



# AutoTGRL: an automatic text-graph representation learning framework

Raeed Al-Sabri<sup>1</sup> · Jianliang Gao<sup>1</sup> · Jiamin Chen<sup>1</sup> · Babatounde Moctard Oloulade<sup>1</sup> · Tengfei Lyu<sup>1</sup>

Received: 3 December 2022 / Accepted: 29 October 2023 / Published online: 8 December 2023  
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2023

## Abstract

Text-graph representation learning is a critical and important area of research with extensive applications in natural language processing (NLP). Recently, graph learning models based on graph neural networks (GNNs) have been effectively utilized for encoding text-graph representation for various tasks due to their ability to handle complex structures and capture global information. However, existing text-graph representation learning models are heavily based on the manual design of model architectures and fine-tuning hyperparameters, which is time-consuming and relies on expert knowledge. To address this challenge, we propose an automatic text-graph representation learning (AutoTGRL) framework for transductive and inductive learning-based downstream tasks on text graphs. Specifically, the AutoTGRL framework first builds a general text-graph representation learning model (TGRL model) for text-graph transductive and inductive learning. Then, to enable the automatic design of TGRL models, we propose an automated TGRL model search module. In the automated TGRL model search module, we propose an effective and customized search space called text-graph representation learning (TGRL) search space, which consists of three subspaces, including large-scale embedding strategy space, text-graph representation strategy space, and GNN structure and hyperparameter space, to build TGRL models. We propose to use a search algorithm to search for the best combinations to construct TGRL models to fulfill different downstream tasks from the TGRL search space. To demonstrate the effectiveness AutoTGRL framework, we apply it to text classification, aspect-based sentiment analysis (ABSA), and entity and relation extraction tasks. The extensive experiments demonstrate the superiority of AutoTGRL to design the optimal TGRL models, which outperform the state-of-the-art models over multiple datasets.

**Keywords** Graph neural network · Automatic machine learning · Text-graph representation · Natural language processing

## 1 Introduction

Text-graph representation learning is one of the most significant problems in the field of natural language processing (NLP) because it provides essential methodologies for a wide spectrum of a diverse range of applications in the real world, such as document organization, spam detection, opinion mining, topic modeling, text classification, sentiment analysis, entity extraction, and relation extraction. [1–5]. In the recent past, graph neural networks (GNNs) [6, 7] have been getting more attention and illustrated their superior results on the text-graph representation learning for various tasks [8, 9]. By constructing text graphs using different representation methods [10], GNN-based models can capture non-consecutive and long-distance interactions among tokens that lead to the improvement of downstream

---

✉ Jianliang Gao  
gaojianliang@csu.edu.cn

Raeed Al-Sabri  
alsabriraeed@csu.edu.cn

Jiamin Chen  
chenjiamin@csu.edu.cn

Babatounde Moctard Oloulade  
oloulademoctard@csu.edu.cn

Tengfei Lyu  
tengfeilyu@csu.edu.cn

<sup>1</sup> School of Computer Science and Engineering, Central South University, Changsha 410083, Hunan, China

task performance. In text-graph representation learning, documents and tokens are modeled as nodes and language properties are modeled as edges. There are two different kinds of GNN-based models for text-graph representation learning: transductive-based learning models and inductive-based learning models. The transductive-based learning models [3, 8, 11] build a graph containing documents and tokens as nodes for the entire text corpus. They collect global information about a corpus and perform node (document) categorization. The inductive-based learning models [12, 13] construct a graph for each document using tokens as nodes. In the inductive learning model, no test set information should be included during the training phase.

Although the promising results of GNN-based models in text graph representation learning tasks, they heavily rely on the manual construction of model architectures and hyperparameter optimization. To achieve high performance, they require costly human efforts and domain knowledge for tuning a vast number of text-graph representation learning model (TGRL model) components, including text-graph representation learning strategies, GNN structures, hyperparameters, and text embedding strategies. When applied to a new dataset, the manually constructed TGRL models often have low performance and need an extremely high number of design trials. The primary advantages and drawbacks of the most recent GNN-based TGRL models are outlined in Table 1. Because of the advancements made in automated machine learning, the methodologies associated with automatic machine learning have been effectively used in an increased number of scenarios [14]. Consequently, we are able to overcome the difficulty of developing TGRL models and achieve optimum performance on a variety of downstream tasks.

In this paper, we made the first attempt to address the problem of manually designing a text-graph representation learning model (TGRL model) and propose the automatic text-graph representation learning (AutoTGRL) framework, which can automatically achieve text-graph

representation learning for downstream tasks on different text datasets. The designed AutoTGRL framework consists of five main modules: large-scale text embedding, transductive graph-based learning, inductive graph-based learning, model evaluation, and automated TGRL model search modules. Specifically, the large-scale text embedding module computes document embeddings using a large-scale embedding strategy (e.g., BERT-style models). Then, the transductive graph-based learning module first builds a transductive text graph based on a text-graph representation learning strategy and learns the node embeddings of the constructed graph using GNNs. In the inductive graph-based learning module, AutoTGRL builds the inductive text graphs based on a text-graph representation learning strategy and learns the node embeddings of each graph. Then, the module integrates node embeddings to generate the graph-level embeddings. The transductive graph-based learning and inductive graph-based learning modules utilize the document embeddings from the large-scale text embedding module as an initial embedding of the constructed graph nodes. The model evaluation module receives the text representation from the transductive graph-based learning or inductive graph-based learning modules and evaluates the searched TGRL model on downstream tasks. This module applies the task-specific layers (prediction layers) on the received text representation and computes the task performance on the validation set and returns it as reward feedback to the search algorithm. The large-scale text embedding, transductive graph-based learning, and model evaluation modules construct the TGRL model based on transductive learning. While the large-scale text embedding, inductive graph-based learning, and model evaluation modules construct the TGRL model based on inductive learning. To enable the automatic design of TGRL models, we propose an automated TGRL model search module. In the automated TGRL search, we design an effective text-graph representation learning search space called TGRL search space, which

**Table 1** Advantages and drawbacks of the most recent GNN-based TGRL models

Advantages	Drawbacks
Graph embeddings keep the global information about the structure of a graph	They require expert knowledge to design model architectures
They are efficient in undertaking tasks that are believed to have a complex relational structure	They are time-consuming to fine-tune hyperparameters
They have the capability of learning the syntactic structure of a variety of texts	When applied to new datasets, they tend to have lower performance
They are able to achieve cutting-edge performance	
They are able to deal with complicated structures and gather information on a global scale	

consists of three sub-spaces, namely, the large-scale embedding strategy space, GNN structure and hyperparameters space, and text-graph representation learning strategy space. Then, we use a search algorithm based on reinforcement learning to obtain the optimal combination from the TGRL search space to build the TGRL model to achieve the downstream tasks. To examine the effectiveness AutoTGRL framework, we evaluate it on text classification and aspect-based sentiment analysis, and entity and relation extraction tasks based on text-graph representation learning. The following is a summary of our work's contributions:

1. For the first time, we propose the AutoTGRL framework to automatically design a text-graph representation learning model for transductive and inductive graph-based learning on different text-graph data. Based on the proposed AutoTGRL framework, we design an effective TGRL search space for text-graph representation learning modeling. Then, a search algorithm is adapted to identify the optimal combinations of the TGRL model in the designed search space.
2. Extensive experiments on nine real-world text datasets demonstrate that AutoTGRL outperforms the state-of-the-art baseline methods on text classification, aspect-based sentiment analysis, and entity and relation extraction tasks.

## 2 Related work

This section introduces the related work about text-graph representation learning, GNNs for text-graph representation learning, and neural architecture search.

### 2.1 Text-graph representation

Text-graph representation is one of the essential research studies in natural language processing (NLP) for various applications such as text mining, text classification, sentiment analysis, and information retrieval [10]. Various methods have been developed to transform text to a graph  $G = (V, E)$ , where  $V$  is a set of nodes (tokens/documents) and  $E$  is a set of graph edges (token–token relation or token–document relation) [9, 15, 16]. They attempt to exploit the best features of text document characteristics. In natural language processing, there are three main text-graph representation strategies to present token–token relationships between two tokens. The first type of token–token relationship is N-grams based on co-occurrence relationship, also called sequential-based relationship [12], in which each edge between two paired tokens indicates the n-hop co-occurrence relationships between two tokens

across several documents. In this type, tokens are supposed to appear with each other within a certain window. The second type is a syntactical-dependent relationship, also called syntactic-based relationship, in which the connection between two tokens illustrates the grammatical dependent relation within each sentence [17]. External syntactical dependency parsing tools such as Stanford CoreNLP, NLKT, etc. are used. The third type of relationship between two tokens is the semantic similarity based relationship [8]. This type calculates the semantic similarity among pairwise tokens. Utilizing similarity-based measures such as cosine similarity, point-wise mutual information (PMI), etc. [11], the semantic weighting scores between two tokens are calculated. The token–document relation is used to normally present the occurrence relationships between a specific token and a document. In the token–document relation, the information on the document–token level is utilized to examine the representations of tokens and documents. In our work, we build text graphs based on the automatic search for text-graph representation strategy. The search algorithm identifies the suitable text-graph representation strategy from a designed TGRL search space.

