Graph Neural Network Architecture Search for Molecular Property Prediction

Shengli Jiang
Department of Chemical and Biological Engineering
University of Wisconsin — Madison
Madison, WI, USA
sjiang87@wisc.edu

Prasanna Balaprakash
Mathematics and Computer Science Division &
Leadership Computing Facility
Argonne National Laboratory
Lemont, IL, USA
pbalapra@anl.gov

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378060

***Abstract*—Predicting the properties of a molecule from its structure is a challenging task. Recently, deep learning methods have improved the state of the art for this task because of their ability to learn useful features from the given data. By treating molecule structure as graphs, where atoms and bonds are modeled as nodes and edges, graph neural networks (GNNs) have been widely used to predict molecular properties. However, the design and development of GNNs for a given dataset rely on labor-intensive design and tuning of the network architectures. Neural architecture search (NAS) is a promising approach to discover high-performing neural network architectures automatically. To that end, we develop an NAS approach to automate the design and development of GNNs for molecular property prediction. Specifically, w e f ocus o n a utomated d evelopment of message-passing neural networks (MPNNs) to predict the molecular properties of small molecules in quantum mechanics and physical chemistry datasets from the MoleculeNet benchmark. We demonstrate the superiority of the automatically discovered MPNNs by comparing them with manually designed GNNs from the MoleculeNet benchmark. We demonstrate that customizing the architecture is critical to enhancing performance in molecular property prediction and that the proposed approach can perform customization automatically with minimal manual effort.**

***Index Terms*—graph neural networks, neural architecture search, regularized evolution, deep learning, AutoML**

I. Introduction

Molecular property prediction is an important task in science and engineering. In pharmaceutical science, predicting the drug molecule properties can enable the design of next-generation drugs [1]. In materials science, predicting the band gap of perovskites is essential in the design of novel solar cells [2]. In electrochemistry, finding n ew e lectrolytes w ith high Li-ion conductivity will speed up solid-state Li-ion battery research [3]. Traditionally, simulation-based methods, such as quantum Monte Carlo (QMC) [4], molecular dynamics (MD) [5], and density functional theory (DFT) [6], are used to predict molecular properties. QMC and MD consider atoms as classical particles and simulate the interactions between atoms or molecules by using empirical potential forms such as Lennard-Jones. DFT can predict the quantum properties of molecules by calculating the ground-state energy of systems using exchange correlation functions [6]. Although they achieve high prediction accuracy, the traditional methods are computationally expensive [7]. Prior studies have shown that predicting the property of a molecule with 20 atoms using DFT can take an hour [8].

In the past decade, because of the availability of cheap computational power and increased coordinated data-collection efforts, a wide range of molecular datasets have been generated by simulations and experiments [9]. However, developing accurate predictive models for these molecular datasets remains a difficult task. For a molecular propriety dataset, the amount of training data is often limited by the expense of simulation and/or experiments. The molecular datasets are diverse in terms of their molecular structure and properties. Consequently, a model trained for one molecular dataset cannot be transferred to another because of the non-Euclidean characteristics of the molecular structure data [10].

Graph neural networks (GNNs) have been widely used to predict molecular properties [8], [10]. By treating molecules as graphs, where nodes are atoms and edges are bonds, GNNs capture the non-Euclidean nature of molecules. Message-passing neural networks (MPNNs) [8], a class of GNNs, provide a generic framework to incorporate complex node (atomic) and edge (bond) features. The common node features include the mass number, the atomic number, and the number of valence electrons. The common edge features comprise the bond type (e.g., single or double bond), the bond length, and whether a bond is in a ring. Following the concepts of MPNNs, node features can be passed as *messages* from one node to another along edges. Researchers can discern the hidden feature of a node by iteratively aggregating features from neighboring nodes and the node itself. Once identified, the formerly hidden features can be used by other machine learning techniques (e.g., fully connected neural networks) to perform specific tasks (e.g., classification, regression, and clustering). The advantage of MPNNs compared with other GNNs can be attributed to their ability to integrate edge features into the message passing of node features. Specifically, edge features are processed by a neural network to generate some weights to guide the message passing between nodes.

Despite the superior performance of MPNNs, arduous tuning of the network architecture impedes their development and wider adoption. To obtain high prediction accuracy for different datasets, we need to tune various MPNN components, including message, aggregate, update, and gather (readout) functions. For example, GG-NN [11] uses a gated graph unit as the update function, whereas an Interaction Network [12] simply updates with a dense neural network. The manually designed MPNNs not only require a substantial number of experiments in the design space but also tend to be lower-performing when applied to a new dataset. The data-specific nature of MPNN design means that it urgently needs an automated MPNN search to identify the best task-specific architecture for a given dataset.

Neural architecture search (NAS) has been designed to automatically search for the best network architecture for a given dataset. Specifically, it uses a search method (e.g., reinforcement learning, evolutionary algorithm, or stochastic gradient descent) to explore the user-defined search space and chooses the best architecture based on the performance of the generated model (e.g., validation accuracy) on a given task. The search space contains all possible architectures. The architectures discovered by NAS have been shown to outperform manually designed architectures in various tasks such as image classification [13], image segmentation [14], natural language processing [15], and time series prediction [16]. Despite their impact, NAS methods for GNNs and, in particular, MPNNs have received relatively little attention in the literature. In two recent studies [17], [18], NAS methods were presented that generate GNNs for classifying node properties. In these studies, however, the NAS-GNN search space lacks the integration of edge features, which are crucial for molecular property prediction [8].

We focus on developing NAS for MPNNs that incorporates both the node and edge features to predict molecular properties. MPNNs have been widely used to study molecular properties [8]. Borrowing the idea of Res-Net [19], we develop a NAS search space for stacked MPNNs with multiple MPNN cells and skip connections. We implement our approach in DeepHyper [20] — an open-source, scalable automated machine learning (AutoML) package — and enhance its capability to generate MPNN neural architectures for molecular property prediction. We evaluate the efficacy of our approach on quantum mechanics and physical chemistry datasets from the MoleculeNet benchmark [10]. The contributions of the paper are as follows.

