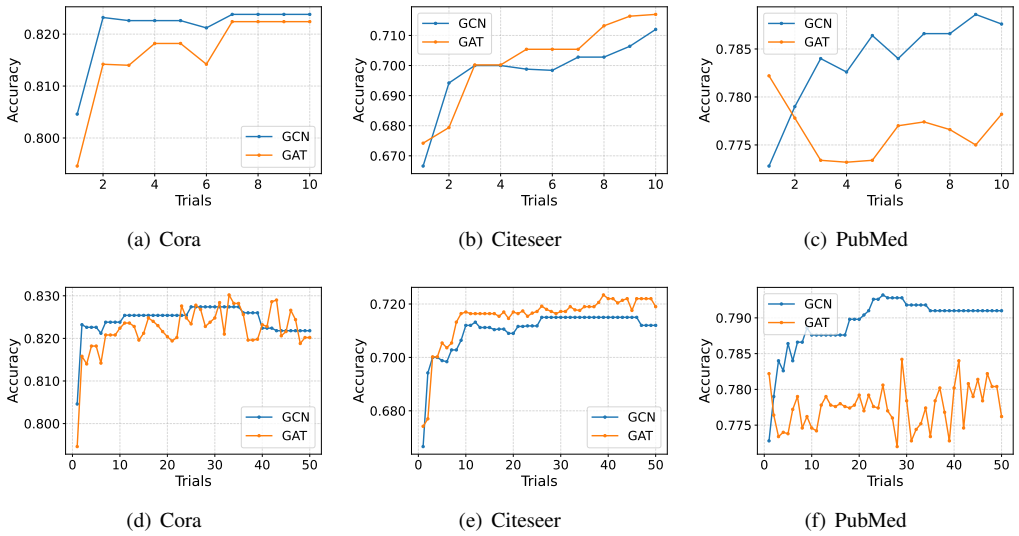
5.4.3 Training Time and Memory Usage. The corresponding training time and memory usage for node classification and link prediction are summarized in Table 11 and 12, indicating that the automated pipeline introduces moderate but manageable computational overhead for most datasets, while link prediction on large graphs like ogbn-arXiv shows more noticeable time and memory growth. To address scalability challenges and assist practical usage, we provide key guidance for users: set computational budgets (e.g., limit HPO trial number to 10–20 for large graphs), enable early stopping based on validation loss (patience=5–10 epochs) to avoid redundant training, and leverage early stopping within HPO iterations to terminate underperforming trials early. This mitigates overfitting risks and reduces unnecessary computation. These practical recipes balance performance and efficiency, enhancing AutoGL's applicability to large-scale graph tasks.

Table 11. Training time (seconds) and memory usage (MB) for node classification.

Model	Cora	CiteSeer	PubMed	ogbn-arXiv
Time (s)				
GCN	11.8	10.5	13.9	128.5
GAT	17.6	14.7	21.4	507.8
GraphSAGE	9.6	9.8	12.4	139.9
AutoGL	13.6	13.4	14.5	140.9
Memory (MB)				
GCN	77.3	177.4	249.9	1728.5
GAT	295.6	364.6	1761.0	24112.1
GraphSAGE	152.2	390.7	381.0	1296.2
AutoGL	338.8	418.7	2095.3	24845.8

Table 12. Training time (seconds) and memory usage (MB) for link prediction.

Model	Cora	CiteSeer	PubMed	ogbn-arXiv
Time (s)				
GCN	13.6	13.6	26.2	1624.1
GAT	15.2	15.0	33.1	840.5
GraphSAGE	12.7	11.0	43.1	1475.5
AutoGL	40.1	41.0	167.9	3504.7
Memory (MB)				
GCN	68.8	106.5	661.0	32307.0
GAT	236.1	395.6	2668.7	55602.2
GraphSAGE	118.9	279.9	713.1	31791.7
AutoGL	393.3	467.6	3048.2	75224.1



(a) Cora (b) Citeseer (c) PubMed

(d) Cora (e) Citeseer (f) PubMed

Fig. 2. Time-to-accuracy curves for TPE under low-budget (top row: 10 trials) and high-budget (bottom row: 50 trials) settings. Results are averaged over 5 runs.

5.4.4 Low-Budget vs High-Budget Regimes. To examine how different search budgets influence the effectiveness of hyper-parameter optimization, we evaluate TPE under varying trial budgets. Fig 2 illustrates the time-to-accuracy curves of TPE under two different trial budgets: a low-budget setting with 10 trials and a high-budget setting with 50 trials. We evaluate both GCN and GAT on Cora, CiteSeer, and PubMed, and report the accuracy achieved at each trial. As shown in the figure, increasing the trial budget generally leads to more stable performance and enables TPE to explore a broader hyper-parameter configuration space. The low-budget setting improves accuracy rapidly within the first few trials, while the high-budget setting provides further refinement and smoother convergence as the search progresses.

5.4.5 Early-Stopping Strategies. To study the effect of early-stopping on the hyper-parameter optimization process, we evaluate both random search and TPE with and without early-stopping. Table 13 reports the corresponding node classification results. As shown in the table, introducing early-stopping leads to only minor performance differences across most settings, with small improvements in some cases and comparable results in others. Although the accuracy changes are limited, early-stopping reduces the training cost of each trial by preventing unnecessary epochs once the validation performance plateaus.

5.4.6 Sensitivity to Search Space Design. To examine how the choice of hyper-parameter search space influences the behavior of hyper-parameter optimization, we evaluate random search under two configurations: a normal range and a wider range for weight decay and dropout. Table 14 reports the results under the two search space settings. As shown in the table, the performance differences between the two configurations are generally small, with the wide range only causing slight variations in some cases. The results indicate that random search is relatively robust to moderate changes in search space design within AutoGL.

Table 13. The results of node classification with/without early-stopping strategies (Accuracy).

Method	ES	Cora		CiteSeer		PubMed	
		GCN	GAT	GCN	GAT	GCN	GAT
random	wo/ES	81.8 ± 0.9	82.3 ± 0.7	71.3 ± 0.6	70.6 ± 0.5	79.0 ± 0.3	77.9 ± 0.9
	w/ES	81.9 ± 1.1	81.7 ± 1.0	70.9 ± 1.1	71.9 ± 0.6	79.0 ± 0.6	77.8 ± 0.8
TPE	wo/ES	81.1 ± 1.1	82.1 ± 1.2	70.9 ± 0.7	71.3 ± 1.1	78.9 ± 0.4	78.0 ± 0.6
	w/ES	82.3 ± 0.6	82.4 ± 1.0	70.3 ± 1.4	71.3 ± 0.7	78.8 ± 0.5	77.7 ± 0.7

Table 14. The results of node classification under different hyper-parameter search space ranges (normal vs wide) using random search. Normal range: weight decay $[10^{-4}, 10^{-2}]$, dropout $[0.4, 0.6]$; Wide range: weight decay $[10^{-6}, 10^{-1}]$, dropout $[0.2, 0.8]$.

Search Space	Cora		CiteSeer		PubMed	
	GCN	GAT	GCN	GAT	GCN	GAT
Normal	81.2 ± 1.0	81.7 ± 1.1	71.6 ± 1.1	72.1 ± 1.0	78.9 ± 0.4	78.2 ± 0.6
Wide	81.5 ± 1.1	82.0 ± 0.9	70.6 ± 1.3	71.7 ± 1.0	78.7 ± 0.8	78.1 ± 0.5

5.4.7 Transferability of HPO. To examine whether HPO methods tuned on one dataset can be effectively transferred to another, we evaluate two HPO strategies using random search: Local, which performs hyper-parameter optimization directly on Amazon-Computers, and Transferred, which uses hyper-parameters obtained from Coauthor-CS. Table 15 summarizes the results on Amazon-Computers. As shown in the table, the transferred hyperparameters lead to noticeably lower accuracy, suggesting that the optimal configurations identified on Coauthor-CS do not generalize well to Amazon-Computers. This result highlights the dataset-specific nature of hyperparameter choices in graph learning tasks.

Table 15. The results of node classification on Amazon-Computers using local and transferred hyper-parameter optimization with random search (Accuracy).

HPO Strategy	GCN	GAT
Local	76.6 ± 3.9	73.0 ± 3.5
Transferred	66.6 ± 14.2	64.4 ± 11.7

5.4.8 Auto Ensemble. Table 16 reports the performance of the ensemble module as well as its base learners for the node classification task. We use voting as the example ensemble method and choose GCN and GAT as the base learners. The table shows that the ensemble module achieves better performance than both the base learners, demonstrating the effectiveness of the implemented ensemble module.

Table 16. The performance of the ensemble module of AutoGL for the node classification task.

Base Model	Cora	CiteSeer	PubMed
GCN	81.1 ± 0.9	69.6 ± 1.1	**78.5 ± 0.4**
GAT	82.0 ± 0.5	70.4 ± 0.6	77.7 ± 0.5
Ensemble	**82.2 ± 0.4**	**70.8 ± 0.5**	**78.5 ± 0.4**

Table 17. Ablation study results on node classification.

Ablation	Cora	CiteSeer	PubMed	ogbn-arXiv
w/o ensemble	82.1 ± 0.6	71.5 ± 1.0	78.6 ± 0.6	53.9 ± 0.7
w/o feature	81.9 ± 0.9	71.5 ± 0.6	78.8 ± 0.4	54.5 ± 0.4
w/o hpo	77.1 ± 3.4	68.4 ± 1.3	72.5 ± 1.2	22.3 ± 2.8
AutoGL	83.2 ± 0.6	72.4 ± 0.6	79.3 ± 0.4	54.5 ± 0.4

5.4.9 Ablation Study. Table 17 reports the ablation study metrics on four modules for AutoGL. The results show that removing the ensemble or feature engineering modules leads to only minor performance drops across all datasets, indicating that these components provide incremental but not decisive improvements. In contrast, removing the hyperparameter optimization module causes a substantial degradation in accuracy, especially on ogbn-arXiv, where performance drops by more than 30 percent. This highlights that hyperparameter optimization serves as the primary contributor to AutoGL's performance gains. This observation does not reduce the value of AutoGL's end-to-end pipeline: the modest impact of feature engineering and ensembling may stem from the high-quality raw features of the benchmark datasets and the strong performance of the specified base models (reducing potential gains from ensemble diversity), not indicative of inadequate module design or misconfiguration.These modules serve as robust supplements, stabilizing training and adapting to diverse data scenarios. The full AutoGL system, which integrates all modules except NAS, consistently achieves the best results, confirming that the framework benefits from the combined effect of its automated components.

5.5 Advanced Functions of AutoGL

We have presented AutoGL, the first library for automated graph machine learning, which is open-source, easy to use, and flexible to be extended. Currently, we are actively developing AutoGL and have supported the following advanced functionalities:

- Support more graph models. We have supported self-supervised graph models and robust graph models now.
- Handle more graph tasks. We have supported heterogeneous node classification tasks now and plan to support more complex spatial-temporal graphs.
- Support more graph library backends. We have supported Deep Graph Library (DGL) [155] backend including homogeneous node classification, link prediction, and graph classification tasks. AutoGL is also compatible with PyG 2.0 [45] now.

All kinds of inputs and suggestions are also warmly welcomed.