### 2.2 GNNs for text-graph representation learning

Graph neural networks (GNNs) have made significant progress in text-graph representation learning due to the fast growth of deep learning techniques [13, 18]. In general, the majority of common GNN models use neighborhood aggregation approaches, and a GNN layer may be defined as follows:

$$h_v^l = \text{AG}^l(h_v^{l-1}, \{h_u^{l-1} | \forall u \in N_v\}) \quad (1)$$

where  $h_v^l$  is the node representation of node  $v$  at layer  $l$  and  $N_v$  are the local neighbors of node  $v$ . AG is an aggregation operation, which has various implementations [7].

GNNs have shown promising performance on text-graph representation for a variety of tasks such as text classification, entity and relation extraction tasks, and aspect-based semantic analysis tasks due to their capacity of capturing long-distance relations between nodes [9, 11, 15, 19, 20]. Specifically, [11] applies graph convolutional networks GCNs on a heterogeneous token–document graph built for the whole corpus, which has the ability to capture the global token co-occurrence information and achieves better results. Later on, [15] proposes a simple graph convolution (SGC) that reduces the computation complexity by removing unnecessary nonlinearity computation, and the weight matrices are collapsed between layers. SGC shows better performance with superior time efficiency. By integrating more context

information such as semantic and syntactic dependencies between tokens, TensorGCN [8] presents a text graph tensor. BiGCN [21] employs a hierarchical graph structure to combine data on token co-occurrences and dependency type information for aspect-level sentiment analysis. By considering tokens as nodes and the relation neighboring tensor as edges, EMC-GCN [22] converts the phrase into a multi-channel graph. GNN-based models have been used successfully in learning node representations and achieved good performance. However, designing TGRL architectures and fine-tuning the learning parameters in the existing methods is done manually, which requires more effort and domain knowledge. In addition, it is difficult to design a TGRL model that is appropriate for different types of text datasets.

### 2.3 Neural architecture search

Neural architecture search (NAS) [23, 24] aims to automatically search better architectures than the expert-designed ones, which have achieved promising performance in designing architectures for recurrent neural network (RNN) and convolution neural network (CNN). To accelerate the reinforcement learning procedure that trains the controller to generate child networks, ENAS [25] used the shared parameters among child trials; DARTS [26] provides a differentiable manner for formulating the task of neural architecture search and does not need reinforcement learning controllers; when generating model parameters, a hyper-network may be used to predict optimal values, SMASH [27] presents a one-shot model architecture search. More recently, studies attempt to automatically search GNN architecture such as reinforcement learning-based approaches and evolutionary-based approaches [14, 28]. Among them, graph neural architecture search (GNAS) [29] has been used to the creation of automated GNN models with great success. Precisely, GNAS utilizes a search algorithm (reinforcement learning-based algorithm) to investigate search space and selects the top architecture in terms of performance (e.g., accuracy on validation set) on a specific task. However, the existing graph neural architecture search frameworks does not consider designing text-graph representation learning model based on graph neural networks. In addition, this is the first work to automatically design an optimization pipeline for text-graph representation learning models.

## 3 Proposed framework AutoTGRL

AutoTGRL is an automatic text-graph representation learning framework for modeling text graphs for transductive and inductive graph-based learning. As shown in

Fig. 1, the AutoTGRL framework consists of five key modules, including large-scale text embedding, transductive graph-based learning, inductive graph-based learning, model evaluation, and automated TGRL model search modules. In this section, we first describe the problem of automatically learning text-graph representations. Then, we introduce each module of the AutoTGRL framework in detail.

### 3.1 Problem definition

A TGRL search space for a text-graph representation learning model consists of three sub-spaces: the large-scale embedding strategy space, GNN structure and hyperparameters space, and text-graph representation strategy space. TGRL search space is represented as  $S \in R^{|S_1| \times |S_2| \times \dots \times |S_t|}$ , where  $S_{i=1,2,3,\dots,t} \in R^{|S_i|}$  is the set of potential choices for the  $i^{\text{th}}$  TGRL model component (e.g., text-graph representation strategy).  $t$  is the number of components required for TGRL models  $\mathcal{M}$ , such that a structured model  $m = \{S_1, S_2, \dots, S_t\}$ .

Given a designed TGRL search space and text dataset, our aim is to find the best TGRL model from the designed TGRL search space,

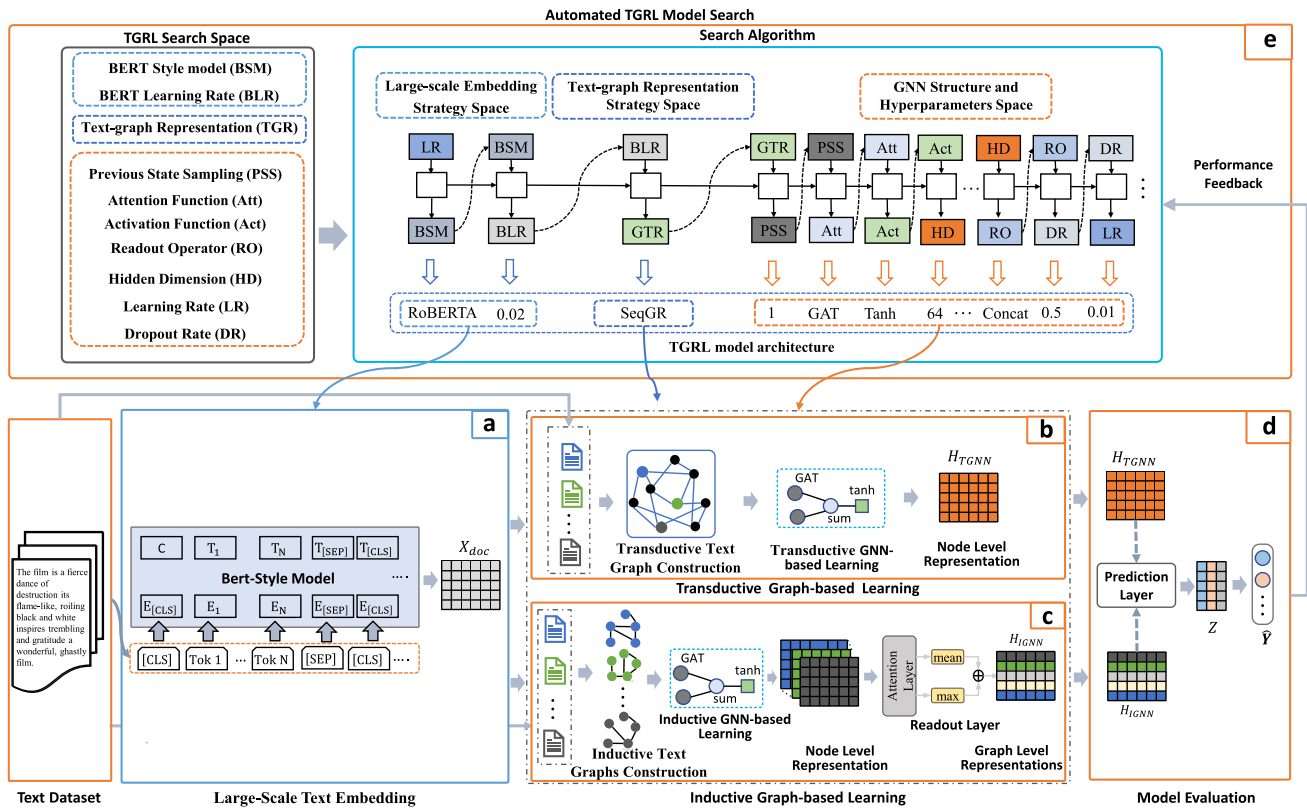
$$m^{\text{best}} = \{S_1^{\text{best}}, S_2^{\text{best}}, \dots, S_t^{\text{best}}\}$$

which maximizes the estimated performance  $E[R_V(m)]$  on validation set  $V$ , i.e.,

$$m^{\text{best}} = \arg \max_{m \in \mathcal{M}} E[R_V(m)]. \quad (2)$$

### 3.2 Large-scale text embedding

Recent studies in text representation learning show that large-scale pre-trained text embedding can significantly enhance the performance of downstream tasks and speed up the convergence of the network with better generalization [30]. The large-scale text embedding module is designed to generate the effective initial representation of document nodes. The large-scale pretraining model has proven their effectiveness on a broad variety of different NLP tasks [31, 32]. They are able to acquire the complex text semantics that is inherent in language at a large scale. They are trained on large-scale unlabeled corpora in an unsupervised manner and have the potential to benefit text-graph representation learning. Obtaining effective document embeddings depends on the used large-scale pretraining embedding model, which varies from one text dataset to another [30]. So, choosing the right large-scale embedding model for a dataset and a certain task is an important step, which has not been taken into consideration in the existing text-graph representation learning models.