1) We develop an NAS approach to generate stacked MPNN architecture to predict the molecular properties of small molecules.
2) We show that automatically obtained stacked MPNNs compare favorably with manually designed networks to predict molecular properties on three quantum mechanics and three physical chemistry datasets from the MoleculeNet benchmark.
3) DeepHyper has been utilized for automated discovery of fully connected neural networks on tabular data [20] and long short-term memory (LSTM) on time series data [16]. In this study, we enhance DeepHyper's capabilities for discovering stacked MPNN architectures.

II. MoleculeNet Benchmark

We use the small molecule datasets provided by MoleculeNet [10] that include two groups of datasets.

A. Quantum Physics

The quantum physics group comprises QM7, QM8, and QM9 datasets. The QM7 dataset is a subset of the GDB-13 database [9], which contains about 1 billion organic molecules with up to seven "heavy" atoms (C, N, O, S). The QM7 dataset includes 7,160 molecules and their corresponding atomization energies. The QM8 dataset is a subset of the GDB-17 database [21]. The dataset contains four excited-state properties of 21,786 molecules with up to eight heavy atoms collected by using three methods, including time-dependent DFTs and second-order approximate coupled-cluster. The QM9 dataset provides twelve properties, including geometric, energetic, electronic, and thermodynamic properties, for a subset of the GDB-17 database [21]. The dataset is made up of 133,865 molecules with up to nine heavy atoms, all modeled using DFT.

B. Physical Chemistry

The physical chemistry groups consists of ESOL, Free Solvation Database (FreeSolv), and Lipophilicity datasets. The ESOL dataset comprises water solubility data for 1,128 molecules [22]. The FreeSolv dataset consists of experimental and calculated hydration-free energy of 642 small molecules in water [23]. The Lipophilicity dataset, chosen from the ChEMBL database [24], provides experimental results of the octanol/water distribution coefficient (important feature for drug design) for 4,200 molecules.

C. Molecule Featurization

MoleculeNet provides multiple methods to represent molecules. In our study, we use the Weave featurizer that encodes both local chemical environment and connectivity between atoms. Specifically, node (atomic) features are in vector format, while the connectivity is represented by a list of edge (pair) features. A node feature vector $\mathbf{h} \in \mathbb{R}^{75}$ contains information on atom type, hybridization type, and valence structure. An edge vector $\mathbf{e} \in \mathbb{R}^{14}$ encodes bond types, graph distance, and ring information.

III. Neural Architecture Search

The NAS approach that we propose comprises three components: (i) a search space that defines a set of stacked MPNN feasible architectures; (ii) a search method that will explore the search space to find the best architecture; and (iii) an evaluation method that computes the accuracy of an MPNN architecture sampled using the search method.

A. Stacked MPNN Search Space

We define the MPNN search space as a directed acyclic graph. An example stacked MPNN search space is shown in Fig. 1. The input and output nodes are fixed and denoted as $\mathcal{I}$ and $\mathcal{O}$, respectively. All other nodes $\mathcal{N}$ are intermediate nodes,

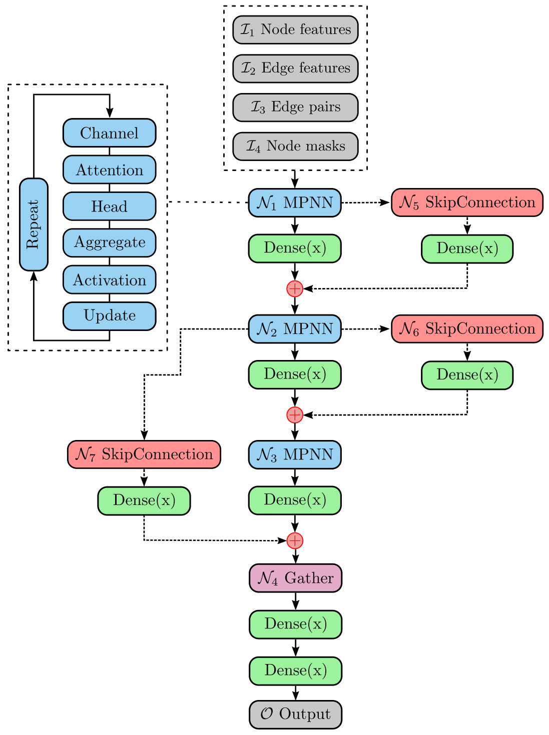

$\mathcal{I}_1$ Node features
$\mathcal{I}_2$ Edge features
$\mathcal{I}_3$ Edge pairs
$\mathcal{I}_4$ Node masks
Repeat
Channel
Attention
Head
Aggregate
Activation
Update
$\mathcal{N}_1$ MPNN
$\mathcal{N}_5$ SkipConnection
Dense(x)
$\mathcal{N}_2$ MPNN
$\mathcal{N}_6$ SkipConnection
$\mathcal{N}_7$ SkipConnection
$\mathcal{N}_3$ MPNN
$\mathcal{N}_4$ Gather
$\mathcal{O}$ Output

Fig. 1: Example stacked MPNN search space with three MPNN variable nodes in blue: $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$. The skip-connection variable nodes are $\mathcal{N}_5$, $\mathcal{N}_6$, and $\mathcal{N}_7$. Dotted lines represent possible skip connections. The gather variable node is $\mathcal{N}_4$. The inputs to the networks are node features $\mathcal{I}_1$, edge features $\mathcal{I}_2$, edge pairs $\mathcal{I}_3$, and node masks $\mathcal{I}_4$. After two constant dense nodes with 32 hidden units, the output node is $\mathcal{O}$. The variables in the MPNN node include the number of channels, the attention mechanisms, the number of attention heads, the aggregation methods, the activation functions, the update functions, and the number of repetitions.

each containing a list of possible operations. Intermediate nodes are made up of two categories: constant node (single operation) and variable node (multiple operations). For a given variable node, an index is assigned for each operation. An architecture from the space can be defined by using a vector $\mathbf{p} \in \mathbb{Z}^n$, where n is the number of variable nodes. Each entry $\mathbf{p}_i$ is an index chosen from a set of possible index values for the variable node i. The stacked MPNN search space is composed of MPNN, skip-connection, and gather variable nodes, described below.