6 Benchmark

To promote the further development of automated graph machine learning, NAS-Bench-Graph [131] is proposed as one benchmark tailored to enable unified, reproducible, and efficient evaluations for graph NAS. Two key issues are identified: i) the lack of a standard experimental setting, making comparisons across research unreliable and non-reproducible, and ii) the computational inefficiency of graph NAS methods. To resolve these issues, NAS-Bench-Graph constructs a unified search space, encompassing 26,206 unique GNN architectures, and offers a standardized evaluation protocol. All architectures have been trained and evaluated on nine representative graph datasets, with metrics such as training, validation, and test performance, latency, and the number of parameters recorded. This results in a look-up table that allows for fair and efficient comparisons without additional computational costs. The authors also showcase the benchmark's compatibility with existing GNAS libraries like AutoGL and NNI. The work claims to be the first of its kind to offer a benchmark in the domain of GNAS.

6.1 Benchmark Design

Here we elucidates the methodology employed in the development of our benchmark for graph NAS, encapsulating search space design (Section 6.1.1), utilized datasets (Section 6.1.2), and experimental configurations (Section 6.1.3).

6.1.1 Search Space Design. To achieve a judicious equilibrium between effectiveness and efficiency, we propose an intricately crafted, yet computationally tractable search space. In particular, we conceptualize the macro search space governing Graph Neural Network (GNN) architectures as a Directed Acyclic Graph (DAG), thereby providing a formal structure for the computational paradigm[5]. In this DAG, each computational node signifies a vertex representation, while each edge symbolizes a specific operation. The DAG is composed of six nodes, including the input and output nodes. Notably, each intermediate node is restricted to having a single incoming edge. As a result, the aforementioned DAG encompasses nine selectable configurations, as delineated in Figure 3. Any intermediate nodes devoid of successor nodes are concatenated to the output node.

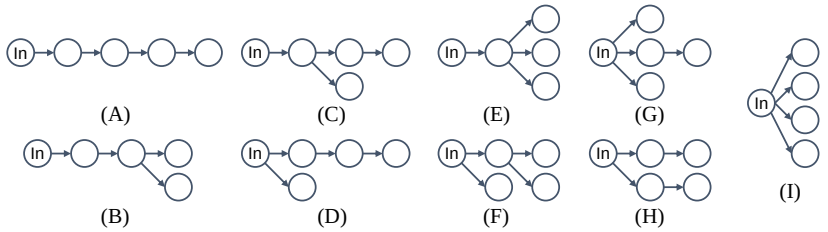


Fig. 3. Nine different choices of our macro search space. Each node is a representation of vertices and each edge is an operation [131].

In addition to this overarching macro space, we also incorporate optional fully-connected layers in the pre-processing and post-processing layers, following the design principles established in GraphGym [191] and PasCa [200]. To circumvent an exponential increase in the complexity of the search space, the number of layers in both pre-processing and post-processing phases are treated as hyperparameters, a point that will be elaborated upon in Section 6.1.3. To culminate the architecture, a task-specific fully-connected layer is employed to generate the model's prediction.

[5]For disambiguation, we refer to nodes and edges in the computational graph, as opposed to vertices and links that constitute the graph data.

Table 18. The hyper-parameters and hardware used for each dataset. #Pre and #Post denotes the number of pre-process and post-process layers, respectively.

Dataset	#Pre	#Post	Dimension	Dropout	Optimizer	LR	WD	# Epoch
Cora	0	1	256	0.7	SGD	0.1	0.0005	400
CiteSeer	0	1	256	0.7	SGD	0.2	0.0005	400
PubMed	0	0	128	0.3	SGD	0.2	0.0005	500
Coauthor-CS	1	0	128	0.6	SGD	0.5	0.0005	400
Coauthor-Physics	1	1	256	0.4	SGD	0.01	0	200
Amazon-Photo	1	0	128	0.7	Adam	0.0002	0.0005	500
Amazon-Computers	1	1	64	0.1	Adam	0.005	0.0005	500
ogbn-arxiv	0	1	128	0.2	Adam	0.002	0	500
ogbn-proteins	1	1	256	0	Adam	0.01	0.0005	500

Table 19. The average training time of architectures on each dataset.

Dataset	Time	Dataset	Time	Dataset	Time
Cora	5.8s	Coauthor-CS	8.6s	Amazon-Computers	9.8s
CiteSeer	6.2s	Coauthor-Physics	15.4s	ogbn-arXiv	71s
PubMed	7.8s	Amazon-Photo	8.8s	ogbn-proteins	50min

For the set of admissible operations, we focus on seven GNN layers that have garnered widespread acceptance in the literature: GCN [76], GAT [151], GraphSAGE [63], GIN [176], ChebNet [33], ARMA [10], and k-GNN [116]. Furthermore, we introduce the Identity operation to facilitate residual connections and a fully-connected layer that does not exploit graph structures.

In summation, the search space we designed is remarkably expansive, consisting of 26,206 unique architectures after accounting for isomorphic structures (i.e., structures that may exhibit different topological characteristics yet perform identical functionalities, further elaborated in Appendix A.5). Importantly, this search space encapsulates a myriad of GNN variants, including but not limited to the previously mentioned methods, and extends to more advanced architectures such as JK-Net [175] and residual- and dense-like GNNs [80].

6.1.2 Datasets. In the course of this study, we employ nine diverse, publicly available datasets extensively utilized in the realm of Graph Neural Architecture Search (GNAS). Specifically, these datasets include Cora, CiteSeer, and PubMed [136], alongside Coauthor-CS, Coauthor-Physics, Amazon-Photo, and Amazon-Computer [138], as well as ogbn-arXiv and ogbn-proteins [67]. These datasets span an array of sizes, ranging from several thousands to millions of edges, and encompass diverse application domains such as citation networks, e-commerce graphs, and protein interaction networks.

Regarding dataset partitioning strategies, we adhere to the publicly available semi-supervised settings for Cora, CiteSeer, and PubMed as delineated by [180]. This involves utilizing 20 labeled nodes per class for training and 500 nodes for validation purposes. For the Amazon and Coauthor datasets, we employ a random partitioning scheme for train/validation/test splits in a semi-supervised fashion, as recommended by [138]. Specifically, each class is represented by 20 nodes for training, 30 nodes for validation, and the remaining nodes are used for testing. In the case of ogbn-arXiv and ogbn-proteins, we abide by the official dataset splits.

Pertaining to the ogbn-proteins dataset, preliminary experiments indicated that the utilization of Graph Isomorphism Network (GIN) and k-Graph Neural Network (k-GNN) operations resulted in parameter explosion, thereby yielding uninterpretable outcomes. Additionally, employing Graph Attention Networks (GAT) and Chebyshev Spectral Graph Convolution Networks (ChebNet) led to out-of-memory errors on our high-capacity GPUs equipped with 32GB of memory. To mitigate these computational inefficiencies, we circumscribed the candidate operations for ogbn-proteins to Graph Convolution Networks (GCN), Attention-based Recurrent Multi-layer Average networks (ARMA), GraphSAGE, Identity mappings, and fully connected layers. Consequently, this constraint yielded a feasible architecture space comprising 2,021 candidate models for ogbn-proteins.

6.1.3 Settings.

Hyper-parameters. For fair and reproducible comparisons, we propose a unified evaluation protocol and consider the following hyper-parameters with tailored ranges:

- Number of pre-process layers: 0 or 1.
- Number of post-process layers: 0 or 1.
- Dimensionality of hidden units: 64, 128, or 256.
- Dropout rate: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8.
- Optimizer: SGD or Adam.
- Learning Rate (LR): 0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001.
- Weight Decay: 0 or 0.0005.
- Number of training epochs: 200, 300, 400, 500.

For each dataset, we establish fixed hyper-parameters across all architectures in order to ensure a fair comparison. It should be noted that exhaustively enumerating combinations of architectures and hyper-parameters would lead to an impractical number of architecture hyper-parameter pairs in the order of billions. Hence, we adopt a two-step approach, first optimizing the hyper-parameters to a suitable value that can accommodate various Graph Neural Network (GNN) architectures, and subsequently focusing on the GNN architectures themselves. Specifically, we select 30 GNN architectures from our search space as "anchors" and employ random search for hyper-parameter optimization [8]. The set of 30 anchor architectures comprises 20 randomly chosen architectures from our search space, along with 10 classic GNN architectures including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), Graph Isomorphism Networks (GIN), GraphSAGE, and ARMA with 2 and 3 layers. We optimize the hyper-parameters by maximizing the average performance of these anchor architectures. The specific hyper-parameters selected for each dataset are shown in Table 18.

Metrics. During the training of each architecture, we record a comprehensive set of metrics covering both model effectiveness and efficiency. These metrics include the loss values and evaluation metric at each epoch for both the training, validation, and testing sets, as well as the model latency and the number of parameters. The hardware and software configurations utilized in our experiments are provided as follows:

- Operating System: Ubuntu 18.04.6 LTS for PubMed, ogbn-arXiv, and CentOS Linux release 7.6.1810 for the others.
- CPU: Intel(R) Xeon(R) Gold 6129 CPU @ 2.30GHz for PubMed, ogbn-arXiv, and Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz for the others.
- GPU: NVIDIA GeForce RTX 3090 with 24GB of memories for PubMed, ogbn-arXiv, and NVIDIA Tesla V100 with 16GB of memories for the others.
- Software: Python 3.9.12, PyTorch 1.11.0+cu113, PyTorch-Geometric 2.0.4 [45].

Moreover, to account for the inherent variability in the training process, all experiments are repeated three times using different random seeds. The average training time of architectures on each dataset is reported in Table 19. The total computational cost incurred in creating our benchmark dataset amounts to approximately 8,000 GPU hours.

6.2 Example Usages

This section demonstrates the utilization of NAS-Bench-Graph in conjunction with established open-source libraries such as AutoGL [58] and NNI [113]. Specifically, we employ two NAS algorithms, namely GNAS [49] and Auto-GNN [216], within the AutoGL framework. Additionally, we employ Random Search [96], Evolutionary Algorithm (EA), and Policy-based Reinforcement Learning (RL) algorithms within NNI. To ensure equitable comparisons, we restrict each algorithm's access to the performance data of only 2% of the total architectures within the search space. The obtained results are presented in Table 20. Furthermore, we include the performance metrics of the top 5% architectures, representing the 20-quantiles for each dataset, within the aforementioned table.

Table 20. The performance of NAS methods in AutoGL and NNI using NAS-Bench-Graph. The best performance for each dataset is marked in bold. We also show the performance of the top 5% architecture (i.e., 20-quantiles) as a reference line. The results are averaged over five experiments with different random seeds and the standard errors are shown in the bottom right.

Library	Method	Cora	CiteSeer	PubMed	CS	Physics	Photo	Computers	arXiv	proteins
AutoGL	GNAS	$82.04_{0.17}$	$\mathbf{70.89}_{0.16}$	$77.79_{0.02}$	$90.97_{0.06}$	$92.43_{0.04}$	$92.43_{0.03}$	$84.74_{0.20}$	$72.00_{0.02}$	$\mathbf{78.71}_{0.11}$
	Auto-GNN	$81.80_{0.00}$	$70.76_{0.12}$	$77.69_{0.16}$	$\mathbf{91.04}_{0.04}$	$92.42_{0.16}$	$92.38_{0.01}$	$84.53_{0.14}$	$\mathbf{72.13}_{0.03}$	$78.54_{0.30}$
NNI	Random	$82.09_{0.08}$	$70.49_{0.08}$	$77.91_{0.07}$	$90.93_{0.07}$	$92.35_{0.05}$	$\mathbf{92.44}_{0.02}$	$84.78_{0.14}$	$72.04_{0.05}$	$78.32_{0.14}$
	EA	$81.85_{0.20}$	$70.48_{0.12}$	$\mathbf{77.96}_{0.12}$	$90.60_{0.07}$	$92.22_{0.08}$	$92.43_{0.02}$	$84.29_{0.29}$	$71.91_{0.06}$	$77.93_{0.21}$
	RL	$\mathbf{82.27}_{0.21}$	$70.66_{0.12}$	$\mathbf{77.96}_{0.09}$	$90.98_{0.01}$	$\mathbf{92.48}_{0.03}$	$92.42_{0.06}$	$\mathbf{84.90}_{0.19}$	$\mathbf{72.13}_{0.05}$	$78.52_{0.18}$
The top 5%		80.63	69.07	76.60	90.01	91.67	91.57	82.77	71.69	78.37

The detailed example codes are provided as follows. All the codes and recorded metrics for the trained models are available at https://github.com/THUMNLab/NAS-Bench-Graph. Next, we provide some example usages.