**Fig. 1** AutoTGRl framework. The automated TGRl model search module **e** utilizes a search algorithm to identify the description of the TGRl model architecture from the proposed TGRl search space. In the large-scale text embedding module **a**, document embeddings are computed by a large-scale text embedding method obtaining initial document embeddings  $X_{doc}$ . The transductive graph-based learning module **b** first constructs a transductive text graph based on a text-graph representation strategy and learns the graph node embeddings  $H_{TGNN}$  by the GNN architecture. The inductive graph-based learning

module **c** constructs the inductive text graphs based on a text-graph representation strategy and learns the node embeddings of each graph by the GNN architecture. Then, the module incorporated the node embeddings to form the graph-level embeddings  $H_{IGNN}$ . Then, the model evaluation module **d** applies the evaluation based on the downstream task on the validation set and returns the performance score as feedback to the search algorithm, which supervises the search direction in the TGRl search space to sample the optimal TGRl model

In the large-scale text embedding module, we utilize large-scale embedding models to produce the document and token embeddings, which are then used as input representations for document nodes in the text-graph representation.

To make AutoTGRl choose the appropriate large-scale pretraining model to obtain effective document embeddings, we design a large-scale embedding strategy search space. The search algorithm will search the optimal large-scale pretraining strategy from the TGRl search space based on the feedback performance. Given a text dataset of  $N$  documents  $T = \{d_1, d_2, \dots, d_N\}$ , we calculate the document embeddings as follows:

$$X_{doc} = f(T) \tag{3}$$

where  $T = \{d_1, d_2, \dots, d_N\}$  is a text dataset of  $N$  documents, and  $f$  is the large-scale pretraining embedding model drawn from the TGRl search space using the search algorithm.

To speed up the running time of our model, we fine-tune document embeddings using different large-scale embedding models with few epochs. Then, during the search process of the models, we just need to read the document embeddings based on the large-scale embedding strategy sampled from the TGRl search space.

### 3.3 Transductive graph-based learning

In the transductive graph-based learning module, we first construct the transductive text graph  $\mathcal{G}$  based on a text-graph representation strategy. Then, transductive GNN-based learning is used to learn the node embeddings of transductive text graphs using a GNN architecture. The text-graph representation strategy and GNN architecture will be sampled from the TGRl search space.

### 3.3.1 Transductive text-graph construction

In transductive text graph construction, a single graph  $\mathcal{G} = (V; E)$  is built for the entire corpus called transductive text graph, where  $V$  represents document and token nodes and  $E$  represents edges. In the transductive text graph, documents and tokens are represented as nodes, while language properties such as token co-occurrence, semantic and syntactic dependencies are used to build the edges among nodes. There are different text-graph representation strategies such as sequential-based graphs, semantic-based graphs, and syntactic-based graphs [10].

The term frequency-inverse document frequency (TF-IDF) method is used in the process of calculating edge weights between token and document nodes in the transductive text graph. While the edges between token nodes are built using a text-graph representation strategy such as sequential-based graph representation, semantic-based graph representation, and syntactic-based graph representation, the search algorithm will draw the appropriate graph representation strategy from among these three options. Specifically, the edges among nodes are calculated as follows:

$$A_{uv} = \begin{cases} \text{PMI}(u, v); \text{Semantic}(u, v); \\ \quad \text{Syntactic}(u, v) & u, v \text{ are tokens} \\ \text{TF-IDF}_{uv} & u \text{ is document, } v \text{ is token} \\ 1 & u = v \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Where  $\text{PMI}(u, v)$ ,  $\text{Semantic}(u, v)$ , and  $\text{Syntactic}(u, v)$  denotes the text sequential-based graph representation, semantic-based graph representation, and syntactic-based graph representation, respectively, which are calculated as follows:

*Sequential-based graph representation* To build the transductive text graph based on this representation, we utilize point-wise mutual information (PMI) to compute the weights between token node pairs. We only build the edges between token nodes with positive PMI values. A fixed-size sliding window is used to utilize token co-occurrence information. Formally, the following is a definition of the weight of the edge that connects node  $v$  and node  $u$ :

$$\text{PMI}(u, v) = \log \frac{p(u, v)}{p(u)p(v)} \quad (5)$$

$$p(u, v) = \frac{\#N_{\text{windows}}(u, v)}{\#N_{\text{windows}}} \quad (6)$$

$$p(u) = \frac{\#N_{\text{windows}}(u)}{\#N_{\text{windows}}} \quad (7)$$

where  $\#N_{\text{windows}}(u)$  is the number of sliding windows with token  $u$  over the whole corpus,  $\#N_{\text{windows}}(u, v)$  is the

number of sliding windows with token both token  $u$  and  $v$ , and  $\#N_{\text{windows}}$  is the total number of sliding windows.

*Semantic-based graph representation* to build the transductive text graph based on this representation, we utilize the LSTM method [33] to construct the edge weights between token node pairs. We extract the token embeddings using LSTM trained on the training set of the given dataset and compute the cosine similarity between tokens. If the semantic value of a pair of tokens exceeds a predetermined threshold  $\rho$ . We count how many each pair of tokens has a semantic relation over the whole corpus. The edge weight of token pairs can be computed as:

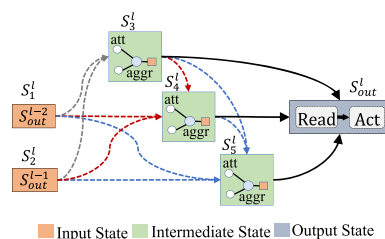
$$\text{Semantic}(u, v) = \frac{\#N_{\text{semantic}}(u, v)}{\#N_{\text{total}}(u, v)} \quad (8)$$

where  $\text{Semantic}(u, v)$  is the edge weight between tokens  $u$  and  $v$ ,  $\#N_{\text{semantic}}(u, v)$  indicates the number of times the tokens  $u$  and  $v$  have semantic relation over all documents in the corpus.  $\#N_{\text{total}}(u, v)$  denotes the number of times the token  $u$  and  $v$  occur across the whole corpus in the same document.

*Syntactic-based graph representation* to build the transductive text graph based on this representation, we use the Stanford CoreNLP parser for a syntactic measure that extracts the dependency between tokens and treats it as an undirected relationship. Over the whole corpus, we count how many times each pair of tokens has a syntactic dependency. The edge of token pairs is calculated by:

$$\text{Syntactic}(u, v) = \frac{\#N_{\text{syntactic}}(u, v)}{\#N_{\text{total}}(u, v)} \quad (9)$$

where  $\text{Syntactic}(u, v)$  is the edge weight between tokens  $u$  and  $v$ ,  $\#N_{\text{syntactic}}(u, v)$  indicates the number of times the tokens  $u$  and  $v$  have syntactic dependency relation over all documents in the corpus.  $\#N_{\text{total}}(u, v)$  denotes the number of times the token  $u$  and  $v$  occur in the same document over corpus.