1) Input node: As shown in Fig. 1, the inputs for any given network include node features, edge features, edge pairs, and node masks. For a given molecule dataset, N and E are the maximum number of nodes (atoms) and edges (bonds), respectively. We use zero padding to create the node feature matrix to $\mathbf{H} \in \mathbb{R}^{N \times F_n}$ and the edge feature matrix to $\mathbf{E} \in \mathbb{R}^{E \times F_e}$, where F_n and F_e are the numbers of node features and edge features, respectively. We also have an edge pair matrix $\mathbf{P} \in \mathbb{Z}^{E \times 2}$, where each row contains the indices of two nodes connected by a given edge. Because molecules have a varied number of atoms (nodes), the node mask vector $\mathbf{m} \in \mathbb{Z}^N$ is used to screen out the non-existent node features that are added by zero padding. An existent node has $\mathbf{m}_i = 1$, and a nonexistent node has $\mathbf{m}_i = 0$.

2) MPNN node: Each MPNN node runs internally for T time steps to update the hidden feature of each node. An MPNN node contains a message function M_t and an update function U_t, defined as follows:

$$\mathbf{m}_v^{t+1} = \text{Agg}_{w \in \mathcal{N}(v)} M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right) \quad (1)$$

$$\mathbf{h}_v^{t+1} = U_t\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right). \quad (2)$$

To update the hidden feature of a node v, the message function M_t at step t takes as inputs the node v feature $\mathbf{h}_v^t$, the neighboring node feature $\mathbf{h}_w^t$ for $w \in \mathcal{N}(v)$, and the edge feature $\mathbf{e}_{vw}$ between node v and w. The output of the message function M_t contains a list of message vectors from neighboring nodes. The aggregate function Agg collects the message vectors and generates the intermediate hidden feature $\mathbf{m}_v^{t+1}$. The aggregate function Agg is one of mean, summation, or max pooling. The update function U_t at step t combines the node feature $\mathbf{h}_v^t$ and the intermediate hidden feature $\mathbf{m}_v^{t+1}$ to create the new hidden feature of step $t+1$ $\mathbf{h}_v^{t+1}$. The detailed structures of the message function M_t and update function U_t are as follows.

$$M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right) = \alpha_{vw} \mathsf{MLP}\left(\mathbf{e}_{vw}\right) \mathbf{h}_w^t \quad (3)$$

$$U_t\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right) = \begin{cases} \mathsf{GRU}\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right) \\ \mathsf{MLP}\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right) \end{cases}. \quad (4)$$

Typically, the message function has a multilayer perceptron (MLP or edge network) to handle the edge feature $\mathbf{e}_{vw}$. The processed edge feature is multiplied with $\mathbf{h}_w^t$ to yield a message from node w to v. In this case, the processed edge feature MLP($\mathbf{e}_{vw}$) can be viewed as a weight for $\mathbf{h}_w^t$. Borrowing the idea of node attention, we add an attention coefficient α_{vw} to further modify the weight of $\mathbf{h}_w^t$. The attention coefficient is a function of $\mathbf{h}_v^t$ and $\mathbf{h}_w^t$. The update function U_t can be either a gated recurrent unit (GRU) or an MLP. To further elucidate the design of the MPNN node, we divide it into the following five categories of operations.

1) **State dimension**: After running T times, the MPNN node maps the input node feature to a d-dimensional vector. The choice of state dimension d is important for final prediction. To reduce the number of parameters and increase the generalizability, the set of state dimensions is set to $\{4, 8, 16, 32\}$.
2) **Attention function**: Although the information passing weight between nodes is governed by the edge feature in traditional MPNNs, the attention mechanism helps focus on the most relevant neighboring nodes to improve information aggregation. Following NAS frameworks in [17], [18], the attention functions used to calculate coefficient α_{vw} are shown in Table I. For constant attention, α_{vw} is always 1. For GCN attention, α_{vw} is $\frac{1}{\sqrt{|\mathcal{N}(v)| \cdot |\mathcal{N}(w)|}}$, where $|\mathcal{N}(v)|$ and $|\mathcal{N}(w)|$ represent the number of

neighboring nodes for node v and w, respectively. For GAT, α_{vw} is LeakyReLU $(\mathbf{a}(\mathbf{W}\mathbf{h}_v || \mathbf{W}\mathbf{h}_w))$, where $\mathbf{a}$ is a trainable vector, $\mathbf{W}$ is a trainable weight matrix, $||$ denotes concatenation, and $\mathbf{h}_v$ and $\mathbf{h}_w$ are the node hidden feature vectors for nodes v and w, respectively. By adding α_{vw} with α_{wv} from GAT, we obtain the attention coefficient for SYM-GAT. For COS attention, α_{vw} is $\mathbf{a}(\mathbf{W}\mathbf{h}_v || \mathbf{W}\mathbf{h}_w)$. For linear attention, α_{vw} is $\tanh(\mathbf{a}_l \mathbf{W}\mathbf{h}_v + \mathbf{a}_r \mathbf{W}\mathbf{h}_w)$, where $\mathbf{a}_l$ and $\mathbf{a}_r$ are two trainable vectors. For gen-linear attention, α_{vw} is $\mathbf{W}_G \tanh(\mathbf{W}\mathbf{h}_v + \mathbf{W}\mathbf{h}_w)$, where $\mathbf{W}_G$ is the trainable matrix.