At first, the benchmark of a certain dataset, e.g., Cora, can be read as:

```
from readbench import lightread
bench = lightread('cora')
```

The data is stored as a Python dictionary. In order to capture the metrics of interest, it is imperative to define an architecture by specifying its macro space and corresponding operations. In our approach, we impose a restriction on the directed acyclic graph (DAG) of the computation graph, dictating that each intermediate node may have only one input node. This constraint allows us to represent the macro space using a list of integers, wherein each integer denotes the index of the input node for a given computing node. Specifically, the value of 0 corresponds to the raw input, 1 corresponds to the first computing node, and so forth. Additionally, the operations associated with the architecture can be described using a list of strings of equal length, providing a concise representation of the architectural choices made. For example, to specify the architecture shown in Figure 4, we can use the following code:

```
from hpo import Arch
arch = Arch([0, 1, 2, 1], ['gcn', 'gin', 'fc', 'cheb'])
```

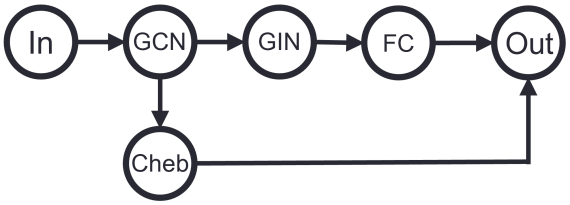


Fig. 4. An example architecture.

We assume all leaf nodes (i.e., nodes without descendants) are connected to the output, so there is no need to specific the output node. Besides, the list can be specified in any order, e.g., the following code can specific the same architecture:

```
arch = Arch([0, 1, 1, 2], ['gcn', 'cheb', 'gin', 'fc'])
```

Then, four recorded metrics in the benchmark including the validation and test performance, the latency, and the number of parameters, can be obtained by a look-up table:

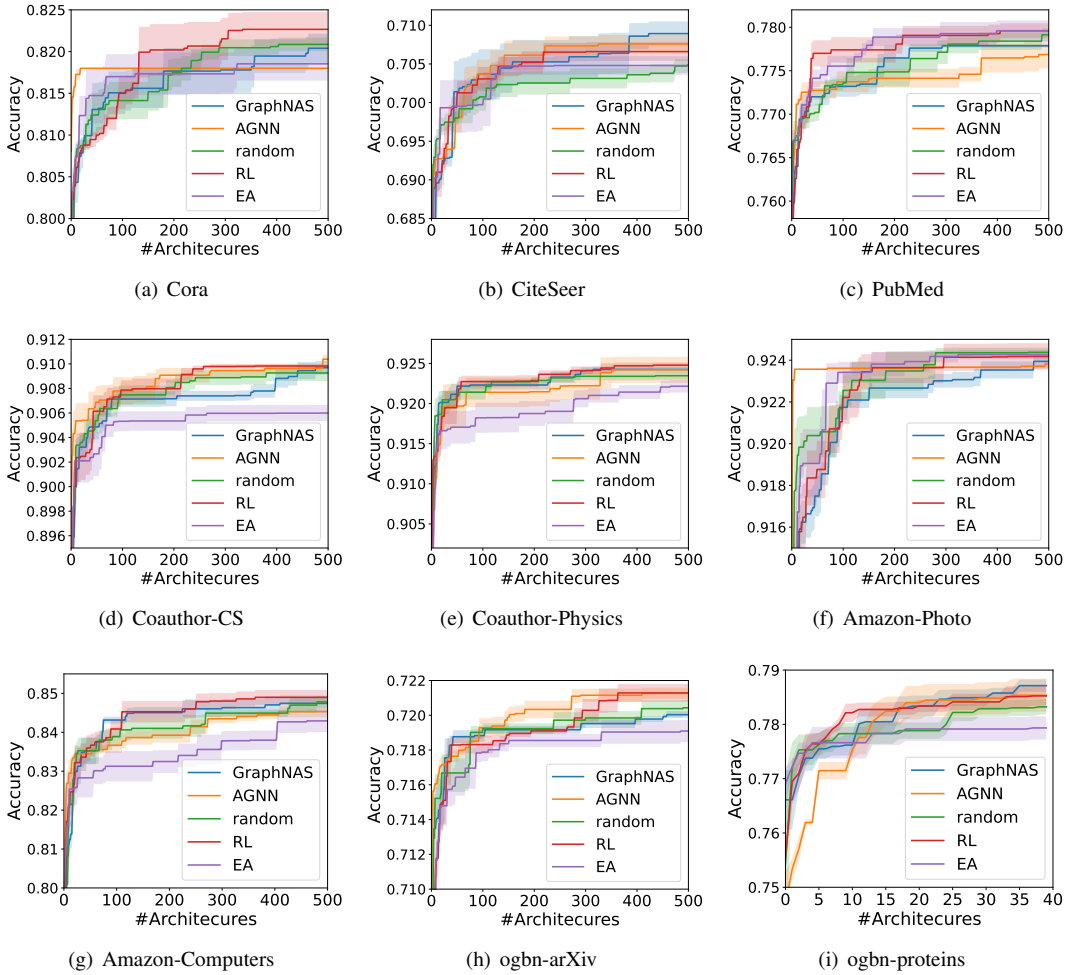
```
info = bench[arch.valid_hash()]
info['valid_perf']  # validation performance
info['perf']        # test performance
info['latency']     # latency
info['para']        # number of parameters
```

We provide the full data, including the training/validation/testing performance at each epoch at: https://figshare.com/articles/dataset/NAS-bench-Graph/20070371. Since we run each dataset with three random seeds, each dataset has 3 files. The full metric can be obtained similarly as follows:

```
from readbench import read
bench = read('cora0.bench')  # dataset and seed
info = bench[arch.valid_hash()]
epoch = 50
info['dur'][epoch][0]   # training performance
info['dur'][epoch][1]   # validation performance
info['dur'][epoch][2]   # testing performance
info['dur'][epoch][3]   # training loss
info['dur'][epoch][4]   # validation loss
info['dur'][epoch][5]   # testing loss
info['dur'][epoch][6]   # best performance
```

We have also provided the source codes of using our benchmark together with two public libraries for GraphNAS, AutoGL and NNI. See https://github.com/THUMNLab/AutoGL/tree/agnn and https://github.com/THUMNLab/NAS-Bench-Graph/blob/main/runnni.py for details.

From the experimental findings, it is evident that all algorithms exhibit superior performance compared to the top 5% benchmark, indicating their ability to acquire meaningful patterns in the NAS-Bench-Graph. Nevertheless, no algorithm demonstrates consistent superiority across all datasets. Notably, Random Search remains a robust baseline in comparison to alternative methods and even achieves the highest performance on two datasets, partially supporting the findings of Li *et al.* [96] in the context of general NAS. These results underscore the pressing need for further research in the domain of graph NAS.



(a) Cora (b) CiteSeer (c) PubMed

(d) Coauthor-CS (e) Coauthor-Physics (f) Amazon-Photo

(g) Amazon-Computers (h) ogbn-arXiv (i) ogbn-proteins

Fig. 5. The learning curve depicting the optimal performance as a function of the number of searched architectures is presented herein. The reported results are obtained by averaging measurements from five independent experiments, each conducted with distinct random seeds. The background of the figure displays the standard errors associated with the reported values.

In order to delve into the learning behaviors of diverse graph NAS methods, we present the curves depicting optimal performance as a function of the number of architectures, as illustrated in Figure 5. It is evident that different algorithms exhibit distinct characteristics. For instance, EA and AGNN demonstrate sporadic "jumps" in performance, indicating substantial performance improvements, while RL exhibits more gradual and consistent performance enhancements. A meticulous examination of these learning curves may serve as a source of inspiration for the development of novel algorithms for graph NAS.

6.3 Additional Experimental Results

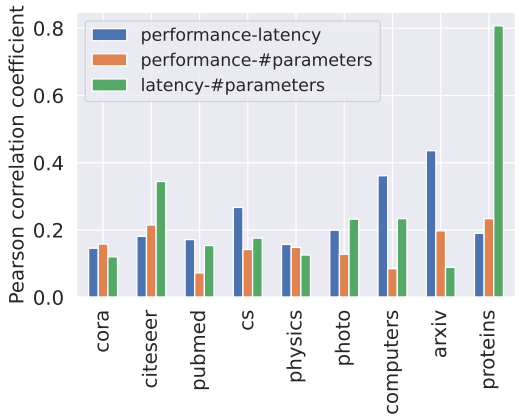


Fig. 6. Pearson correlation coefficient between pairs among performance, latency, and the number of parameters.

6.3.1 Correlations between Performance, Latency, and the Number of Parameters. We report the Pearson correlation coefficients among performance, latency, and parameter counts in Figure 6. Our analysis shows: i) parameter count and latency are generally positively correlated; ii) higher performance often comes at the cost of increased latency and larger models, highlighting the need to balance accuracy with computational overhead.

6.3.2 Sensitivity Experiment. As the 2% budget seems arbitrary, we conduct experiments to examine sensitivity by varying the search space from 1%, 5% to 10%. We report the results in Table 21. From the data presented in Table 21, we observe that the accuracy and its rankings remains relatively stable as the search space. We examine whether enlarging the search space alters the relative performance among different NAS algorithms. Across most datasets and all search budgets (1%, 5%, and 10%), the performance ordering of GNAS, Auto-GNN, Random, EA, and RL remains generally unchanged, especially among proteins, Computers, and Physics datasets. This indicates that increasing the number of queried architectures does not fundamentally shift the comparative strengths of different approaches. Consequently, the stability suggests that the NAS-Bench-Graph search space is relatively insensitive to budget scaling, indicating method comparison can already be drawn from small search budgets.

6.3.3 Discussion on NAS-Bench-Graph Limitations. We further examine two inherent limitations of NAS-Bench-Graph: (i) the potential bias introduced by fixing a single hyperparameter configuration for each dataset based on 30 anchor architectures; and (ii) the restricted macro search space that enforces single input edges, which may affect conclusions regarding transferability.

Anchor Sensitivity Analysis. NAS-Bench-Graph selects hyperparameters by optimizing the average performance of 30 randomly selected anchor architectures, which may bias the results toward architectures similar to these anchors. To examine this effect, we utilized auxiliary LUT slice experiments involving small-scale hyperparameter optimization around the top-performing architectures. Table 24 shows that sensitivity to hyperparameter selection varies across datasets. On arXiv, the rankings between LUT slice and LUT total exhibit some fluctuations—Auto-GNN and RL, the top performers in LUT total, remain at the forefront in LUT slice, but GNAS's ranking improves. On Photo, Random (the best in LUT total) drops in rank within LUT slice, while GNAS becomes the top performer. On CiteSeer, although GNAS remains the best architecture, its performance

Table 21. The performance of NAS methods in AutoGL using **1%**, **5%**, and **10%** of architectures in NAS-Bench-Graph. OOT means it fails to finish within the allotted time budget.