**Fig. 2** A directed acyclic graph representation of the GNN layer. The dotted arrows indicate that a single preceding state is sampled for each intermediate state

### 3.3.2 Transductive GNN-based Learning

For the transductive GNN-based learning, we design the GNN layer as a directed acyclic graph [34] as illustrated in Fig. 2, where a node  $S_i^l$  corresponds to a state, and a directed dotted edge  $(i, j)$  connects the candidate previous states for each intermediate state. Each layer consists of two input states  $S_1^l$  and  $S_2^l$ , intermediate states  $S_3^l$  to  $S_5^l$  (can be two or more), and output state  $S_{out}^l$ . The output states from the two levels before the current layer are used to initialize the input states, (e.g., the first state  $S_1^l$  in the  $l$ -th layer is equal to the output of the  $(l-2)$ -th layer, and the second state  $S_2^l$  in the  $l$ -th layer is equal to the output of the  $(l-1)$ -th layer). The sampled prior state is used as the starting point for the calculation of the intermediate states, which are then calculated using a message computation operator, a sampling operator, and an aggregate operator. The output state  $S_{out}^l$  is processed by a Read operator from all intermediate states  $S_3^l$  to  $S_5^l$ , then the activation function is applied. The architecture processes the information according to the partial order flow. In particular, the order allows updating the embedding state based on the selected state from all its predecessors sequentially. The input states are configured to mirror the states that were produced by the two layers that came before it:

$$S_i^l = \text{Smpl}(S_j^l | j \leq i) \tag{10}$$

The output state is calculated as the following:

$$S_{out}^{l+1} = \sigma(\text{Read}(F^l(S_i^l | S_i^l \in S_m))) \tag{11}$$

where Smpl,  $\sigma$ , Read, and  $F^l$  are the previous state sampling operator, the activation function, the read operator, and the GNN structure with hyperparameters, respectively.  $S_m$  denotes the intermediate states.  $S_0 = X$  is the representation of the initial node obtained as follows:

$$X = \begin{pmatrix} X_{\text{doc}} \\ 0 \end{pmatrix}_{(n_{\text{doc}} + n_{\text{token}}) \times d} \tag{12}$$

Where  $X_{\text{doc}}$  is the document embeddings produced by the large-scale text embedding module,  $n_{\text{doc}}$  is the total number of nodes in the document,  $n_{\text{token}}$  is the total number of token nodes, and  $d$  is the embedding dimensionality.

In the designed GNN layer, the intermediate state is computed by a GNN function  $F^l$  drawn from a TGRL search space, which contains the attention operation  $g^l$ , aggregation operation Agg, and update operation  $U^l$ , formulated as follows:

$$m_v^{l+1} = \text{Agg}_{u \in N(v)} g^l(h_v^l, h_u^l, A_{vu}) \tag{13}$$

$$h_v^{l+1} = U^l(h_v^l, m_v^{l+1}) \tag{14}$$

The attention operation  $g^l$  receives node representations  $h_v^l$ , neighbor nodes representations  $h_u^l$  for  $u \in N(v)$ , and edge weights  $A_{vu}$  between node  $v$  and  $u$ . The embedding vectors from  $g^l$  are gathered using aggregation operation Agg to obtain hidden features  $m_v^{l+1}$ . The update operation  $U^l$  integrates the hidden features  $m_v^{l+1}$  and the current node features  $h_v^l$  to generate new hidden features  $h_v^{l+1}$ .

### 3.4 Inductive graph-based learning

In the Inductive graph-based learning module, we first construct the inductive text graphs based on the text graph representation strategy sampled from the TGRL search space. Then, inductive GNN-based learning is used to learn the node embeddings of inductive text graphs based on the GNN structure and hyperparameters sampled from the TGRL search space.

#### 3.4.1 Inductive text-graph construction

The inductive graph-based learning module builds the inductive text graphs based on a text-graph representation strategy. In inductive text graph construction, graphs are built for each given text called inductive text graphs. Given a text of  $l$  tokens  $T = \{t_1, \dots, t_i, \dots, t_l\}$ , where  $t_i$  is the embedding of  $i$ -th token. We construct a graph for each text  $\mathcal{G}_d = (T_d; E_d)$ , where  $T_d$  represents token nodes in the text and  $E_d$  represents edges between nodes. Each edge begins with a token in the text and finishes with tokens adjacent to it. The graph of text  $T$  is formulated as follows:

$$N = \{t_i \mid i \in [1, l]\}, \tag{15}$$

$$E = \{e_{ij} \mid i \in [1, l]; j[i - r, i + r]\} \tag{16}$$

where  $N$  denotes the node set,  $E$  denotes the edge set.  $r$  is the number of connected tokens adjacent to each token in the graph. The edge weights in  $E$  are calculated using text-graph representation strategies such as the sequential-based graph, syntactic-based graph, and semantic-based graph representations as described in Sect. 3.3.1.

#### 3.4.2 Inductive GNN-based learning

In inductive GNN-based learning, the initial embedding of token nodes is calculated as follows:

$$w_i = V_i \times f(X_{\text{doc}}) \tag{17}$$

where  $X_{\text{doc}}$  is the text embedding produced by the large-scale embedding module,  $f$  is a linear layer, and  $v_i$  is the pre-trained token embedding (e.g., GloVe [35]). Then, we utilize the DAGNN layer designed in Sect. 3.3.2 to learn the embedding of token nodes in the inductive text graphs.

The token node embeddings are combined to form a graph-level representation of the text after they have been sufficiently updated. Then, the soft attention weight is calculated as follows:

$$h_v = \sigma(g_1(h'_v)) \odot \tanh(g_2(h'_v)) \quad (18)$$

where  $g_1$  and  $g_2$  are two multilayer perceptrons (MLP). The readout layer to compute the graph-level embedding is defined as:

$$H_{\text{IGNN}} = \frac{1}{|V|} \sum_{v \in V} h_v + \text{Maxpooling}(h_1, \dots, h_V) \quad (19)$$

where  $h_v$  is the token embedding,  $V$  is the number of tokens, and max pooling is a max-pooling function.

### 3.5 Automated TGRL model search

To achieve the automatic modeling of text-graph representation learning, we propose the automated TGRL model search module for the AutoTGRL framework. In this module, we designed an effective and customized TGRL search space based on the designed unified TGRL model architecture. Then, we use a search algorithm based on reinforcement learning to search for the optimal TGRL model.

#### 3.5.1 TGRL search space

To determine the TGRL search space, we considered the trade-offs between the complexity of the search space and the performance of the TGRL model. Our aim is to strike a balance between the two by keeping the search space small enough to reduce computation time while being comprehensive enough to explore a broad range of potential TGRL models. To construct an effective TGRL search space, we determined the search space based on a thorough analysis of the text-graph representation learning (TGRL) model structures in the existing literature. Then, we identified all possible combinations of components and their candidate values that could be used to optimize the TGRL model. We also consider the computational resources for conducting experiments. Therefore, we narrowed down the search space by reducing the candidate values of each component that were unlikely to significantly affect the performance of the model or were highly correlated with other candidate values.

On the basis of the TGRL model structure, we developed an effective and customized search space. The TGRL search space consists of three sub-space, including GNN structure and hyperparameters space, large-scale text embedding strategy space, and text-graph representation strategy space. Each sub-space consists of a set of

components with several candidate operations, as shown in Table 2.

*GNN structure and hyperparameters space* We build the GNN structure and hyperparameters space based on the designed DAGNN layer. It consists of five GNN structure components and two hyperparameters. Each DAGNN layer is associated with the following structure components:

1. *Previous state sampling operator* (PSS) It samples the previous state  $S'_j \in \{S'_i | j < i\}$  of the intermediate state  $S'_i$  in the designed GNN layer (Fig. 2). For example, for  $S'_3$ , the previous state is sampled from ( $S'_1$  and  $S'_2$ ), the previous state of the second intermediate state  $S'_4$  is sampled from input states and the first intermediate state ( $S'_1, S'_2, S'_3$ ), and so on. The previous state sampling operator is an essential component of text-graph representation learning because it allows the model to capture dependencies between nodes in the graph. The search space design for this operator must consider all previous states of the current state.
2. *GNN function type* (GNNF) This operator stands for four components: neighbor sampling operator, which chooses the receptive field  $N(v)$  for a given node  $v$ , message computation operator, which calculates the feature representation for each node  $u$  in the receptive field  $N(v)$ , message aggregation operator, required to aggregate the neighborhood structure to form a representation  $h_i$  for each node  $v_i$  [36], and attention head, which helps to perform the stability of the learning process, and final representation  $h_i$  of node  $v_i$  is generated by concatenating the multiple representation outputs from multi-head attentions [37]. Different types of GNN functions have different strengths and weaknesses in capturing the structural information of graphs for different downstream tasks. Therefore, including multiple types of GNN functions in the search space can help to find the best model that can effectively capture the graph structure. As shown in Table 2, we use neighbor sampling operator, message computation, and message aggregation introduced in the original works, including GCN [6], SGC [15], Chebnet [38], ARMA [39], Mean Sage (*GraphSage*) [40], and GAT [7] with multi-attention heads. The initial E in EGCN, EGAT, and ESGC means integrating the edge features in the message passing.
3. *Hidden dimension* (HD) This component determines the size of the intermediate representations in the TGRL model. A larger hidden dimension can increase the capacity of the model to capture more complex relationships between nodes, but it also increases the risk of overfitting. Therefore, including a range of hidden dimensions in the search space can help to find



**Table 2** TGRL search space components

Sub-spaces	Component	Candidate values
GNN structure and hyperparameters space	PSS	Previous states $S_j^l \in \{S_j^l   j < i\}$
	GNNF	GCN, SGC, Chebnet, ARMA, GAT_1, Graph_Conv, Transformer_Conv
	HD	16, 32, 64, 128
	RO	Concatenation, summation, multiplication
	Act	Tanh, Sigmoid, Relu, Leaky_Relu
	DR	0.4, 0.5, 0.6, 0.7
	LR	1e-2, 1e-3, 1e-4, 5e-3, 5e-4
Large-scale embedding strategy space	BSM	Bert-base-uncased, roberta-base, bert-large-uncased, roberta-large
	BLR	1e-5, 5e-3, 1e-3, 1e-4, 5e-3
Text-graph representation strategy space	TGR	SeqGR, SemGR, SynGR

a balance between model capacity and generalization performance. The design of the search space for this hyperparameter should consider the complexity of the task and the available computational resources.