3) **Attention head**: Multihead attention could be useful to stabilize the learning process [25], [26]. We select the number of heads from the set of $\{1, 2, 4, 6\}$.
4) **Aggregate function**: Aggregation function is critical to capture the neighborhood structures for extracting node representation [27]. The aggregation functions are selected from the set of {mean, summation, max-pooling}.
5) **Activation function**: Following NAS frameworks in [17], [18], the possible activation functions used in the MPNN are {Sigmoid, Tanh, ReLU, Linear, Softplus, LeakyReLU, ReLU6, ELU}.
6) **Update function**: Node features $\mathbf{h}_v^t$ and intermediate hidden feature $\mathbf{m}_v^{t+1}$ are combined and propagated by an update function to generate the new feature $\mathbf{h}_v^{t+1}$. The update functions we use are {GRU, MLP}.

TABLE I: Set of attention functions, where $||$ denotes the concatenation; $\mathbf{a}, \mathbf{a}_l, and \mathbf{a}_r$ are the trainable vectors; and $\mathbf{W}_G$ is the trainable matrix

Attention Mechanism	Equations
Constant	1
GCN	$\frac{1}{\sqrt{\lvert\mathcal{N}(v)\rvert\lvert\mathcal{N}(w)\rvert}}$
GAT	LeakyReLU$(\mathbf{a}(\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_w))$
SYM-GAT	$\alpha_{vw} + \alpha_{wv}$ based on GAT
COS	$\mathbf{a}(\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_w)$
Linear	$\tanh(\mathbf{a}_l \mathbf{W}\mathbf{h}_v + \mathbf{a}_r \mathbf{W}\mathbf{h}_w)$
Gen-linear	$\mathbf{W}_G \tanh(\mathbf{W}\mathbf{h}_v + \mathbf{W}\mathbf{h}_w)$

The skip-connection node is a special case of variable node. Given three nodes $\mathcal{N}_{i-1}$, $\mathcal{N}_i$, and $\mathcal{N}_{i+1}$ in a sequence, the skip-connection allows the connection between $\mathcal{N}_{i-1}$ and $\mathcal{N}_{i+1}$. The skip-connection includes two operations: identity for skip-connection and empty for no skip-connection. In a skip-connection operation, the tensor output from $\mathcal{N}_{i-1}$ is processed by a dense layer and a summation operator. Because the output from variable nodes can have different shapes, the dense layer projects the incoming tensor to a right shape for summation. The summation operator adds the output from $\mathcal{N}_{i-1}$ and $\mathcal{N}_i$ and passes the result to $\mathcal{N}_{i+1}$. The skip-connection operation can be applied to any length of node sequences. In our study, we limit the skip-connection to cross at a maximum of three nodes. For example, $\mathcal{N}_{i-1}$ can be added with $\mathcal{N}_{i+2}$ to be passed to $\mathcal{N}_{i+3}$.

3) Gather variable node: The gather node contains eleven operations belonging to five categories. Let $\mathbf{H} \in \mathbb{R}^{N \times F}$ be the node feature input to the gather node, where N is the number of nodes and F is the number of hidden features. Following the gather operations in the Spektral [28] GNN package, the graph operations in stacked MPNN can be divided into five categories.

1) **Global pool**: Pools a graph by computing the sum, mean, or maximum of its node features. The output has a shape of $\mathbb{R}^N$.
2) **Global gather**: Calculates the sum, mean, or maximum of a feature for all the nodes. The output has a shape of $\mathbb{R}^F$.
3) **Global attention pool**: Computes [11] the output $\mathbf{H}_{out} \in \mathbb{R}^{F'}$ as $\mathbf{H}_{out} = \sum_{i=1}^{N} (\sigma(\mathbf{H}\mathbf{W}_1 + \mathbf{b}_1) \odot (\mathbf{H}\mathbf{W}_2 + \mathbf{b}_2))_i$, where σ is the sigmoid activation function; $\mathbf{W}_1, \mathbf{W}_2$ are trainable weights; and $\mathbf{b}_1, \mathbf{b}_2$ are biases. The output dimension F' is selected to be in $\{16, 32, 64\}$.
4) **Global attention sum pool**: Pools a graph by learning attention coefficients to sum node features. The operation can be defined as $\mathbf{H}_{out} = \sum_{i=1}^{N} \alpha_i \cdot \mathbf{H}_i$, where $\alpha = \text{softmax}(\mathbf{H}\mathbf{a})$ and $\boldsymbol{\alpha} \in \mathbb{R}^F$ is a trainable vector. The softmax activation is applied across nodes.
5) **Flatten**: Flattens $\mathbf{H}$ to a 1D vector.

B. Search Methods

To find a high-performing MPNN from the search space, we adopt regularized evolution (RE) [29], an asynchronous search method implemented and available in the DeepHyper package. RE searches for new architectures by applying a mutation to existing models within a population. It starts the search with a population of P random architectures, evaluates them, and records the validation loss from each individual. Following the initialization, it samples S random architectures uniformly from the population with replacement. The architecture with the lowest validation loss within the sample is selected as a parent. A mutation is performed on the parent, and a new child architecture is constructed. A mutation corresponds to choosing a different operation for a variable node in the search space. Specifically, RE randomly samples a variable node with a random operation that excludes the current value. The child is trained, and the validation loss is recorded. Consequently, the child is added to the population by replacing the oldest architecture in the population. Over multiple cycles, architectures with lower validation loss are retained in the population via repeated sampling and mutation. The sampling and mutation are computationally inexpensive and can be performed quickly. When RE finishes an evaluation, a new architecture for training is obtained by the mutation on the previously evaluated architecture (stored throughout the duration of the experiment in memory). The key advantage of RE stems from its scalability. It can leverage multiple compute nodes to evaluate architectures in parallel, which results in faster convergence to high-performing architectures. It has been shown that RE, because of its minimal algorithmic overhead and synchronization, outperforms reinforcement learning methods for NAS [16], [29].