1% architectures										
Library	Method	Cora	CiteSeer	PubMed	CS	Physics	Photo	Computers	arXiv	proteins
AutoGL	GNAS	$81.91_{0.33}$	$70.43_{0.32}$	$77.79_{0.26}$	$90.76_{0.15}$	$92.36_{0.23}$	$\mathbf{92.40}_{0.14}$	$\mathbf{85.16}_{0.26}$	$72.04_{0.14}$	$\mathbf{78.28}_{0.30}$
	Auto-GNN	$81.83_{0.05}$	$70.52_{0.44}$	$77.46_{0.22}$	$\mathbf{90.85}_{0.11}$	$92.17_{0.47}$	$92.36_{0.00}$	$83.93_{0.55}$	$72.00_{0.07}$	$77.56_{0.45}$
NNI	Random	$81.42_{0.03}$	$70.64_{0.35}$	$\mathbf{78.16}_{0.29}$	$90.83_{0.12}$	$92.57_{0.22}$	$92.23_{0.10}$	$84.47_{0.40}$	$71.99_{0.08}$	$77.46_{0.44}$
	EA	$81.72_{0.36}$	$70.63_{0.30}$	$77.77_{0.15}$	$90.78_{0.16}$	$92.49_{0.17}$	$92.23_{0.20}$	$84.83_{0.29}$	$\mathbf{72.21}_{0.13}$	$77.74_{0.30}$
	RL	$\mathbf{81.93}_{0.10}$	$\mathbf{70.70}_{0.06}$	$78.02_{0.28}$	$90.84_{0.15}$	$\mathbf{92.58}_{0.07}$	$92.36_{0.03}$	$85.04_{0.18}$	$72.05_{0.13}$	$78.02_{0.28}$
5% architectures										
Library	Method	Cora	CiteSeer	PubMed	CS	Physics	Photo	Computers	arXiv	proteins
AutoGL	GNAS	$\mathbf{82.31}_{0.07}$	$70.91_{0.14}$	$78.19_{0.09}$	$\mathbf{91.08}_{0.05}$	$92.64_{0.18}$	$92.56_{0.07}$	$\mathbf{85.47}_{0.14}$	$72.23_{0.04}$	$\mathbf{78.73}_{0.26}$
	Auto-GNN	$82.01_{0.20}$	$\mathbf{71.06}_{0.26}$	$77.96_{0.19}$	$91.01_{0.05}$	$92.70_{0.05}$	$92.45_{0.01}$	$85.44_{0.24}$	$72.22_{0.14}$	$78.37_{0.25}$
NNI	Random	$81.96_{0.35}$	$70.90_{0.21}$	$\mathbf{78.24}_{0.22}$	$91.02_{0.11}$	$\mathbf{92.79}_{0.17}$	$\mathbf{92.62}_{0.14}$	$85.27_{0.34}$	$72.17_{0.13}$	$78.19_{0.29}$
	EA	$81.86_{0.32}$	$70.82_{0.38}$	$78.03_{0.31}$	$90.95_{0.17}$	$92.58_{0.01}$	$92.45_{0.18}$	$84.78_{0.27}$	$\mathbf{72.26}_{0.07}$	$78.34_{0.12}$
	RL	OOT	OOT	OOT	OOT	OOT	OOT	OOT	OOT	$78.34_{0.25}$
10% architectures										
Library	Method	Cora	CiteSeer	PubMed	CS	Physics	Photo	Computers	arXiv	proteins
AutoGL	GNAS	$\mathbf{82.32}_{0.07}$	$71.03_{0.14}$	$\mathbf{78.41}_{0.13}$	$\mathbf{91.13}_{0.06}$	$92.78_{0.12}$	$\mathbf{92.67}_{0.10}$	$\mathbf{85.54}_{0.00}$	$\mathbf{72.31}_{0.03}$	$\mathbf{78.75}_{0.25}$
	Auto-GNN	$82.13_{0.18}$	$71.16_{0.18}$	$78.35_{0.32}$	$91.05_{0.07}$	$92.79_{0.11}$	$92.46_{0.03}$	$85.44_{0.24}$	$72.30_{0.10}$	$78.62_{0.29}$
NNI	Random	$82.13_{0.26}$	$\mathbf{71.22}_{0.16}$	$78.37_{0.13}$	$91.09_{0.08}$	$\mathbf{92.87}_{0.12}$	$92.62_{0.12}$	$85.45_{0.14}$	$72.25_{0.05}$	$78.45_{0.19}$
	EA	$82.00_{0.30}$	$70.96_{0.32}$	$77.84_{0.33}$	$90.94_{0.05}$	$92.57_{0.15}$	$92.52_{0.14}$	$85.45_{0.18}$	$72.23_{0.14}$	$78.42_{0.23}$
	RL	OOT	OOT	OOT	OOT	OOT	OOT	OOT	OOT	$78.63_{0.32}$

decreases from 70.89% to 69.76% (a drop of approximately 1.13%), and the rankings of RL and EA also change. These results indicate that anchor bias does exist and is dataset-dependent; in certain datasets, the fixed-hyperparameter evaluation method may affect the stability of architecture rankings. Therefore, we suggest that future benchmarks consider providing auxiliary LUT slices (as we have done) to test ranking robustness.

Search Space Constraint Analysis. NAS-Bench-Graph adopts a fixed macro DAG design where each intermediate node is forced to have only one incoming edge, thereby excluding multi-branch structures. This design choice is primarily based on two considerations: (i) maintaining the tractability of the search space (26,206 architectures) to make full pre-computation possible; and (ii) empirical studies on GNNs suggesting that most successful models (such as GCN, GAT, and GraphSAGE) rely on single-path designs, as multi-branch patterns are not as prevalent in the graph domain as they are in computer vision. While it is impossible to directly evaluate the impact of multi-input edges, we can indirectly address this by analyzing the impact of search space variations on the robustness of our conclusions. We constructed an auxiliary LUT slice by reducing the number of GNN operator types from 5 to 3, shrinking the total architecture count to approximately 1/10 of the original. Table 24 shows that in this significantly narrowed search space, the rankings of different NAS methods remain largely consistent with the full LUT: the optimal methods did not change on arXiv and CiteSeer; on Photo, despite slight fluctuations, the overall ranking pattern remained similar. This suggests

Table 22. Wilcoxon signed-rank test results highlighting key comparisons. The table reports p-values and win/tie/loss counts across 9 datasets. Bold values indicate statistical significance at $\alpha = 0.05$.

Comparison Type	Comparison	P-value	Win/Tie/Loss
Same Method, Different Budgets			
Random Search	Rand 1% vs. Rand 5%	**0.0109**	0/0/9
	Rand 1% vs. Rand 10%	**0.0077**	0/0/9
	Rand 5% vs. Rand 10%	**0.0109**	0/0/9
Evolutionary Algorithm	EA 1% vs. EA 5%	0.1736	2/0/7
	EA 1% vs. EA 10%	0.0663	2/0/7
	EA 5% vs. EA 10%	0.3743	3/0/6
Different Methods, Same Budget			
1% Budget	Rand 1% vs. EA 1%	0.1109	2/0/7
	Rand 1% vs. RL 1%	**0.0143**	1/0/8
	EA 1% vs. RL 1%	0.0669	3/0/6
5% Budget	Rand 5% vs. EA 5%	**0.0266**	6/0/3
10% Budget	Rand 10% vs. EA 10%	**0.0109**	7/1/1

Note: Win/Tie/Loss counts indicate the number of datasets where the first method wins/ties/loses against the second method. OOT = Out-of-Time. RL 5% and RL 10% failed to complete on 8 out of 9 datasets, making statistical comparison infeasible. All p-values are computed using Wilcoxon signed-rank test.

Table 23. Wall-clock time (in hours) comparison across methods and budgets. OOT (Out of Time) indicates runs exceeding 24 hours.

	Random Search			Evolutionary Algorithm			Reinforcement Learning		
Dataset	1%	5%	10%	1%	5%	10%	1%	5%	10%
Cora	0.4	1.1	2.1	0.42	1.2	1.9	5.2	>24	>24
CiteSeer	0.39	1.07	2.05	0.41	1.17	1.88	5.1	>24	>24
PubMed	0.41	1.12	2.15	0.43	1.22	1.95	3.9	>24	>24
CS	0.42	1.15	2.2	0.44	1.25	2.0	4.6	>24	>24
Physics	0.41	1.13	2.16	0.43	1.23	1.96	5.3	>24	>24
Photo	0.4	1.1	2.1	0.42	1.2	1.9	4.6	>24	>24
Computers	0.41	1.12	2.14	0.43	1.22	1.94	4.2	>24	>24
arXiv	0.42	1.15	2.2	0.44	1.25	2.0	4.7	>24	>24
Proteins	0.11	0.21	0.30	0.12	0.22	0.30	0.12	0.45	1.20

Note: OOT = Out of Time (>24 hours). All datasets except Proteins are within 0.95-1.05× of Cora's size. Proteins is the smallest dataset with only 2,000 architectures (while others have ~26,000 architectures), hence its significantly lower times across all methods. RL completes all datasets at 1% budget. At 5% and 10% budgets, only Proteins completes, others OOT. Random and EA complete all runs successfully.

that even with substantial changes to the search space, the comparative conclusions regarding NAS methods on NAS-Bench-Graph remain reasonably robust.

Hardware Heterogeneity and Reproducibility.NAS-Bench-Graph utilized heterogeneous hardware/software environments (different operating systems, CPUs, and GPUs) across different datasets,

which may influence latency and even convergence behavior—this is particularly prominent on the ogbn-proteins dataset, where each architecture evaluation takes approximately 50 minutes and the operator set was reduced. We acknowledge this limitation and have taken the following measures in this study:

First, our primary comparisons are based on intra-dataset analysis rather than cross-dataset comparisons, as the lookup tables are generated independently for each dataset. For intra-dataset analysis, hardware variability does not affect the conclusions because all architectures within the same dataset are evaluated on the same platform. Cross-dataset discussions (such as transferability) serve only as qualitative observations and are explicitly qualified by this limitation.

Second, to promote reproducibility, we have unified the hardware and software environments for all newly added experiments. The specific configuration is as follows:

- **OS**: Ubuntu 20.04.6 LTS
- **CPU**: AMD EPYC 7763 64-Core Processor @ 2.45GHz (256 CPUs)
- **GPU**: 8 × NVIDIA A100-SXM4-40GB
- **Software**: Python 3.9, CUDA 12.2, PyTorch 1.13.1

We have publicly released the performance analysis scripts used to obtain the runtime measurements in Table 23 at https://github.com/THUMNLab/NAS-Bench-Graph. These scripts include instructions for setting up the aforementioned standardized environment (e.g., using Docker containers), allowing users to reproduce experimental results or recalibrate absolute timings according to their own environments.

Third, given the unique nature of ogbn-proteins (operator reduction due to memory constraints and significantly longer evaluation times), we treat it separately in our analysis and do not include it in aggregate statistics that might be affected by hardware heterogeneity. The observations of RL timeouts on larger datasets in Table 23 are based on consistent hardware platforms for each respective dataset; thus, the conclusion regarding RL's relative inefficiency remains robust.