4. *Readout operator* (RO) This component determines how the final representation is computed from the hidden intermediate states in the designed DAGNN layer. Different readout operators have different strengths and weaknesses in capturing global information from graphs. Therefore, including multiple types of readout operators in the search space can help to find the best operator that can effectively capture global information from graphs. The readout operator is applied to process all the intermediate hidden states in the designed GNN layer to represent the output state  $S_{out}^l$ . The *Readout* operator can perform operations such as concatenation, addition, and multiplication.
5. *Activation function* (Act) After forming node representations of the output state,  $S_{out}^l$ , a nonlinear operation (e.g., *Tanh*) is applied to smooth and transform  $S_{out}^l$  as a vector of probabilities for classifying nodes. This component determines the nonlinear function applied to the output of the convolution function. Different activation functions, such as ReLU, sigmoid, and tanh, have different properties that affect how well they perform on different tasks. Therefore, including multiple types of activation functions in the search space can help to find the best function that can effectively introduce nonlinearity into the computation of the model.

The optimization of GNN-based TGRL models is also based on learning parameters. For example, a slight change in learning parameters affects the learning capability of GNN-based TGRL models. Thus, we also aim to optimize the following learning parameters.

1. *Dropout rate* (DR) The dropout technique addresses the overfitting issue during the training of the neural network model. Dropout reduces the complexity of large network training by removing units and their connections during neural network training [41]. A higher dropout rate can increase regularization but may also decrease model performance if too many nodes are dropped out. Therefore, including a range of dropout rates in the search space can help to find a balance between regularization and performance.
2. *Learning rate* (LR) When training a neural network with the gradient-descent algorithm, the learning rate determines the rate of change in the loss at each iteration. A higher learning rate may lead to faster convergence but may also cause instability or overshooting during training if it is too high. Therefore, including a range of learning rates in the search space can help to find an optimal learning rate that balances convergence speed and stability during training.

*Large-scale embedding strategy space* The objective of large-scale text embedding is to generate the embeddings of each document. The document embeddings will be used as initial features of document nodes in transductive and inductive graph-based learning. To enable AutoTGRL to choose the optimal large-scale pretraining model in order to achieve effective document embeddings, a large-scale embedding strategy subspace is designed. We add two components to the large-scale embedding strategy subspace: Bert style model (BSM) and Bert learning rate (BLR) with several candidate options as illustrated in Table 2.

1. *Bert style model* Different types of Bert models, such as Bert-base-uncased, Roberta-base, Bert-large-uncased, and Roberta-large, have different strengths and weaknesses in capturing the sequential information of

text for different downstream tasks. Therefore, including multiple types of Bert models in the search space can help to find the best model that can effectively capture the textual information. The search algorithm will search the TGRL search space for the ideal Bert-style pretraining model.

2. *Bert learning rate* This component determines the learning rate used specifically for the BERT-style layers in the model. A higher learning rate may lead to faster convergence but may also cause instability or overshooting during training if it is too high. Therefore, including a range of learning rates in the search space can help to find an optimal learning rate that balances convergence speed and stability during training.

*Text-graph representation strategy space* Text graph representation strategy plays an important role in constructing transductive and inductive text graphs. AutoTGRL constructs the transductive text graph by representing documents and tokens as nodes in a single graph, but it constructs an inductive text graph for each document by representing tokens in a document as nodes. AutoTGRL generates the edges between token and document nodes using various text-graph representation strategies as described in Sect. 3.3.1. To enable AutoTGRL to choose the optimal text-graph representation strategies for a given dataset, we design a subspace called text-graph representation strategy space. The text-graph representation strategy space consists of a text-graph representation (TGR) component. The choices of graph-based representation strategy are sequential-based graph representation (SeqGR), semantic-based graph representation (SemGR), and syntactic-based graph representation (SynGR).

### 3.5.2 Search algorithm

In the search algorithm, the controller (RNN network) with parameters  $\theta$  predicts a TGRL model  $m$ . The TGRL model contains a set of operators  $(m_{1:T})$  including text-graph representation strategy, large-scale embedding strategy, GNN structure components, and hyperparameters, where  $T$  is the length of the operator set. The operator  $m_j(1 \leq j \leq T)$  is drawn from the designed TGRL search space  $\mathcal{M}$ .

Parameters  $\theta$  are updated using a policy gradient algorithm to maximize the objective function in Eq.(2). Thus, better models are generated over time. After generating the list of operators  $m_{1:T}$  of the TGRL model, the model is built and trained on a given text dataset to generate the text representation which will go through prediction layers based on the downstream tasks to produce the task performance. The task performance on a given validation set  $V$  is returned as a reward  $R_V(m)$  to train the RNN network. We use reinforcement learning to update  $\theta$  as follows:

$$\begin{aligned} &\nabla_{\theta} \mathbb{E}_{\text{pre}(m_{1:T};\theta)} [R] \\ &= \sum_{t=1}^T \mathbb{E}_{\text{pre}(m_{1:T};\theta)} [\nabla_{\theta} \log \text{Pre}(m_t | m_{t-1:1}; \theta) (R - b)], \end{aligned} \tag{20}$$

where  $b$  is the preceding awards' exponential moving average. When training  $m$  model, we utilize the cross-entropy loss function. After AutoTGRL generates 500 TGRL models, we choose the top  $K = 5$  models with the highest evaluation performance and train them 20 times to choose the optimal models. Figure 3 shows an illustrative example of AutoTGRL constructing a DAGNN Layer with two intermediate states.

## 3.6 Model evaluation

We evaluate AutoTGRL on two text-graph representation learning tasks: text classification and aspect-based sentiment analysis, and entity and relation extraction tasks.

### 3.6.1 Text classification

To evaluate AutoTGRL on text classification tasks, we conduct transductive and inductive graph learning to classify texts. This module receives the document node representation  $H_{\text{TGNN}}$  from the transductive graph-based learning module or graph level representation  $H_{\text{IGNN}}$  from the inductive graph-based learning. Then, we apply *softmax* classifier as follows:

$$Z = \text{softmax}(H_{\text{TGNN}}) \tag{21}$$

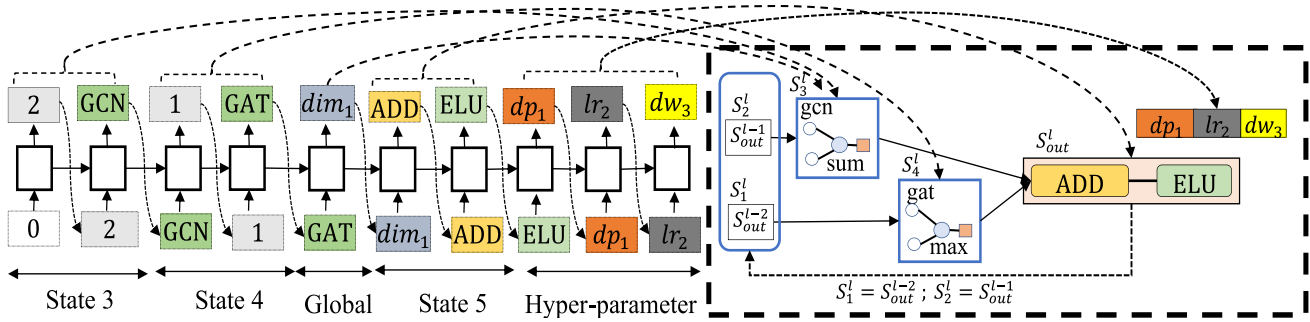
The cross-entropy error across all labeled documents is the loss function for classification tasks as follows:

$$\mathcal{L}_c = - \sum_{d \in \mathcal{Y}_D} \sum_{f=1}^F Y_{df} \log Z_{df} \tag{22}$$

where  $\mathcal{Y}_D$  is the list of document indices with labels and  $F$  is the output feature dimension, which is equivalent to the number of categories. The label indication matrix is represented by  $Y$ .