As a baseline, we also include random search (RS) that explores the search space by randomly assigning an operation to each variable node. Specifically, RS samples a random vector of indices to represent the architecture and trains the architecture independently from other nodes. Although the search is parallel, the discovered architectures typically do not improve over time because of the lack of a feedback loop. However, comparison with RS is critical for the new dataset to quantify the effectiveness of the RE search method.

C. Evaluation Strategy

Each evaluation consists of training a generated network and computing the accuracy metric (reward) on the validation data. The evaluation uses a single node (no multinode data-parallel training). Following the implementation of MoleculeNet, the QM7, QM8, and QM9 datasets are evaluated by mean absolute error (MAE); ESOL, FreeSolv, and Lipophilicity are evaluated by the root mean square error (RMSE). RE maximizes the reward; therefore, negative MAE and RMSE are given the reward. To enable rapid exploration of the search space, we use a subset of the training dataset and fewer epochs for training. After the search is over, the best architecture (selected based on the reward) is retrained with the full training data and a larger number of epochs.

IV. Experiments

We used Bebop, a 1,024-node cluster at Argonne's Laboratory Computing Resource Center. Bebop has two types of nodes: Broadwell and Knights Landing. We used Broadwell nodes for the experiments. Each node is a 36-core Intel Xeon E5-2695v4 processor with 128 GB of DDR4 memory. The compute nodes are interconnected by Omni-Path Fabric. The software environment that we used consists of Python 3.7.6, TensorFlow 1.14 [30], DeepHyper 0.1.11, DeepChem 2.4, and Spektral 0.1.2. DeepHyper NAS API utilized TensorFlow Keras.

For the stacked MPNN search space, we used three MPNN variable nodes ($m = 3$), resulting in the creation of six skip-connection variable nodes. We used one gather variable node and two constant dense nodes with 32 hidden units between gather node and prediction. The total number of architectures in the search space is 23,626,761,124,184,064 ($\approx 2 \times 10^{16}$).

As shown in Fig. 1, the inputs are node features, edge features, edge pairs, and node masks. We use zero padding to create the node feature matrix to $\mathbf{H} \in \mathbb{R}^{N\times 75}$ and the edge feature matrix to $\mathbf{E} \in \mathbb{R}^{E\times 14}$, where N is the number of nodes, E is the number of edges, 75 is the number of node features, and 14 is the number of edge features. Our edge pair matrix $\mathbf{P} \in \mathbb{Z}^{E\times 2}$ contains the source and target node indices of the edges. The graph can be traversed from the source node to the target node. We treat molecules as undirected graphs, where edges are bidirectional. Specifically, the graph can be traversed from node i to node j, as well as from node j to node i. We also add self-loop, which is an edge that connects a node with itself. The node mask vector $\mathbf{m} \in \mathbb{Z}^{N}$ has entries of 1 when nodes exist and 0 when nodes do not exist. The optimizer for training was ADAM [31].

For the split of training, validation, and test data, we followed the MoleculeNet implementation and used the given stratified splitter to split the QM7 dataset and the random splitter to split other datasets using fixed random seeds. The training, validation, and test split ratio is 8:1:1. Specifically, QM7 has 5,728 training, 716 validation, and 716 test data. The maximum number of nodes N is 7, and the maximum number of edges E is 9. QM8 has 17,428 training, 2,179 validation, and 2,179 test data with $N = 9$ and $E = 14$. QM9 has 107,107 training, 13,389 validation, and 13,389 test data with $N = 9$ and $E = 16$. ESOL has 902 training, 113 validation, and 113 test data with $N = 55$ and $E = 68$. FreeSolv has 512 training, 65 validation, and 65 test data with $N = 24$ and $E = 25$. Lipophilicity has 3,360 training, 420 validation, and 420 test data with $N = 115$ and $E = 236$.

RE was configured to run with a population size (P) of 100 and a sample size (S) of 10 to process the mutation of architectures. RS evaluated a random architecture on each node. Since both RE and RS were asynchronous, all nodes were workers that could evaluate architectures independently without any master.

MoleculeNet has various benchmark models, including random forest (RF), graph convolution (GC), deep tensor network (DTNN), message-passing neural network (MPNN), and Weave network. Only MPNN and Weave network use Weave featurizer to represent molecule data with both atom (node) and node (edge) information. Because stacked MPNN uses the same Weave featurizer to generate data, we compared it with the better MPNN and Weave network from MoleculeNet.

For the evaluation, we limited the training time for each architecture to less than 10 minutes. QM8 and Lipophilicity training data were halved. QM9 training data were shrunk to one-tenth. Other data sets had full training data. QM7, QM8, and QM9 were trained for 50, 40, and 50 epochs, respectively. ESOL, FreeSolv, and Lipophilicity were trained for 80, 200, and 20 epochs, respectively. For post-training, we selected the architecture discovered to have the lowest validation loss and trained it from scratch for 200 epochs using the full dataset. Following the implementation of MoleculeNet, three independent runs with different random seeds were performed. The three fixed randoms seeds were used to split the dataset into training, validation, and testing sets. The results were the average accuracy metrics of three runs with standard deviations as errors. MoleculeNet provided various data representations for molecules. Each MoleculeNet featurizer could generate a unique data representation. Both stacked MPNN and MoleculeNet GNN benchmark results were based on the same data representation generated by the Weave featurizer, except for the QM7 dataset, for which the MoleculeNet benchmark used the GCN featurizer to generate molecule representations without edge features.