6.3.4 Statistical Significance and Computational Overhead Analysis. To comprehensively evaluate the performance of different search methods under limited budgets, we conduct Wilcoxon signed-rank tests (Table 22) and report wall-clock time per dataset (Table 23).

Table 22 presents statistical comparisons across three dimensions. First, for the same method with different budgets, Random Search achieves significant performance gains when increasing budget from 1% to 5% (p=0.0109) and from 1% to 10% (p=0.0077), while Evolutionary Algorithm shows no significant gains (p>0.05), suggesting EA may converge prematurely with diminishing returns. Second, comparing different methods at the same budget reveals that Random Search significantly outperforms RL at 1% budget (p=0.0143), and significantly outperforms EA at both 5% (p=0.0266) and 10% (p=0.0109) budgets. Notably, RL at 5% and 10% budgets failed to complete on 8 out of 9 datasets, rendering statistical comparisons infeasible.

Table 23 reveals substantial efficiency differences. Random Search and EA complete all 9 datasets within 2.2 hours even at 10% budget, while RL at 5% and 10% budgets exceeds 24 hours on all datasets except the smallest (Proteins). This indicates that RL requires substantial samples to train its controller, causing search overhead to scale poorly with search space size and limiting its practicality in real-world NAS scenarios.

Combining statistical and efficiency analyses, Random Search achieves the best balance between accuracy and efficiency; EA offers marginally better accuracy on some datasets (as indicated by the win counts in Table 22) but at slightly higher overhead; RL is less competitive in both accuracy and efficiency despite its sophisticated modeling. This analysis provides guidance for practitioners selecting NAS methods based on their computational budgets.

6.3.5 Auxiliary LUT Subset Experiments. To further examine whether the conclusions remain stable under a smaller search space, we construct an auxiliary LUT subset by reducing the number of GNN types from five to three. This reduction decreases the total number of architectures to roughly 10% of the original space. We then run the same NAS-Bench-Graph methods—GNAS, Auto-GNN, Random, EA, and RL—on this subset using a 2% search budget. Experiments are conducted on the arXiv, Photo, and CiteSeer datasets, and the results are compared against those obtained from the full LUT. Table 24 shows that the rankings of different NAS methods remain largely consistent between the full LUT and the sliced LUT. On the arXiv and CiteSeer datasets, the highest performance method remained the same, indicating that reducing the architecture space does not significantly affect the relative ordering of the methods. On the Photo dataset, performance has minor shifts, but the overall ranking pattern remains similar. These results suggest that the NAS-Bench-Graph conclusions are stable even when applying a significantly reduced LUT subset.

Table 24. An auxiliary LUT slice with a small HPO around top architectures.

		LUT total			**LUT slice**		
Library	Method	arXiv	Photo	CiteSeer	arXiv	Photo	CiteSeer
AutoGL	GNAS	$72.00_{0.02}$	$92.43_{0.03}$	$\mathbf{70.89}_{0.16}$	$71.82_{0.09}$	$\mathbf{92.29}_{0.19}$	$\mathbf{69.76}_{0.21}$
	Auto-GNN	$\mathbf{72.13}_{0.03}$	$92.38_{0.01}$	$70.76_{0.12}$	$71.86_{0.17}$	$92.23_{0.00}$	$69.33_{0.29}$
NNI	Random	$72.04_{0.05}$	$\mathbf{92.44}_{0.02}$	$70.49_{0.08}$	$71.89_{0.12}$	$92.07_{0.09}$	$69.57_{0.39}$
	EA	$71.91_{0.06}$	$92.43_{0.02}$	$70.48_{0.12}$	$71.88_{0.14}$	$92.21_{0.14}$	$69.50_{0.17}$
	RL	$\mathbf{72.13}_{0.05}$	$92.42_{0.06}$	$70.66_{0.12}$	$\mathbf{71.90}_{0.11}$	$92.08_{0.20}$	$69.59_{0.32}$

7 Future Directions

We have discussed existing literature in automated graph machine learning approaches and libraries. Our discussion in detail contains how HPO and NAS can be applied to graph machine learning to handle problems in automated graph machine learning. We also introduce AutoGL, a dedicated framework and library for automated graph machine learning. In this section, we will suggest future directions deserving further investigations from both academia and industry. There exist plenty of challenges and opportunities worthy of future explorations.

- **Scalability**: AutoML has been successfully applied to various graph scenarios, however, there are still lots of future directions deserving further investigation regarding scalability to large-scale graphs. On the one hand, although HPO for large-scale graph machine learning has been preliminarily explored in literature [150], the Bayesian Optimization utilized in the model suffers from limited efficiency. Thus it will be interesting and challenging to explore how we can reduce the computational costs to realize fast hyper-parameter optimization. On the other hand, the scalability of NAS for graph machine learning has drawn few attentions from the researchers despite applications involving large-scale graphs are very common in real world, leaving a large space for further explorations.
- **Explainability**: Existing automated graph machine learning approaches are mainly based on black-box optimizations. For example, it is unclear why certain NAS models can perform better compared with others, and the explainability of NAS algorithms still lack systematic research efforts. There have been some preliminary studies on explainability of graph machine learning [193], and on explainable graph hyper-parameter optimization [158] via hyper-parameter importance decorrelation. However, further and deeper investigations on the explainability of automated graph machine learning are still of great importance.

- **Out-of-distribution generalization**: When applied to new graph datasets and tasks [43], there still need huge human efforts to construct task-specific graph HPO configurations and graph NAS frameworks, e.g., spaces and algorithms. The generalization of current graph HPO configurations and NAS frameworks are limited, especially training and testing data come from different distributions [22, 23, 51, 82, 84, 88, 89, 94, 95, 122, 161, 182, 206]. It will be a promising direction to study the out-of-distribution generalization abilities for both graph HPO and graph NAS algorithms which are capable of handling continuously and rapidly changing tasks [157, 160, 162].
- **Robustness**: Since many applications of AutoML on graphs are risk-sensitive, e.g., finance and healthcare, the robustness of the models is indispensable for actual usages. Though there exist some initial studies on the robustness [146] of graph machine learning, how to generalize these techniques into automated graph machine learning has not been explored.
- **Graph models for AutoML**: In this paper, we mainly focus on how AutoML methods are extended to graphs. The other direction, i.e., using graphs to help AutoML, is also feasible and promising. For example, we can model neural networks as a directed acyclic graph (DAG) to analyze their structures [172, 190] or adopt GNNs to facilitate NAS [36, 127, 140, 198]. Ultimately, we expect graphs and AutoML to form tighter connections and further facilitate each other.
- **Hardware-aware models**: To further improve the scalability of automated graph machine learning, hardware-aware models may be a critical step, especially in real industrial environments. Both hardware-aware graph models [5] and hardware-aware AutoML models [17, 71, 147] have been studied, but integrating these techniques is still in the early stage and poses significant challenges.
- **Comprehensive evaluation protocols**: Currently, most AutoML on graphs are tested on small traditional benchmarks such as three citations graphs, i.e., Cora, CiteSeer, and PubMed [136]. However, these benchmarks have been identified as insufficient to compare different graph machine learning models [137], not to mention AutoML on graphs. More comprehensive evaluation protocols are needed, e.g., on recently proposed graph machine learning benchmarks [37, 66], or new dedicated graph AutoML benchmarks [131] similar to the NAS-bench series [184] are needed.
- **Broader Scope of Applications**: While automated graph machine learning techniques have been applied to a range of practical use-cases, there's considerable potential for using these newly-developed techniques in the information retrieval (such as search engines, recommender systems) for achieving effective, reliable, and user-friendly predictions. A viable approach could involve using this specialized domain knowledge as a form of prior to guide both the hyperparameter optimization and architecture search strategies.

8 Discussion

8.1 Automated Graph Machine Learning vs Others

Automated Machine Learning tends to be more beneficial in graph learning than in CV or NLP due to several fundamental characteristics of graph-structured data: i) Structural heterogeneity and instability: Graph architectures are highly sensitive to dataset topology (e.g., homophily level, degree distribution, sparsity), and a model that performs well on one dataset often degrades substantially on another, making robust hand-tuning difficult; ii) Lack of universally strong "default architectures": Unlike CNNs and Transformers, GNNs do not yet exhibit a dominant backbone; small architectural changes (aggregation, propagation depth, message functions) can lead to large performance swings.

iii) Evolving application scenarios: Many practical graph tasks involve growing or dynamic graphs where the optimal architecture changes as the distribution shifts. Because of these factors, automated methods can provide consistency across diverse, less predictable graph regimes, while still remaining lightweight for small-scale tasks.

8.2 AutoML vs Manual Design

AutoML tends to excel in scenarios with large or complex search spaces, where exploring combinations of architectures and hyperparameters manually would be difficult or impractical. It also shows advantages on heterogeneous or evolving datasets, where the optimal architecture may vary across domains or over time, and in settings with limited human expertise, as it can systematically identify high-performing configurations without relying on domain knowledge. Conversely, for small or well-understood datasets, where the underlying structure is simple or standard architectures are well-established, careful manual tuning can achieve comparable or even better performance with significantly lower computational cost. Similarly, in resource-constrained settings or domains with strong prior knowledge, such as known motifs in graphs or feature interactions, manual design can effectively exploit this information to achieve strong results. Overall, AutoML provides the greatest benefits in complex, large-scale, or unfamiliar scenarios, while manual design remains competitive in simpler, well-understood cases.

8.3 Search Space Identifiability

Some operators exhibit symmetry or near-identical behaviors. For example, CONST vs GCN on regular or low-degree-variance graphs produce equivalent or very similar outputs across nodes, creating ambiguity in the search space. Such symmetries reduce identifiability, meaning that the NAS optimization cannot easily distinguish between these equivalent operations. As a result, search difficulty increases, and the optimization process may require more evaluations or become trapped in suboptimal regions. This issue also has implications for differentiable relaxations. Symmetric or nearly indistinguishable operators can lead to flat or ambiguous gradients, which may prevent differentiable NAS from effectively optimizing the architecture. Recognizing these symmetries and their impact on identifiability helps guide the design of more robust search spaces and strategies, avoiding common pitfalls in architecture search.

8.4 Theoretical Analyses of Automated Graph Machine Learning

This subsection reviews existing theoretical perspectives that are directly related to automated graph machine learning. From an optimization viewpoint, the methods can be formulated as bi-level optimization problems, where architectural or hyper-parameter variables are optimized with respect to validation performance while model parameters are optimized on training data. General theoretical treatments of bi-level optimization establish conditions for gradient consistency, implicit differentiation, and convergence of approximate solutions, providing a foundation for gradient-based HPO and differentiable NAS methods used in graph learning [46, 104]. The analyses clarify when the optimization of search variables faithfully reflects the underlying validation objective and when approximation errors may introduce bias. A second line of theoretical work focuses on the expressiveness of search spaces. For NAS, several studies analyze cell-based and modular search spaces and show that expanding architectural connectivity, such as allowing skip connections or flexible aggregation operators, monotonically enlarges the set of representable functions [103, 217]. These results partially justify the design of over-parameterized search spaces commonly adopted in graph

NAS, while also highlighting the risk of redundancy and non-identifiability in architecture representations. Another important theoretical issue concerns the consistency of performance estimation. To reduce computational cost, many graph HPO and NAS methods rely on low-fidelity proxies, such as subgraph sampling, early stopping, or weight sharing. Theoretical analyses of one-shot and weight-sharing NAS show that parameter coupling across candidate architectures can introduce ranking inconsistency, meaning that architectures preferred during search may not be optimal when trained independently [139, 174].