### 3.6.2 Aspect-based sentiment analysis

The goal of aspect-based sentiment analysis (ABSA) is to mine fine-grained opinion information for specific aspects [42]. ABSA has various subtasks, in which the aspect target is the primary item. Aspect sentiment classification (ASC) [17, 19] is one of the popular ABSA tasks, which estimates the sentiment polarity of a given aspect target. However, assuming the aspect target is given is not always feasible. Aspect sentiment triplet extraction (ASTE) [43, 44] is also



**Fig. 3** An illustrative example of AutoTGRl constructing a single DAGNN Layer with two intermediate states. Left: the outputs of the search algorithm that describe a DAGNN layer architecture. Right: a single DAGNN layer generated by AutoTGRl

another ABSA task, which extracts a triplet comprising an aspect target term, the related opinion term, and the expressed sentiment to provide a more comprehensive sentiment picture. To evaluate AutoTGRl on ABSA tasks, we use inductive graph learning.

For the ASC task, a sentence-aspect pair  $(s, a)$  is given, where  $a = \{a_1, a_2, \dots, a_m\}$  indicates the aspect. The aspect is a sub-sequence of the entire sentence  $T = \{t_1, t_2, \dots, t_n\}$ . After getting the token-level representation, the *Readout* layer is applied to aggregate the representations of all the aspect nodes and then concatenate the obtained representations to form the final aspect representation  $H_{IGNN}$  Eq. (19). Then, the generated representation  $H_{IGNN}$  is input into a linear layer. Finally, we apply a softmax function to generate a sentiment probability distribution  $pro$  as shown below:

$$pro(a) = \text{softmax}(W_{pro} \cdot H_{IGNN} + b_{pro}) \tag{23}$$

Where  $b_{pro}$  is a learnable bias and  $W_{pro}$  is the learnable weight. Therefore, the loss function of the ASC task can be formulated by using cross-entropy loss as follows:

$$\mathcal{L}_{pro} = - \sum_{(s,a) \in \mathcal{Z}} \sum_{p \in \mathcal{P}} \log pro(a) \tag{24}$$

where  $\mathcal{Z}$  indicates all sentence-aspect pairs and  $\mathcal{P}$  is the distinct sentiment polarities.

For the ASTE task, a sentence  $T = \{t_1, t_2, \dots, t_n\}$  is given, where  $n$  is the number of tokens. ASTE produces triplets consisting of an aspect target, the sentiment that is associated with it, and the opinion term that corresponds to it. Each triplet is formed as (aspect, opinion, sentiment), where  $sentiment \in \{\text{positive, neural, negative, invalid}\}$ . where *invalid* indicates that the aspect and opinion have no valid sentiment relationship.

To evaluate AutoTGRl on the ASTE task, we use inductive graph learning after removing the *Readout* layer and keeping the token-level representation. After getting the token-level representation  $H_{IGNN}$ , we apply the token classifier. We assume  $\tau$  to be a predefined set of token

categories, including aspect, opinion, and Null  $\tau \in \{\text{aspect, opinion, null}\}$ . Where null indicates that tokens do not belong to opinion or aspect terms. Therefore, the token representation is fed as input to a feed-forward neural network and nonlinear activation such as softmax as follows:

$$p_t(\tau | h_v) = \text{softmax}(\text{FFNN}_\tau(h_v)) \tag{25}$$

where FFNN indicates a feed-forward neural network, and softmax indicates a nonlinear activation. Then, another feed-forward neural network is used to determine the probability of a sentiment relationship  $r \in \{\text{positive, neural, negative, invalid}\}$  as follows:

$$p_s(r | h_a, h_o) = \text{softmax}(\text{FFNN}_r(h_a, h_o)) \tag{26}$$

where  $h_a, h_o$  are the aspect and opinion token representations, respectively, determined by Eq. (25). FFNN indicates a feed-forward neural network. The training objective is specified as the sum of the negative log-likelihood from both the token type prediction and sentiment prediction.

$$L = - \sum_{h_v \in H} \log p_t(\tau_v^* | h_v) - \sum_{h_a \in H^a, h_o \in H^o} \log p_s(r^* | h_a, h_o) \tag{27}$$

where  $\tau_v^*$  is the gold token type and  $r^*$  is the gold sentiment relationship of aspect and opinion pair  $(h_a, h_o)$ ;  $H^a$  and  $H^o$  are the aspect and opinion token candidates.

### 3.6.3 Entity and relation extraction

Identifying entities and their relationships constitutes a fundamental undertaking within the realm of text mining, particularly in light of the exponential expansion of texts such as biomedical literature and clinical documentation [45, 46].

The objective of the entity and relation task is to extract relational triplets from the text. The input text is represented as a set of tokens,  $X = \{x_1, x_2, \dots, x_n\}$ , while  $S = \{s_{1,1}, s_{1,2}, \dots, s_{i,j}, \dots, s_{n,n}\}$  is the set of all possible spans in

$X$ , where  $i$  and  $j$  represent the starting and ending positions of a span in  $X$ . The span-based graph,  $G = (S;A)$ , is constructed from the input text. The predefined set of entity types and relationships are represented by  $T$  and  $R$ , respectively. A relational triplet is formulated as  $(e_i, r_{ij}, e_j)$ , where  $e_i \in T$ ,  $e_j \in T$ , and  $r_{ij} \in R$ , indicating that the relationship  $r_{ij}$  exists between the entity span type  $e_i$  and entity span type  $e_j$ . We apply AutoTGRL to extract entities and the relations between them from biomedical text. We use inductive graph learning to evaluate the proposed AutoTGRL on the entity and relation extraction task. Since biomedical entities may contain more than one word, text-graph representation methods introduced in the inductive text-graph representation section are not suitable candidate options for the text-graph representation component in the text-graph representation strategy space. Therefore, a new text-graph representation method called span-based graph representation (SpanGR) is introduced.

*Span-based graph representation* The primary objective of the span graph is to encompass all potential connections among spans, thereby promoting message propagation between span representations that exhibit semantic dissimilarities. First, we use SciBERT [47] to create contextualized token representations  $H = \{h_1, h_2, \dots, h_n\}$  for a text with  $n$  tokens. We then calculate span representations  $s_i$  for each possible span  $s_i$  as follows:

$$s_i = \text{FFN}_g([\hat{h}_{s_i}, h_{\text{start}(i)}, h_{\text{end}(i)}, \phi(s_i)]) \tag{28}$$

where  $s_i \in \mathbb{R}^d$ , using a feed-forward network with an activation function  $\text{FFN}_g$ , boundary token representations  $h_{\text{start}(i)}$  and  $h_{\text{end}(i)}$ , an embedding vector  $\phi(s_i)$  denoting span width, and a weighted-attention mechanism  $\hat{h}_{s_i}$  over the token representations in span  $s_i$ . The attention mechanism is computed as:

$$\hat{h}_{s_i} = \sum_{t=\text{start}}^{\text{end}} \text{softmax}(V \cdot \tanh(W_{\text{att}} \cdot h_t)) \cdot h_t \tag{29}$$

where  $V$  and  $W_{\text{att}}$  are attention parameters.

Afterward, span representations are used to predict both the entity type and relation type between each pair of spans. The set of predefined entity types (including non-entity) is denoted by  $\mathcal{T}$ , and the set of predefined relation types (including non-relation) is denoted by  $\mathcal{R}$ . To estimate the entity type for each span  $s_i$ , a feedforward network  $\text{FFN}_e$  is employed, which maps from  $\mathbb{R}^d$  to  $\mathbb{R}^{\mathcal{T}}$ . The probability of  $s_i$  belonging to each entity type in  $\mathcal{T}$  is computed using the softmax function applied to  $\text{FFN}_e(s_i)$ , as follows:

$$\text{Pr}_i^e = \text{Softmax}(\text{FFN}_e(s_i)) \tag{30}$$

To predict the relation type between each span pairs  $\langle s_i, s_j \rangle$ ,

another feedforward network  $\text{FFN}_r$  is used. This network maps from  $\mathbb{R}^{3 \times d}$  to  $\mathbb{R}^{\mathcal{R}}$ , where  $\mathcal{R}$  is the cardinality of  $\mathcal{R}$ . The input to  $\text{FFN}_r$  is a concatenation of  $s_i, s_j$ , and the element-wise multiplication of  $s_i$  and  $s_j$ . The probability of each relation type in  $\mathcal{R}$  between  $s_i$  and  $s_j$  is computed using the softmax function applied to  $\text{FFN}_r([s_i, s_j, s_i \circ s_j])$  as follows:

$$\text{Score}_{ij}^r = \text{Softmax}(\text{FFN}_r([s_i, s_j, s_i \circ s_j])) \tag{31}$$

Here,  $\text{Score}_{ij}^r$  represents the probability of the span pair  $\langle s_i, s_j \rangle$  having the relation type  $r$ .