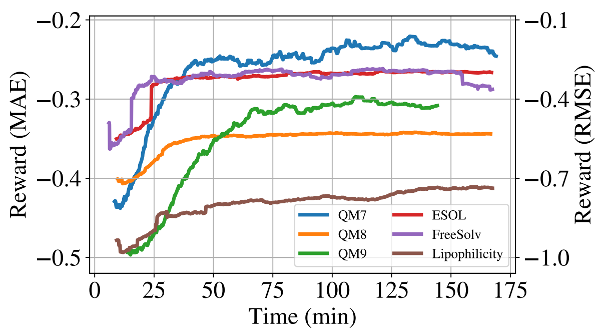


Fig. 2: Search trajectories for all datasets. The reward for QM7, QM8, and QM9 is the negative validation MAE. The reward for ESOL, FreeSolv, and Lipophilicity is the negative validation RMSE.

A. *Neural Architecture Search Using Regularized Evolution*

Here, we show that RE discovers high-performing architectures within short computation times and that these architectures outperform the manually designed baseline GNN architectures from MoleculeNet on the six datasets.

We ran the RE search method on 30 nodes for 3 hours of wall-clock time. Fig. 2 shows the search trajectory of reward (negative MAE/RMSE) with respect to wall-clock time. We calculated the search trajectory by using a running average of window size 100 on the reward obtained by the architectures found during the search. For the quantum mechanics datasets (QM7, QM8, and QM9), RE converged after 50, 40, and 75 minutes, respectively. The longer convergence time on QM9 can be attributed to the large dataset size, which made each individual training slower. For the physical chemistry datasets (ESOL, FreeSolv, and Lipophilicity), RE converged after 25, 20, and 75 minutes, respectively. Because ESOL and FreeSolv were small datasets, they were expected to have shorter convergence times. The Lipophilicity dataset had more samples, and each sample was larger, thus making the convergence slower.

The single-class regression results for each test dataset are shown in Table II. The MAE of stacked MPNN on QM7 is 48.0±0.7, which is a 38% drop from the MoleculeNet MAE of 77.9±2.1. The RMSEs of stacked MPNN on ESOL and Lipophilicity are 0.54±0.01 and 0.598±0.043, respectively, which are 7% and 16% lower than the MoleculeNet results of 0.58±0.03 and 0.715±0.035. MoleculeNet has a lower RMSE of 1.15±0.12 on FreeSolv compared with 1.21±0.03 on stacked MPNN. FreeSolv is a small dataset with 64 molecules in each test set. The difference of random seeds used in MoleculeNet and the over-fitting problem brought on by a small dataset can lead to a worse result for stacked MPNN.

QM8 and QM9 are multiclass regression datasets. As shown in Table III, stacked MPNN has a smaller MAE for each feature in the multiclass regression of QM8 compared with MoleculeNet benchmarks. Moreover, Table III illustrates that for most of the features, stacked MPNN outmatches MoleculeNet GNN except for mu and R2. In particular, the R2 value is

TABLE II: Single-class regression test set performance comparison

Data Set	Stacked MPNN	MoleculeNet GNN
QM7 (MAE)	**48.0±0.7**	77.9±2.1
ESOL (RMSE)	**0.54±0.01**	0.58±0.03
FreeSolv (RMSE)	1.21±0.03	**1.15±0.12**
Lipophilicity (RMSE)	**0.598±0.043**	0.715±0.035

TABLE III: QM8 and QM9 test set performance comparison (MAE)

QM8	Stacked MPNN	Molecule Net GNN	QM9	Stacked MPNN	Molecule Net GNN
E1-CC2	**0.0068±0.0003**	0.0084	mu	0.564±0.003	**0.358**
E2-CC2	**0.0079±0.0003**	0.0091	alpha	**0.69±0.01**	0.89
f1-CC2	**0.0129±0.0002**	0.0151	HOMO	**0.00560±0.00004**	0.00541
f2-CC2	**0.0291±0.0005**	0.0314	LUMO	**0.00602±0.00002**	0.00623
E1-PBE0	**0.0064±0.0001**	0.0083	gap	**0.0080±0.0000**	0.0082
E2-PBE0	**0.0072±0.0001**	0.0086	gap	**0.0080±0.0000**	0.0082
f1-PBE0	**0.0104±0.0002**	0.0123	gap	**0.0080±0.0000**	0.0082
f2-PBE0	**0.0216±0.0005**	0.0236	gap	**0.0080±0.0000**	0.0082
E1-CAM	**0.0063±0.0001**	0.0079	R2	41.3±0.6	**28.5**
E2-CAM	**0.0069±0.0002**	0.0082	ZPVE	**0.00130±0.00014**	0.00216
f1-CAM	**0.0117±0.0002**	0.0134	ZPVE	**0.00130±0.00014**	0.00216
f2-CAM	**0.0236±0.0001**	0.0258	U0	**0.65±0.06**	2.05
			U	**0.62±0.05**	2.00
			H	**0.68±0.11**	2.02
			G	**0.66±0.05**	2.02
			Cv	**0.35±0.01**	0.42

at least a magnitude larger than other features. The suboptimal loss value at which the training was stuck could lead to the inconsistency in the R2 value [32].

B. *Comparison between RE and RS*

Here, we compare RE to RS and study their scaling behavior. We show that RE outperforms RS with respect to search trajectory, number of architectures evaluated, and number of high-performing architectures. In the following study, we focused on running both search methods on 15, 30, and 60 nodes for QM8, QM9, ESOL, and Lipophilicity.