9 Conclusion

In this paper, we discuss the current state-of-the-art automated graph machine learning approaches and libraries. In particular, we in depth elaborate how graph hyperparameter optimization (HPO) and graph neural architecture search (NAS) have been developed to facilitate automated graph machine learning. We also introduce AutoGL, our dedicated framework and open source library for automated graph machine learning, and NAS-Bench-Graph, our tailored benchmark that enables fair, fully reproducible, and efficient empirical comparisons. Last but not least, we point out challenges and suggest promising directions deserving further investigations.

Scope, Assumptions, and Limitations. To clarify the applicability and boundaries of this work, we summarize the scope, assumptions, and limitations of our study as follows:

- **Scope.** This work focuses on automated graph machine learning, with an emphasis on two key paradigms: hyper-parameter optimization (HPO) and neural architecture search (NAS) for graph machine learning. In addition to surveying representative methods, we introduce the AutoGL library as a unified framework that integrates automated feature engineering, model training, HPO, NAS, and ensemble learning. We also discuss a benchmark designed to facilitate reproducible and standardized evaluation of graph NAS methods.
- **Assumptions.** The survey and benchmark assume commonly adopted graph learning settings, including node classification, link prediction, and graph classification tasks. The AutoGL framework assumes that graph datasets follow standard supervised or semi-supervised learning protocols and that candidate models can be trained within reasonable computational budgets. The benchmark settings also assume predefined search spaces and evaluation protocols to enable fair comparison across methods.
- **Limitations.** Several limitations should be noted. First, the scalability of automated graph machine learning remains challenging, especially for billion-scale graphs where both HPO and NAS may incur significant computational cost. Second, many current AutoML approaches for graphs rely on black-box optimization, leading to limited interpretability of the resulting architectures and hyper-parameters. Third, the generalization ability of existing HPO and NAS configurations across different datasets and distribution shifts is still limited. Finally, the experimental evaluation mainly relies on widely used benchmark datasets, which may not fully reflect the diversity and complexity of real-world graph applications.

Acknowledgments

This work was supported by the National Key Research and Development Program of China No.2023YFF1205001, Beijing National Research Center for Information Science and Technology under Grant No.BNR2026TD03005. All opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *DMKD* (2015).

[2] Alibaba. 2019. Euler: A distributed graph deep learning framework. https://github.com/alibaba/euler.

[3] OpenHGNN Team at GAMMA LAB and DGL Team. 2021. Heterogeneous Graph Neural Network. https://github.com/BUPT-GAMMA/OpenHGNN. [Online; accessed 28-Dec-2021].

[4] Natural Language Processing Lab at Tsinghua University. 2018. OpenNE: An open source toolkit for Network Embedding. https://github.com/thunlp/OpenNE.

[5] Adam Auten, Matthew Tomei, and Rakesh Kumar. 2020. Hardware acceleration of graph neural networks. In *DAC*.

[6] Peter W Battaglia et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261* (2018).

[7] James Bergstra et al. 2011. Algorithms for hyper-parameter optimization. *NeurIPS* (2011).

[8] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *JMLR* (2012).

[9] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*.

[10] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 2021. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[11] Karsten M Borgwardt et al. 2005. Protein function prediction via graph kernels. *Bioinformatics* (2005).

[12] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* (1998).

[13] Shaked Brody, Uri Alon, and Eran Yahav. 2022. How Attentive are Graph Attention Networks?. In *International Conference on Learning Representations*. https://openreview.net/forum?id=F72ximsx7C1

[14] Michael M Bronstein et al. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* (2017).

[15] Chenyang Bu, Yi Lu, and Fei Liu. 2021. Automatic Graph Learning with Evolutionary Algorithms: An Experimental Study. In *PRICAI*.

[16] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *TKDE* (2018).

[17] Han Cai, Ligeng Zhu, and Song Han. 2018. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations*.

[18] Jie Cai, Xin Wang, Haoyang Li, Ziwei Zhang, and Wenwu Zhu. 2024. Multimodal graph neural architecture search under distribution shifts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 8227–8235.

[19] Rongshen Cai et al. 2021. ALGNN: Auto-Designed Lightweight Graph Neural Network. In *PRICAI*.

[20] Shaofei Cai et al. 2021. Edge-featured Graph Neural Architecture Search. *arXiv:2109.01356* (2021).

[21] Shaofei Cai et al. 2021. Rethinking Graph Neural Architecture Search from Message-passing. *CVPR* (2021).

[22] Haibo Chen, Xin Wang, Jiaheng Chao, Ling Feng, and Wenwu Zhu. 2026. A Unified Graph Language Model for Multi-Domain Multi-Task Graph Alignment Instruction Tuning. *arXiv preprint arXiv:2605.12197* (2026).

[23] Haibo Chen, Xin Wang, Guanheng Chen, Yuan Meng, Haoyang Li, Yang Yao, Zeyang Zhang, Zhiqiang Zhang, JUN ZHOU, Ling Feng, and Wenwu Zhu. 2026. Adaptive Mixture of Disentangled Experts for Dynamic Graphs under Distribution Shifts. In *The Fourteenth International Conference on Learning Representations*.

[24] Haibo Chen, Xin Wang, Zeyang Zhang, Haoyang Li, Ling Feng, and Wenwu Zhu. 2025. Autogfm: Automated graph foundation model with adaptive architecture customization. In *Forty-second International Conference on Machine Learning*.

[25] Jiamin Chen et al. 2021. GraphPAS: Parallel Architecture Search for Graph Neural Networks. In *SIGIR*.

[26] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International conference on machine learning*. PMLR, 1725–1735.

[27] CA Coello Coello and Maximino Salazar Lechuga. 2002. MOPSO: A proposal for multiple objective particle swarm optimization. In *CEC*.

[28] Hejie Cui, Jiaying Lu, Yao Ge, and Carl Yang. 2022. How can graph neural networks help document retrieval: A case study on cord19 with concept map generation. In *European Conference on Information Retrieval*. Springer, 75–83.

[29] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE transactions on knowledge and data engineering* 31, 5 (2018), 833–852.

[30] CSIRO's Data61. 2018. StellarGraph Machine Learning Library. https://github.com/stellargraph/stellargraph.

[31] Axel de Romblay et al. 2018. MLBox, Machine Learning Box. https://github.com/AxeldeRomblay/MLBox.

[32] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34, 2 (1991), 786–797.

[33] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* 29 (2016), 3844–3852.
[34] Yuhui Ding et al. 2021. Diffmg: Differentiable meta graph search for heterogeneous graph neural networks. In *KDD*.
[35] Yuhui Ding, Quanming Yao, Huan Zhao, and Tong Zhang. 2021. Diffmg: Differentiable meta graph search for heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 279–288.
[36] Lukasz Dudziak et al. 2020. Brp-nas: Prediction-based nas using gcns. *NeurIPS* (2020).
[37] Vijay Prakash Dwivedi et al. 2020. Benchmarking graph neural networks. *arXiv:2003.00982* (2020).
[38] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *JMLR* (2019).
[39] Nick Erickson et al. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *arXiv:2003.06505* (2020).
[40] Federico Errica et al. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *ICLR*.
[41] Chenchen Feng, Yu He, Shiyang Wen, Guojun Liu, Liang Wang, Jian Xu, and Bo Zheng. 2022. DC-GNN: Decoupled Graph Neural Networks for Improving and Accelerating Large-Scale E-commerce Retrieval. In *Companion Proceedings of the Web Conference 2022*. 32–40.
[42] Guosheng Feng, Chunnan Wang, and Hongzhi Wang. 2021. Search For Deep Graph Neural Networks. *arXiv:2109.10047* (2021).
[43] Wei Feng, Haoyang Li, Xin Wang, Xuguang Duan, Zi Qian, Wu Liu, and Wenwu Zhu. 2023. Multimedia cognition and evaluation in open environments. In *Proceedings of the 1st International Workshop on Multimedia Content Generation and Evaluation: New Methods and Practice*. 9–18.
[44] Matthias Feurer et al. 2019. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*.
[45] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop*.
[46] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. 2018. Bilevel programming for hyperparameter optimization and meta-learning. In *International conference on machine learning*. PMLR, 1568–1577.
[47] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *Proceedings of the 36th International Conference on Machine Learning*.
[48] Yang Gao et al. 2020. Graph Neural Architecture Search. In *IJCAI*.
[49] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2020. Graph Neural Architecture Search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Vol. 20. 1403–1409.
[50] Yang Gao, Peng Zhang, Chuan Zhou, Hong Yang, Zhao Li, Yue Hu, and S Yu Philip. 2023. HGNAS++: efficient architecture search for heterogeneous graph neural networks. *IEEE Transactions on Knowledge and Data Engineering* (2023).
[51] Chendi Ge, Xin Wang, Zeyang Zhang, Hong Chen, Jiapei Fan, Longtao Huang, Hui Xue, and Wenwu Zhu. 2025. Dynamic mixture of curriculum lora experts for continual multimodal instruction tuning. *arXiv preprint arXiv:2506.11672* (2025).
[52] Chendi Ge, Xin Wang, Ziwei Zhang, Yijian Qin, Hong Chen, Haiyang Wu, Yang Zhang, Yuekui Yang, and Wenwu Zhu. 2025. Behavior importance-aware graph neural architecture search for cross-domain recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 11708–11716.
[53] Justin Gilmer et al. 2017. Neural message passing for Quantum chemistry. In *ICML*.
[54] Palash Goyal and Emilio Ferrara. 2018. GEM: A Python package for graph embedding methods. *JOSS* (2018).
[55] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *KBS* (2018).
[56] Daniele Grattarola and Cesare Alippi. 2020. Graph neural networks in tensorflow and keras with spektral. *ICML workshop* (2020).
[57] Chaoyu Guan, Xin Wang, Hong Chen, Ziwei Zhang, and Wenwu Zhu. 2022. Large-scale graph neural architecture search. In *International Conference on Machine Learning*. PMLR, 7968–7981.
[58] Chaoyu Guan, Xin Wang, and Wenwu Zhu. 2021. AutoAttend: Automated Attention Representation Search. In *ICML*.
[59] Mengying Guo et al. 2021. JITuNE: Just-In-Time Hyperparameter Tuning for Network Embedding Algorithms. *arXiv:2101.06427* (2021).
[60] Zichao Guo et al. 2020. Single path one-shot neural architecture search with uniform sampling. In *ECCV*.
[61] Deisy Morselli Gysi et al. 2020. Network medicine framework for identifying drug repurposing opportunities for covid-19. *arXiv:2004.07229* (2020).
[62] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL).