*Objective learning* To improve computational efficiency in entity extraction, we employ two filtering strategies. Firstly, we can filter out spans that have a low probability of being entities by setting a probability threshold. This helps to reduce the number of candidate spans and we keep only the top  $n\lambda$  spans with the highest probability of being entities. Secondly, we can restrict the maximum width of the spans to a fixed number  $L$ . This means that we only consider spans that are less than or equal to  $L$  in length. These filtering strategies are necessary because the number of candidate spans in a sentence of  $n$  tokens is given by  $N = \frac{n(n+1)}{2}$ , and this number can significantly impact computational efficiency.

The total loss for training AutoTGRL on an entity and relation extraction task is calculated by adding two individual loss terms,  $\mathcal{L}^e$  and  $\mathcal{L}^r$ , as shown below:

$$\mathcal{L}_{\text{total}} = \mathcal{L}^e + \mathcal{L}^r \tag{32}$$

Here,  $\mathcal{L}^e$  represents the cross-entropy loss for entity classification of the span graph, and  $\mathcal{L}^r$  represents the binary cross-entropy loss for relation classification. These terms correspond to the loss functions described in Eqs. (30) and (31).

### 3.7 Balancing module errors

For a given dataset, we use four modules of the AutoTGRL framework to achieve a specific task. The modules are large-scale text embedding, graph-based learning module (whether inductive-based or transductive-based), automated TGRL model search module, and model evaluation module. There are some errors that may appear between modules. For example, large-scale text embedding may struggle to capture relationships between text entities, whereas graph-based models may struggle to capture the semantic and contextual information of the text. An automated TGRL model search module may generate TGRL model architectures that have lower performance than the current model during the search process. Therefore, developing an effective TGRL model requires careful

balancing of errors between the automated TGRL model search module and the model evaluation module. In this section, we introduce two methods to balance errors between AutoTGRL modules.

### 3.7.1 Fine-tuning embedding method

The AutoTGRL uses large-scale text embedding to generate document and word embeddings using pre-trained models such as BERT and RoBERTa that capture the semantic and contextual information of the text. These embeddings are then used as initial embeddings for the graph-based learning module. In the graph-based learning module, a graph is constructed based on different text graph representation methods. The graph structure is used to capture the relationships between words, and graph neural networks (GNNs) are used to learn the node representations of the graph. GNNs are effective in capturing complex relationships and dependencies between text data. However, large-scale text embedding and graph-based learning can produce different errors due to their inherent characteristics. Large-scale text embedding may struggle to capture relationships between text entities, whereas graph-based models may struggle to capture the semantic and contextual information of the text.

Balancing errors between large-scale text embedding and graph-based learning modules is crucial for building accurate TGRL models. To balance errors between the large-scale text embedding and graph-based learning modules, in the large-scale text embedding module, we fine-tune document embeddings using large-scale models based on truth labels involves training large-scale text embedding models on a given labeled dataset [48]. Then, the graph-based learning module is trained on the same labeled dataset using the initialized embeddings to capture the relationship between word and document nodes while preserving their semantic and contextual information. Fine-tuning document embeddings using large-scale models based on truth labels is an effective approach to balance errors between AutoTGRL modules and improve the overall performance of TGRL models for various NLP tasks such as text classification, aspect-based sentiment analysis, entity extraction, and relation extraction.

### 3.7.2 Performance-based feedback method

The automated TGRL model search module is responsible for exploring a vast TGRL search space of possible model architectures to find the best TGRL model for a particular task. The model evaluation module is responsible for assessing the performance of the generated TGRL models

on a given task. This module uses various evaluation metrics, such as accuracy and F1-score, to assess the performance of the candidate TGRL models. However, developing an effective TGRL model requires careful balancing of errors between the automated TGRL model search module and the model evaluation module. To balance the errors in the automated TGRL model search module, it is crucial to strike a balance between exploration and exploitation. This means that the controller should explore the space of possible models to find new architectures that may perform better than the current ones. At the same time, it should exploit the knowledge gained from previous evaluations to guide the search toward more promising architectures.

To address this issue, a performance-based feedback loop is introduced between the automated TGRL model search module and the model evaluation module [28]. By incorporating feedback from the model evaluation module, the automated TGRL model search module can guide the controller of the search algorithm to sample better models. This module uses a reinforcement learning-based search algorithm to guide the search process. The RL-based controller receives feedback from the model evaluation module in the form of a reward signal. The reward signal can be positive, negative, or zero, depending on the performance of the generated model. The controller then uses this reward signal to update its policy and select better models in the next iteration. The policy gradient algorithm is used to maximize the objective function of generating better TGRL models. The algorithm adjusts the parameters of the search algorithm based on the feedback from the model evaluation module. This results in the search algorithm sampling a new set of TGRL models that are more likely to meet the desired performance metrics. The model evaluation module continues to provide feedback, which is used to refine the search algorithm and the policy gradient algorithm. The iterative process continues until the search module generates the optimal TGRL model. During training, the RL algorithm aims to maximize the expected reward signal by learning an optimal policy. This policy guides the controller to sample better models that perform well on the given task.

## 4 Experiments

Extensive tests and evaluations of the performance of AutoTGRL are carried out in this section so that we may better understand its capabilities.

## 4.1 Implementation details

We configure the experiment settings as follows:

*Hyperparameters of the search algorithm controller* The controller is a single-layer LSTM with 100 hidden units. Model training is accomplished with a learning rate of  $3.5e - 4$  using the ADAM optimizer. The default range for the controller weights is  $- 0.1$  to  $0.1$ . To avoid premature convergence, we use a tanh of 2.5 and a temperature of 5.0 for the sampling logits, and we add the entropy of the controller sample to the reward, weighted by  $1e - 4$ .

*Hyperparameters of the text classification and aspect-based sentiment analysis tasks* Following the construction of the TGRL model by the RNN network (controller), the model is designed and trained for 300 epochs with a batch size of 64. Two intermediate states are designated in the proposed GNN layer. The search space is sampled for the learning rates, dropout rates, and hidden units. The value of the weight decay is  $5e-05$ . For baseline models, we got the results from their original papers.

*Hyperparameters of the entity and relation extraction task* To assess the sampled TGRL model on the validation set  $V$ , we reserve 10% of the ADE training set. The sampled model is built and trained for 50 epochs. The optimal learning rate identified by the search algorithm for the ADE dataset is  $1e-04$ , while the optimal dropout rate is 0.2. AutoTGRL utilizes SciBERT as the transformer encoder and the maximum span width is limited to 15 tokens. For the span filtering strategy, the pruning parameter  $\lambda$  is set to 0.6. The model is trained for 50 epochs with a batch size of 32, while the learning rate for the pre-trained transformer encoder is set to  $5e-05$  for both datasets. The hidden dimension of the GNN layer is set to 500. The maximum width for a span is set at 20 tokens. We added a new candidate value to the Bert style model component, the biomedical Bert model called SciBERT [47], and a candidate value to the text graph representation component called span-based graph representation (SpanGR).

## 4.2 Baselines

In our experiments, we compare our optimum TGRL models created by AutoTGRL to state-of-the-art networks created by human specialists using manual modeling. We also compare AutoTGRL to Text-NAS approaches to show the success of our proposed method. Four kinds of models are considered: token embedding-based models, text-graph representation models, Text-NAS approaches, and large-scale pre-trained representation models.

### 4.2.1 Manually designed networks

For evaluation on the text classification task, we choose the SOTA manual designed models to compare against AutoTGRL with transductive learning: FastText, presented by [49], a text classification strategy which deals with the n-gram or token average embeddings as document embeddings. It is evaluated with and without bigrams. SWEM, presented by [50], a token embedding method, which applies pooling strategies over token embeddings. LEAM, proposed by [51], a label embedding with attention method, which uses label description for embedding tokens and labels for classifying text in the same joint. SGC, proposed by [15], a simple version of GCN that reduces the unnecessary redundant and complex computation applied to a text-graph representation. GCN-with-label, A graph convolutional network [9], which incorporates the label information along with texts and tokens while building a heterogeneous graph. Text-GCN (with transductive settings), proposed by [11], text-graph-based text classification method, in which a convolutional network is applied on a large graph built from the entire corpus. TensorGCN, a model that integrates more context information such as semantic and syntactic dependencies between tokens, proposed by [8]. HieGAT, a graph neural network based on hierarchical learning (HieGNN) that extracts information from token, sentence, and document levels [52]. GCN-CNN-SB, a model that utilizes GCN to catch the global information and CNN to extract local features [53]. HINT, a strategy with the goal of extracting the maximum amount of information from the hierarchical data that is stored in the text tree-based graph neural network [54]. MP-GCN, a semi-supervised text categorization using an MP-GCN (multi-head-pooling-based GCN) [18]. TEGNN [3] provides a text categorization algorithm that topologically improves upon the structural features of the corpus graph and the sentence graph.