1) Search trajectory: From two example Figs. 3a and 3c, we observe that RE converges to a better search reward for all datasets with 15, 30, and 60 compute nodes. However, RS without any feedback mechanism fails to find high-performing architectures. Specifically, the RS search rewards for QM8, QM9, ESOL, and Lipophilicity are from -0.4 to -0.37, -0.51 to -0.47, -0.7 to -0.4, and -1.1 to -0.9, respectively; these rewards are worse than their corresponding RE results, which converge to between -0.35 and -0.31, -0.33 and -0.29, -0.31 and -0.3, and -0.82 and -0.64, respectively. In general, with an increasing number of nodes, RE search trajectories converge faster to a better search reward. For example, the RE search reward for QM8 reaches -0.36 at 53, 28, and 22 minutes with 15, 30, and 60 nodes, respectively.

2) Number of architectures evaluated: The number of evaluations for RE and RS on a varying number of nodes for a fixed wall-clock time can be found in Table. IV. For all datasets, given the same number of nodes, the number of architectures evaluated is similar. For instance, RE evaluates 563 architectures with 30 nodes for QM8, whereas RS evaluates 540 architectures. With more nodes, however, the

TABLE IV: Total number of evaluations for RE and RS on varying number of compute nodes

	QM8		QM9		ESOL		Lipo	
No. of nodes	RE	RS	RE	RS	RE	RS	RE	RS
15	219	239	282	295	284	337	356	368
30	563	540	726	694	742	775	773	667
60	1490	1237	1393	1474	1880	1647	1851	1748

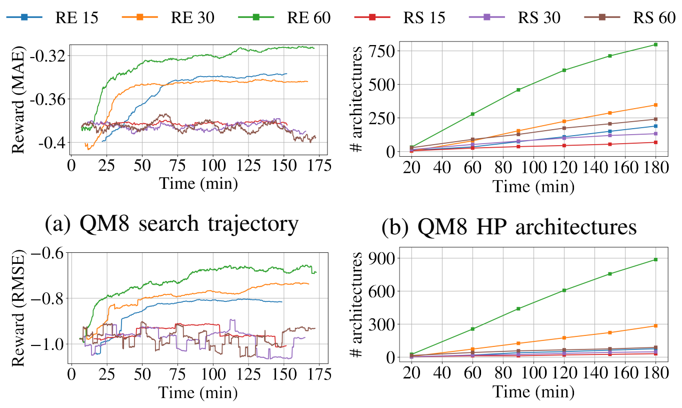
number of evaluations increases significantly. Specifically, RE for QM8 performs 219, 563, and 1490 evaluations with 15, 30, and 60 nodes. Similarly, RS for QM8 leads to 239, 540, and 1237 evaluations. The same trend is also seen for QM9, ESOL, and Lipophilicity. The difference in the number of evaluations between the two methods is caused by the training time variation for each architecture. For example, with 60 nodes, RE performs 253 more evaluations than RS does for QM8, but 81 less for QM9. The variation is because RE discovers more architectures that require less training time for QM8 than RS does. Unlike for QM8, RE for QM9 finds more architectures that are trained for a longer time than RS does.

3) High-performing architectures discovered: In two example Figs. 3b and 3d, we demonstrate the temporal breakdown of the number of unique high-performing architectures discovered by RE and RS on varying numbers of nodes. We define high-performing architectures as any architecture with a higher search reward than a given threshold value. The threshold values are -0.35, -0.3, -0.32, and -0.77 for QM8, QM9, ESOL, and Lipophilicity, respectively. The figure shows that the number of unique high-performing architectures grows considerably with a greater number of nodes for RE but not for RS. For example, the number of such architectures obtained by RE for QM8 at 180 minutes using 30 nodes is achieved by RE with 60 nodes in 70 minutes. We observe a similar trend for all datasets. Without a feedback mechanism, RS discovered a limited number of high-performing architectures, and the number increases insignificantly with more nodes. Specifically, after 180 minutes of search, RS for QM8 discovers 68, 132, and 240 high-performing architectures for 15, 30, and 60 nodes, respectively. In comparison, RE for QM8 finds 189, 346, and 796 architectures.

V. Related Work

For a detailed review of NAS for graphs, we refer the reader to [33]. Here, we review prior works related to GNNs and NAS.

Graph Neural Networks. A wide variety of GNNs have been invented to study node embeddings. Most of these can be viewed as MPNNs with information passed from one node to another. GCN [34] focuses on node information only, and the information passed between any two nodes is equally weighted. GAT [25] calculates an attention coefficient as a weight for information passed between two nodes based on the properties of neighboring nodes. By incorporating edge features into message passing, RGCN [35], GGNN [11], and LanczosNet [36] have better performance but lack the flexibility for continuous edge information. The formal MPNNs



(a) QM8 search trajectory (b) QM8 HP architectures

(c) Lipo. search trajectory (d) Lipo. HP architectures

Fig. 3: Comparison of search trajectories between RE and RS and the temporal breakdown of the number of unique high-performing (HP) architectures discovered by RE and RS with 15, 30, and 60 compute nodes. On all the datasets, the RE search strategy is more effective at obtaining high-performing architectures.

[8] train the edge features to govern the node information passing. Our work combines edge information and attention mechanisms to further guide the node information passing.

Neural Architecture Search. Several studies of automated GNN architecture searches have been conducted for a diverse set of applications. GraphNAS, proposed in [18], applies RNN directly to search GNN structures; the whole neural architecture is sampled and reconstructed in GraphNAS. AGNN [17], in contrast, explores the offspring architecture by modifying only a specific operation class; the best architectures are retained to provide a good start for architecture modification. Both works contain state dimension, attention function, attention head, aggregate functions, and activation functions in the search space. The merge or combine function is similar to the update function in MPNNs. However, both works lack support for edge features. Our work not only embeds edge information but also enables skip-connection and modification to the graph gather node.

VI. Conclusion and Future Work

We introduced a neural architecture for the automated development of stacked MPNNs to predict the molecular properties of small molecules from the MoleculeNet benchmark dataset. This is the first work that employs NAS for MPNNs and establishes the first graph-related search space to incorporate edge features with skip connections.