[63] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
[64] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *IEEE DEBU* (2017).
[65] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2020. AutoML: A Survey of the State-of-the-Art. *KBS* (2020).
[66] Weihua Hu et al. 2020. Open graph benchmark: Datasets for machine learning on graphs. *NeurIPS* (2020).
[67] 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.
[68] Vassilis N Ioannidis, Da Zheng, and George Karypis. 2020. Few-shot link prediction via graph neural networks for covid-19 drug-repurposing. *arXiv:2007.10261* (2020).
[69] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *ICLR*.
[70] Shengli Jiang and Prasanna Balaprakash. 2020. Graph Neural Network Architecture Search for Molecular Property Prediction. In *IEEE Big Data*.
[71] Yuhang Jiang, Xin Wang, and Wenwu Zhu. 2020. Hardware-Aware Transformable Architecture Search with Efficient Search Space. In *ICME*.
[72] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *KDD*.
[73] Amol Kapoor et al. 2020. Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv:2007.03113* (2020).
[74] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).
[75] T Kipf et al. 2018. Neural Relational Inference for Interacting Systems. *ICML* (2018).
[76] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[77] Tsinghua University Knowledge Engineering Group. 2020. CogDL: An Extensive Research Toolkit for Deep Learning on Graphs. https://github.com/THUDM/cogdl.
[78] Adam Lerer et al. 2019. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *SysML*.
[79] Chao Li, Hao Xu, and Kun He. 2023. Differentiable meta multigraph search with partial message propagation on heterogeneous information networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 8518–8526.
[80] Guohao Li et al. 2019. Deepgcns: Can gcns go as deep as cnns?. In *ICCV*.
[81] Guohao Li et al. 2020. Sgas: Sequential greedy architecture search. In *CVPR*.
[82] Haoyang Li, Haibo Chen, Xin Wang, and Wenwu Zhu. 2026. Out-of-Distribution Generalization in Graph Foundation Models. *arXiv preprint arXiv:2601.21067* (2026).
[83] Haoyang Li, Peng Cui, Chengxi Zang, Tianyang Zhang, Wenwu Zhu, and Yishi Lin. 2019. Fates of microscopic social ecosystems: Keep alive or dead?. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 668–676.
[84] Haoyang Li, Xin Wang, Zeyang Zhang, Haibo Chen, Ziwei Zhang, and Wenwu Zhu. 2024. Disentangled graph self-supervised learning for out-of-distribution generalization. In *Forty-first International Conference on Machine Learning*.
[85] Haoyang Li, Xin Wang, Ziwei Zhang, Jianxin Ma, Peng Cui, and Wenwu Zhu. 2021. Intention-aware sequential recommendation with structured intent transition. *IEEE Transactions on Knowledge and Data Engineering* 34, 11 (2021), 5403–5414.
[86] Haoyang Li, Xin Wang, Zeyang Zhang, Zongyuan Wu, Linxin Xiao, Yaofei Wu, and Wenwu Zhu. 2025. Self-supervised Masked Graph Autoencoder via Structure-aware Curriculum. (2025).
[87] Haoyang Li, Xin Wang, Ziwei Zhang, Zehuan Yuan, Hang Li, and Wenwu Zhu. 2021. Disentangled contrastive learning on graphs. *Advances in Neural Information Processing Systems* 34 (2021), 21872–21884.
[88] Haoyang Li, Xin Wang, Ziwei Zhang, and Wenwu Zhu. 2022. Ood-gnn: Out-of-distribution generalized graph neural network. *IEEE Transactions on Knowledge and Data Engineering* (2022).
[89] Haoyang Li, Xin Wang, Ziwei Zhang, and Wenwu Zhu. 2025. Out-of-distribution generalization on graphs: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025).
[90] Haoyang Li, Xin Wang, and Wenwu Zhu. 2023. Curriculum graph machine learning: A survey. *International Joint Conference on Artificial Intelligence* (2023).
[91] Haoyang Li, Xin Wang, Xueling Zhu, Weigao Wen, and Wenwu Zhu. 2025. Disentangling invariant subgraph via variance contrastive estimation under distribution shifts. In *International Conference on Machine Learning*.

[92] Haoyang Li, Chengxi Zang, Zhenxing Xu, Weishen Pan, Suraj Rajendran, Yong Chen, and Fei Wang. 2025. Federated target trial emulation using distributed observational data for treatment effect estimation. *npj Digital Medicine* 8, 1 (2025), 387.
[93] Haoyang Li, Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2022. Disentangled graph contrastive learning with independence promotion. *IEEE Transactions on Knowledge and Data Engineering* (2022).
[94] Haoyang Li, Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2022. Learning invariant graph representations for out-of-distribution generalization. *Advances in Neural Information Processing Systems* 35 (2022), 11828–11841.
[95] Haoyang Li, Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2023. Invariant Node Representation Learning under Distribution Shifts with Multiple Latent Environments. *ACM Transactions on Information Systems* 42, 1 (2023), 1–30.
[96] Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*. PMLR, 367–377.
[97] Peiwen Li, Xin Wang, Zeyang Zhang, Ziwei Zhang, Fang Shen, Jialong Wang, Yang Li, and Wenwu Zhu. 2025. Causal-aware graph neural architecture search under distribution shifts. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 1458–1469.
[98] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.
[99] Yaguang Li et al. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*.
[100] Yanxi Li et al. 2021. One-shot Graph Neural Architecture Search with Dynamic Search Space. *AAAI* (2021).
[101] Yaoman Li and Irwin King. 2020. AutoGraph: Automated Graph Neural Network. In *ICONIP*.
[102] Ke Liang, Jim Tan, Dongrui Zeng, Yongzhe Huang, Xiaolei Huang, and Gang Tan. 2023. Abslearn: a gnn-based framework for aliasing and buffer-size information retrieval. *Pattern Analysis and Applications* (2023), 1–19.
[103] Bo Liu, Huiwen Zhao, Tongtong Yuan, Ting Zhang, and Zhaoying Liu. 2025. Rethinking cell-based neural architecture search: A theoretical perspective. *Neural Networks* (2025), 107557.
[104] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. In *ICLR*.
[105] Siwei Liu, Zaiqiao Meng, Craig Macdonald, and Iadh Ounis. 2023. Graph neural pre-training for recommendation with side information. *ACM Transactions on Information Systems* 41, 3 (2023), 1–28.
[106] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. 2021. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems* 34, 2 (2021), 550–570.
[107] Yue Liu, Xin Wang, Xue Xu, Jianbo Yang, and Wenwu Zhu. 2021. Meta hyperparameter optimization with adversarial proxy subsets sampling. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1109–1118.
[108] Oliver Lloyd, Yi Liu, and Tom R Gaunt. 2023. Assessing the effects of hyperparameters on knowledge graph embedding quality. *Journal of big Data* 10, 1 (2023), 1–15.
[109] Qing Lu, Weiwen Jiang, Meng Jiang, Jingtong Hu, Sakyasingha Dasgupta, and Yiyu Shi. 2020. Fgnas: Fpga-aware graph neural architecture search. (2020).
[110] Weigang Lu, Yibing Zhan, Binbin Lin, Ziyu Guan, Liu Liu, Baosheng Yu, Wei Zhao, Yaming Yang, and Dacheng Tao. 2024. SkipNode: On alleviating performance degradation for deep graph convolutional networks. *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2024), 7030–7043.
[111] Jianxin Ma et al. 2019. Learning disentangled representations for recommendation. In *NeurIPS*.
[112] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*.
[113] Microsoft. 2021. Neural Network Intelligence. https://github.com/microsoft/nni
[114] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.
[115] MLJAR. 2019. MLJAR Automated Machine Learning. https://github.com/mljar/mljar-supervised.
[116] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.
[117] Mark Newman. 2018. *Networks*. Oxford university press.
[118] NLP and Paddle Teams at Baidu. 2021. Paddle Graph Learning. https://github.com/PaddlePaddle/PGL. [Online; accessed 28-Dec-2021].
[119] Matheus Nunes et al. 2020. Neural Architecture Search in Graph Neural Networks. In *Brazilian Conference on Intelligent Systems*.
[120] Randal S. Olson et al. 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *GECCO*.
[121] Zheyi Pan et al. 2021. AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graphs. *WWW* (2021).

[122] Zirui Pan, Xin Wang, Yipeng Zhang, Hong Chen, Kecheng Zheng, and Wenwu Zhu. 2026. Reasoning Diffusion for Unpaired Test Time Out-of-distribution Text-Image to Video Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 636–646.
[123] Wei Peng et al. 2020. Learning Graph Convolutional Network for Skeleton-Based Human Action Recognition by Neural Searching. *AAAI* (2020).
[124] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*.
[125] Hieu Pham et al. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *ICML*.
[126] Pourchot and Sigaud. 2019. CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *ICLR*.
[127] Yijian Qin, Xin Wang, Peng Cui, and Wenwu Zhu. 2002. GQNAS: Graph Q Network for Neural Architecture Search. In *ICDM*.
[128] Yijian Qin, Xin Wang, Ziwei Zhang, Hong Chen, and Wenwu Zhu. 2023. Multi-task graph neural architecture search with task-aware collaboration and curriculum. *Advances in neural information processing systems* 36 (2023), 24879–24891.
[129] Yijian Qin, Xin Wang, Ziwei Zhang, Pengtao Xie, and Wenwu Zhu. 2022. Graph neural architecture search under distribution shifts. In *International Conference on Machine Learning*. PMLR, 18083–18095.
[130] Yijian Qin, Xin Wang, Zeyang Zhang, and Wenwu Zhu. 2021. Graph differentiable architecture search with structure learning. *Advances in neural information processing systems* 34 (2021), 16860–16872.
[131] Yijian Qin, Ziwei Zhang, Xin Wang, Zeyang Zhang, and Wenwu Zhu. 2022. NAS-Bench-Graph: Benchmarking graph neural architecture search. *Advances in Neural Information Processing Systems* 35 (2022), 54–69.
[132] Esteban Real et al. 2017. Large-scale evolution of image classifiers. In *ICML*.
[133] Esteban Real et al. 2019. Regularized evolution for image classifier architecture search. In *AAAI*.
[134] Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. 2018. Deep inductive graph representation learning. *TKDE* (2018).
[135] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *CIKM*.
[136] Prithviraj Sen et al. 2008. Collective classification in network data. *AI magazine* (2008).
[137] Oleksandr Shchur et al. 2018. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS 2018* (2018).
[138] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS 2018* (2018).
[139] Xuan Shen, Pu Zhao, Yifan Gong, Zhenglun Kong, Zheng Zhan, Yushu Wu, Ming Lin, Chao Wu, Xue Lin, and Yanzhi Wang. 2024. Search for efficient large language models. *Advances in Neural Information Processing Systems* 37 (2024), 139294–139315.
[140] Han Shi et al. 2020. Bridging the Gap between Sample-based and One-shot Neural Architecture Search with BONAS. *NeurIPS* (2020).
[141] Min Shi et al. 2020. Evolutionary Architecture Search for Graph Neural Networks. *arXiv:2009.10199* (2020).
[142] Jan Pfeifer Sibon Li et al. 2021. TensorFlow GNN. https://github.com/tensorflow/gnn. [Online; accessed 28-Dec-2021].
[143] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *NeurIPS* (2012).
[144] Chang Su et al. 2020. Network embedding in biomedical data science. *Briefings in bioinformatics* (2020).
[145] Junwei Sun, Bai Wang, and Bin Wu. 2021. Automated Graph Representation Learning for Node Classification. In *IJCNN*.
[146] Lichao Sun et al. 2018. Adversarial Attack and Defense on Graph Data: A Survey. *arXiv:1812.10528* (2018).
[147] Mingxing Tan et al. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*.
[148] Sibo Tian, Xin Wang, Zeyang Zhang, Haibo Chen, and Wenwu Zhu. [n. d.]. Out-of-Distribution Generalized Graph Anomaly Detection with Homophily-aware Environment Mixup. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
[149] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2347–2356.
[150] Ke Tu et al. 2019. Autone: Hyperparameter optimization for massive network embedding. In *KDD*.
[151] Petar Veličković et al. 2018. Graph Attention Networks. In *ICLR*.
[152] Chunnan Wang et al. 2021. FL-AGCNS: Federated Learning Framework for Automatic Graph Convolutional Network Search. *arXiv:2104.04141* (2021).
[153] Long Wang, Xiangpeng Li, Haisu Zhang, Yalan Dai, and Sheng Zhang. 2022. GNN-based retrieval and recommadation system: A semantic enhenced graph model. In *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Vol. 5. IEEE, 1823–1830.