We developed a MPNN search space that comprises MPNN nodes and skip connections. The MPNN node comprises various state dimensions, attention functions, attention head, aggregate function, activation functions, update functions, and gather operations. As a search strategy, we utilized RE, an asynchronous evolutionary algorithm within DeepHyper, an open-source automated machine learning package. We compared RE with RS in DeepHyper and showed that RE outperforms RS in terms of the number of high-performing architectures discovered. In addition, RE achieves architectures with better performance in a shorter wall-clock time and matches the scalability of the completely asynchronous RS. We compared the manually designed MoleculeNet benchmarks with the best architectures obtained using RE that was retrained for a longer number of epochs. We showed that the automatically designed architecture outperformed the baselines with respect to loss on the validation and test data.

Our future work will seek to overcome the limitations of processing large molecules by transforming the search space from Keras to TensorFlow 2.0, allowing us to eliminate the padding of input data. Moreover, we will develop a search space for graph autoencoders for generative modeling. The results from this work also show promise for continued investigation into AutoML-enhanced, data-driven surrogate discovery for scientific machine learning.

ACKNOWLEDGMENT

This material is based on work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided on Bebop, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

REFERENCES

[1] R. Burbidge, M. Trotter, B. Buxton, and S. Holden, "Drug design by machine learning: support vector machines for pharmaceutical data analysis," *Computers & chemistry*, vol. 26, no. 1, pp. 5–14, 2001.

[2] S. X. Tao, X. Cao, and P. A. Bobbert, "Accurate and efficient band gap predictions of metal halide perovskites using the dft-1/2 method: Gw accuracy with dft expense," *Scientific reports*, vol. 7, no. 1, pp. 1–9, 2017.

[3] A. D. Sendek, E. D. Cubuk, E. R. Antoniuk, G. Cheon, Y. Cui, and E. J. Reed, "Machine learning-assisted discovery of solid li-ion conducting materials," *Chemistry of Materials*, vol. 31, no. 2, pp. 342–352, 2018.

[4] M. Mascagni and N. A. Simonov, "Monte carlo methods for calculating some physical properties of large molecules," *SIAM journal on scientific computing*, vol. 26, no. 1, pp. 339–357, 2004.

[5] V. Varshney, S. S. Patnaik, A. K. Roy, and B. L. Farmer, "A molecular dynamics study of epoxy-based networks: cross-linking procedure and prediction of molecular and material properties," *Macromolecules*, vol. 41, no. 18, pp. 6837–6842, 2008.

[6] A. D. Becke, "Perspective: Fifty years of density-functional theory in chemical physics," *The Journal of chemical physics*, vol. 140, no. 18, p. 18A301, 2014.

[7] C. Lu, Q. Liu, C. Wang, Z. Huang, P. Lin, and L. He, "Molecular property prediction: A multilevel quantum interactions modeling perspective," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1052–1060.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[9] L. C. Blum and J.-L. Reymond, "970 million druglike small molecules for virtual screening in the chemical universe database GDB-13," *J. Am. Chem. Soc.*, vol. 131, p. 8732, 2009.

[10] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.

[11] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[12] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in neural information processing systems*, 2016, pp. 4502–4510.

[13] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *arXiv preprint arXiv:1703.01041*, 2017.

[14] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 82–92.

[15] Y. Fan, F. Tian, Y. Xia, T. Qin, X.-Y. Li, and T.-Y. Liu, "Searching better architectures for neural machine translation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2020.

[16] R. Maulik, R. Egele, B. Lusch, and P. Balaprakash, "Recurrent neural network architecture search for geophysical emulation," *arXiv preprint arXiv:2004.10928*, 2020.

[17] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-gnn: Neural architecture search of graph neural networks," *arXiv preprint arXiv:1909.03184*, 2019.

[18] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," *arXiv preprint arXiv:1904.09981*, 2019.

[19] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[20] P. Balaprakash, M. Salim, T. Uram, V. Vishwanath, and S. Wild, "Deephyper: Asynchronous hyperparameter search for deep neural networks," in *2018 IEEE 25th international conference on high performance computing (HiPC)*. IEEE, 2018, pp. 42–51.

[21] L. Ruddigkeit, R. Van Deursen, L. C. Blum, and J.-L. Reymond, "Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17," *Journal of chemical information and modeling*, vol. 52, no. 11, pp. 2864–2875, 2012.

[22] J. S. Delaney, "Esol: estimating aqueous solubility directly from molecular structure," *Journal of chemical information and computer sciences*, vol. 44, no. 3, pp. 1000–1005, 2004.

[23] D. L. Mobley and J. P. Guthrie, "Freesolv: a database of experimental and calculated hydration free energies, with input files," *Journal of computer-aided molecular design*, vol. 28, no. 7, pp. 711–720, 2014.

[24] M. Davies, M. Nowotka, G. Papadatos, N. Dedman, A. Gaulton, F. Atkinson, L. Bellis, and J. P. Overington, "Chembl web services: streamlining access to drug discovery data and utilities," *Nucleic acids research*, vol. 43, no. W1, pp. W612–W620, 2015.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[26] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 13, no. 6, pp. 1–25, 2019.

[27] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[28] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral," *arXiv preprint arXiv:2006.12138*, 2020.

[29] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

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

[32] G. Chen, P. Chen, C.-Y. Hsieh, C.-K. Lee, B. Liao, R. Liao, W. Liu, J. Qiu, Q. Sun, J. Tang *et al.*, "Alchemy: A quantum chemistry dataset for benchmarking ai models," *arXiv preprint arXiv:1906.09427*, 2019.

[33] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *arXiv preprint arXiv:1908.00709*, 2019.

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

[35] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[36] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel, "Lanczosnet: Multi-scale deep graph convolutional networks," *arXiv preprint arXiv:1901.01484*, 2019.