[154] Mingzi Wang, Yuan Meng, Chen Tang, Weixiang Zhang, Yijian Qin, Yang Yao, Yingxin Li, Tongtong Feng, Xin Wang, Xun Guan, et al. 2025. JAQ: Joint Efficient Architecture Design and Low-Bit Quantization with Hardware-Software Co-Exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 21171–21179.
[155] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).
[156] Quan Wang et al. 2017. Knowledge graph embedding: A survey of approaches and applications. *TKDE* (2017).
[157] Qiyi Wang, Yinning Shao, Yunlong Ma, and Min Liu. 2025. Nodenas: Node-specific graph neural architecture search for out-of-distribution generalization. *arXiv preprint arXiv:2503.02448* (2025).
[158] Xin Wang et al. 2021. Explainable Automated Graph Representation Learning with Hyperparameter Importance. In *ICML*.
[159] Xin Wang, Hong Chen, Zirui Pan, Yuwei Zhou, Chaoyu Guan, Lifeng Sun, and Wenwu Zhu. 2025. Automated disentangled sequential recommendation with large language models. *ACM Transactions on Information Systems* 43, 2 (2025), 1–29.
[160] Xin Wang, Haoyang Li, Haibo Chen, Zeyang Zhang, and Wenwu Zhu. 2025. Modular machine learning: An indispensable path towards new-generation large language models. *arXiv preprint arXiv:2504.20020* (2025).
[161] Xin Wang, Haoyang Li, Zeyang Zhang, Haibo Chen, Tong Xiao, Kehan Li, and Wenwu Zhu. 2025. Uncertainty-aware Disentangled Dynamic Graph Attention Network for Out-of-Distribution Generalization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025).
[162] Xin Wang, Zeyang Zhang, Linxin Xiao, Haibo Chen, Chendi Ge, and Wenwu Zhu. 2025. Towards multimodal graph large language model. *Science China Information Sciences* 68, 11 (2025), 1–20.
[163] Xin Wang, Yuwei Zhou, Bin Huang, Hong Chen, and Wenwu Zhu. 2026. Multi-modal Generative AI: Multi-modal LLMs, Diffusions and the Unification. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)* (2026).
[164] Zhili Wang, Shimin Di, and Lei Chen. 2021. AutoGEL: An Automated Graph Neural Network with Explicit Link Information. *NeurIPS* (2021).
[165] Lanning Wei, Huan Zhao, and Zhiqiang He. 2021. Learn Layer-wise Connections in Graph Neural Networks. *arXiv:2112.13585* (2021).
[166] Lanning Wei, Huan Zhao, Quanming Yao, and Zhiqiang He. 2021. Pooling architecture search for graph classification. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2091–2100.
[167] Zhikun Wei, Xin Wang, and Wenwu Zhu. 2021. AutoIAS: Automatic Integrated Architecture Searcher for Click-Trough Rate Prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2101–2110.
[168] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. Pmlr, 6861–6871.
[169] Zonghan Wu et al. 2020. A comprehensive survey on graph neural networks. *TNNLS* (2020).
[170] Linxin Xiao, Xin Wang, Zeyang Zhang, Yang Yao, and Wenwu Zhu. 2025. DyNAS-DDI: Dynamic Pairwise Architecture Search for Generalizable Drug-Drug Interaction LLM. In *Proceedings of the 33rd ACM International Conference on Multimedia*. 2216–2225.
[171] Beini Xie, Heng Chang, Ziwei Zhang, Xin Wang, Daixin Wang, Zhiqiang Zhang, Rex Ying, and Wenwu Zhu. 2023. Adversarially Robust Neural Architecture Search for Graph Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8143–8152.
[172] Saining Xie et al. 2019. Exploring randomly wired neural networks for image recognition. In *ICCV*.
[173] Sirui Xie et al. 2019. SNAS: stochastic neural architecture search. *ICLR* (2019).
[174] Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. 2022. Analyzing and mitigating interference in neural architecture search. In *International Conference on Machine Learning*. PMLR, 24646–24662.
[175] Keyulu Xu et al. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*.
[176] Keyulu Xu et al. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations (ICLR)*.
[177] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD*.
[178] Dingqi Yang, Bingqing Qu, Rana Hussein, Paolo Rosso, Philippe Cudré-Mauroux, and Jie Liu. 2022. Revisiting Embedding Based Graph Analyses: Hyperparameters Matter! *IEEE Transactions on Knowledge and Data Engineering* (2022).
[179] Xueying Yang, Jiamian Wang, Xujiang Zhao, Sheng Li, and Zhiqiang Tao. 2022. Calibrate Automated Graph Neural Network via Hyperparameter Uncertainty. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 4640–4644.

[180] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*. PMLR, 40–48.
[181] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv:1810.13306* (2018).
[182] Yang Yao, Xin Wang, Yuan Meng, Zeyang Zhang, Hong Mei, and Wenwu Zhu. 2026. Disentangled Graph LLM for Molecule Graph Editing under Distribution Shifts. In *Proceedings of the ACM Web Conference 2026*. 1149–1159.
[183] Yang Yao, Xin Wang, Yijian Qin, Ziwei Zhang, Wenwu Zhu, and Hong Mei. 2024. Data-augmented curriculum graph neural architecture search under distribution shifts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 16433–16441.
[184] Chris Ying et al. 2019. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*.
[185] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Badly for Graph Representation?. In *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=OeWooOxFwDa
[186] Rex Ying et al. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.
[187] Rex Ying et al. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*.
[188] Minji Yoon et al. 2020. Autonomous Graph Mining Algorithm Search with Best Speed/Accuracy Trade-off. In *IEEE ICDM*.
[189] Minji Yoon, Théophile Gervet, Bryan Hooi, and Christos Faloutsos. 2022. Autonomous graph mining algorithm search with best performance trade-off. *Knowledge and Information Systems* 64, 6 (2022), 1571–1602.
[190] Jiaxuan You et al. 2020. Graph structure of neural networks. In *ICML*.
[191] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *NeurIPS* (2020).
[192] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *IJCAI*.
[193] Hao Yuan et al. 2020. Explainability in Graph Neural Networks: A Taxonomic Survey. *arXiv:2012.15445* (2020).
[194] Yingfang Yuan et al. 2021. A Novel Genetic Algorithm with Hierarchical Evaluation Strategy for Hyperparameter Optimisation of Graph Neural Networks. *arXiv:2101.09300* (2021).
[195] Yingfang Yuan, Wenjun Wang, and Wei Pang. 2021. A systematic comparison study on hyperparameter optimisation of graph neural networks for molecular property prediction. In *GECCO*.
[196] Yingfang Yuan, Wenjun Wang, and Wei Pang. 2021. Which Hyperparameters to Optimise? An Investigation of Evolutionary Hyperparameter Optimisation in Graph Neural Network for Molecular Property Prediction. In *GECCO*.
[197] Chengxi Zang et al. 2018. On power law growth of social networks. *TKDE* (2018).
[198] Chris Zhang, Mengye Ren, and Raquel Urtasun. 2018. Graph HyperNetworks for Neural Architecture Search. In *ICLR*.
[199] Quanlu Zhang et al. 2020. Retiarii: A Deep Learning Exploratory-Training Framework. In *OSDI*.
[200] Wentao Zhang, Yu Shen, Zheyu Lin, Yang Li, Xiaosen Li, Wen Ouyang, Yangyu Tao, Zhi Yang, and Bin Cui. 2022. Pasca: A graph neural architecture search system under the scalable paradigm. In *Proceedings of the ACM Web Conference 2022*. 1817–1828.
[201] Yongan Zhang, Haoran You, Yonggan Fu, Tong Geng, Ang Li, and Yingyan Lin. 2021. G-CoS: Gnn-accelerator co-search towards both better accuracy and efficiency. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.
[202] Yongqi Zhang, Zhanke Zhou, Quanming Yao, and Yong Li. 2022. KGTuner: Efficient Hyper-parameter Search for Knowledge Graph Learning. *arXiv preprint arXiv:2205.02460* (2022).
[203] Ziwei Zhang et al. 2018. Arbitrary-order proximity preserved network embedding. In *KDD*.
[204] Ziwei Zhang, Peng Cui, Jian Pei, Xin Wang, and Wenwu Zhu. 2021. Eigen-gnn: A graph structure preserving plug-in for gnns. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[205] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *TKDE* (2020).
[206] Zeyang Zhang, Xin Wang, Haibo Chen, Haoyang Li, and Wenwu Zhu. 2024. Disentangled dynamic graph attention network for out-of-distribution sequential recommendation. *ACM Transactions on Information Systems* 43, 1 (2024), 1–42.
[207] Zizhao Zhang, Xin Wang, Chaoyu Guan, Ziwei Zhang, Haoyang Li, and Wenwu Zhu. 2023. AutoGT: Automated Graph Transformer Architecture Search. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=GcM7qfl5zY
[208] Zeyang Zhang, Xin Wang, Yijian Qin, Hong Chen, Ziwei Zhang, Xu Chu, and Wenwu Zhu. 2024. Disentangled continual graph neural architecture search with invariant modular supernet. In *Forty-first International Conference on Machine Learning*.

[209] Zeyang Zhang, Ziwei Zhang, Xin Wang, Yijian Qin, Zhou Qin, and Wenwu Zhu. 2023. Dynamic Heterogeneous Graph Attention Neural Architecture Search. *AAAI* (2023).
[210] Huan Zhao et al. 2021. Efficient Graph Neural Architecture Search. https://openreview.net/forum?id=IjIzIOkK2D6
[211] Huan Zhao, Lanning Wei, and Quanming Yao. 2020. Simplifying Architecture Search for Graph Neural Network. *arXiv:2008.11652* (2020).
[212] Huan Zhao, Quanming Yao, and Weiwei Tu. 2021. Search to aggregate neighborhood for graph neural network. *ICDE* (2021).
[213] Yiren Zhao et al. 2020. Probabilistic Dual Network Architecture Search on Graphs. *arXiv:2003.09676* (2020).
[214] Yiren Zhao et al. 2021. Learned Low Precision Graph Neural Networks. In *EuroMLSys*.
[215] Jie Zhou et al. 2018. Graph neural networks: A review of methods and applications. *arXiv:1812.08434* (2018).
[216] Kaixiong Zhou et al. 2019. Auto-gnn: Neural architecture search of graph neural networks. *arXiv:1909.03184* (2019).
[217] Pan Zhou, Caiming Xiong, Richard Socher, and Steven Chu Hong Hoi. 2020. Theory-inspired path-regularized differential network architecture search. *Advances in Neural Information Processing Systems* 33 (2020), 8296–8307.
[218] Rong Zhu et al. 2019. AliGraph: A Comprehensive Graph Neural Network Platform. *VLDB* (2019).
[219] Ronghang Zhu, Zhiqiang Tao, Yaliang Li, and Sheng Li. 2021. Automated graph learning via population based self-tuning GCN. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2096–2100.
[220] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* (2017).
[221] Barret Zoph et al. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*.
[222] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

Received xx xx xx; revised xx xx xx; accepted xx xx xx