To compare AutoTGRL with inductive text graph-based learning, we selected the following SOTA inductive-based learning models: TIGNN and TextING-M, proposed by [12, 13], a GNN models in which a graph is built for each input text with global parameter sharing. While Graph-CNN-C, Graph-CNN-F, and Graph-CNN-S are used to compare with both inductive and transductive learning settings, which are proposed by [38, 55, 56], which apply convolution over token embedding similarity graphs. GraphCNN-C, GraphCNN-F, and GraphCNN-S utilized Cheby-shev, Fourier, and Spline filters, respectively. InducT-GCN, a graph convolutional network and an inductive model for classifying text based on graphs were proposed by [57]. HyperGAT, a GNN model proposing to model texts using document-level hypergraphs that manage high-order interactions between tokens. [16]. TLGCN, a

text graph constructed using an inductive light graph convolution network [58]. GFN [59], a graph fusion network enables quick inference and captures structural information by combining multiple perspectives of text graphs. A new GNN-based model that constructs graphs for each text input while sharing global parameters, as opposed to a unique graph for the entire corpus.

To evaluate the performance of the AutoTGRL model on the ABSA task with giving the aspects (called ASC), we selected the following SOTA manual designed networks to compare against AutoTGRL: BiGCN, in which hierarchical graphs integrate token co-occurrence and dependence type information [21]. InterGCN applies a GCN on a dependency tree to model syntactically informed representations of aspects [17]. DualGCN is a model that simultaneously analyzes the syntactic complementarity of structures with semantic correlations [19]. R-GAT [60] proposes a structure for aspect-oriented dependency trees and encodes dependency trees using a relational GAT. DGEDT [61] suggests a dual-transformer network with a better dependency graph. kumaGCN [62] leverages a latent graph structure to augment syntactic characteristics, taking both flat and graph-based representations into account at the same time. SSEGCN [1] offers a syntactic and semantically improved graph convolutional network that uses an aspect-aware attention mechanism paired with self-attention to generate sentence attention score matrices. To evaluate the performance of the AutoTGRL model on the ABSA task including aspect and opinion term extraction, and the sentiment between them (called ASTE), we selected the following SOTA models: RINANTE [63] mines aspect and opinion term extraction rules from sentence dependency relations; CMLA [64] attention-based co-extraction model that uses direct and direction dependence relations; Li-unified [2], an aspect extraction and sentiment classification model; Li-unified-R modified Li-unified for opinion extraction. GTS [43], a grid tagging scheme with CNN, BiLSTM, and BERT as encoders, and WHW-ASTE [44] extracting sentiment triplets model.

To evaluate the performance of the AutoTGRL model on the entity and relation extraction tasks, we compared it against several state-of-the-art biomedical entities and relation extraction models, including Relation-Metric [4], Multihead with AT [5], Multihead [65], SpERT [66], KECI [20], SPAN<sub>Multi-Head</sub> [67], Neural joint model [68], TabLERT [45], TabLERT-CNN [46], and EINET[69]. The results of TabLERT are replicated in TabLERT-CNN [46], and the results of the others are taken from the original papers. These models were evaluated against the optimal TGRL models identified by AutoTGRL for biomedical entities and relation extraction tasks.

## 4.2.2 NAS and large-scale pretrained methods

We select the following SOTA NAS and large-scale pretrained methods as the baselines for the text classification task: ENAS, proposed by [25] that apply two search spaces: macro search space ENAS-MACRO and micro search space ENAS-MICRO. When running ENAS-MACRO, the method looks for a neural network that has a total of 24 layers. ENAS-MICRO is a micro-search space that covers normal cells as well as reduction cells. We alternate stacking the regular cells and the reduction cells. DARTS, a differentiable architecture search that is based on the ongoing relaxing of architectural representations [26]. SMASH, a one-shot model architecture search that involves building a hyper-network to generate parameter values to every model [27]. One-shot, proposed by [70], which studies sharing weights for one-shot architecture search to determine optimal architectures without reinforcement learning or hyper-networks. Random Search, a random search with weight sharing and early stopping [71]. TextNas, multi-path ensemble using a blend of convolutional, recurrent, pooling, and self-attention layers as a text representation architecture search space [24]. The large-scale pre-trained methods baselines are as follows: BERT [31], RoBERTa [32], BERTGCN, and RoBERTaGCN [48], models for linguistic representation based on a graph convolutional network and transformers. We utilize the Bert-Based model with 12 layers and 768 attention heads. ELMo [72] and OpenAI GPT [73] are also pre-trained language representation models with around a hundred million parameters.

## 4.3 Evaluation datasets

To evaluate the AutoTGRL model on text classification tasks, we used six benchmark datasets to compare AutoTGRL with manually designed baseline networks, including a movie review dataset (MR), two subsets (R52 and R8) of the Reuters 21578 dataset (with 52 and 8 categories, respectively), and medical abstracts corpus Ohsumed with 23 cardiovascular diseases categories. Text-graph representation classification performance is often assessed using the datasets [8, 11, 12]. Our experiments use the same preprocessing and train/test split methods in [11]. We use STT datasets to evaluate the efficacy of our proposed method to the standard NAS methodologies. Stanford Sentiment Treebank (SST) is a popular sentiment classification dataset. SST has almost 12,000 reviews with one of the five emotion kinds. SST-Binary exclusively classifies positive and negative values, deleting neutral

**Table 3** Summary statistics of the evaluation datasets for the text classification task

Dataset	MR	R52	R8	Ohsumed	SST-B	SST
#Docs	10,662	9100	7674	7400	9613	11,855
#Training	7108	6532	5485	3357	7792	9645
#Test	3554	2568	2189	4043	1821	2210
#Tokens	18,764	8892	7688	14,157	16,162	17,087
#Nodes	29,426	17,992	15,362	21,557	25,775	28,942
#Classes	2	52	8	23	2	5
Avg Len	20.39	69.82	65.72	135.82	18.4	18.3

samples. The original dataset's train/validation/test split has been followed.<sup>1</sup> Table 3 shows the summary statistics of the datasets for the text classification task.

To test AutoTGRL on ABSA tasks, we use three standard public datasets: The SemEval ABSA challenge [74] provided the Laptop and Restaurant datasets. Tweets comprise the Twitter dataset [75]. Positive, negative, and neutral sentiments characterize the three datasets. The databases categorize each statement with aspects and polarity. Table 4 presents the statistics of the ASC task evaluation dataset. For ASTE task evaluation, we use the datasets annotated by Fan et al. [76]. Table 5 summarizes the statistics of the dataset.

In order to evaluate the performance of the AutoTGRL model in biomedical entity and relation extraction, we conducted experiments on the adverse drug event extraction (ADE) dataset [77]. The ADE dataset comprises medical reports describing the negative effects of drug use and contains a total of 4272 sentences, out of which 1695 involve overlapping entities and 6821 relations. The dataset defines two pre-determined entity types, namely, Adverse-Effect and Drug, as well as a single relation type, namely, Adverse-Effect. To ensure the reliability of our results, we performed 10-fold cross-validation and reported the macro-averaged precision (P), recall (R), and F1 scores.

## 4.4 Overall results

### 4.4.1 Manually designed model comparisons

To validate the effectiveness of the AutoTGRL in designing the optimal TGRL models for text classification and ABSA tasks, it is compared with the state-of-the-art text-graph representation learning models, token embedding-based models, and transformer-based language representation models that are used for both tasks. Figure 4 shows an example of the DAGNN layers designed by AutoTGRL for the text classification tasks. Table 6 presents the text classification accuracy of the models designed by AutoTGRL with transductive graph-based learning against

the manually designed architectures over MR, R52, R8, and Ohsumed datasets. While Table 7 presents the text classification accuracy of the models designed by AutoTGRL with inductive graph-based learning against the manually designed architectures over the same datasets. Table 8 reports the accuracy and macro-averaged F1-score of the experimental result on the ASC task using inductive graph-based learning compared with the models. Table 9 reports the recall, precision, and F1-score of the experimental results on the ASTE task using inductive graph-based learning compared with the state-of-the-art models. From the results, we can notice that the models designed by AutoTGRL under both transductive settings achieve competitive performances compared with SOTA models over all datasets for different text classification tasks. These results show superior capability AutoTGRL of constructing the optimal TGRL models in the text classification task. We also see that models designed by AutoTGRL with inductive graph-based learning outperform the existing inductive-based learning models on text classification, ASC, and ASTE tasks. We used boldface to emphasize the greatest performance in each column. The findings demonstrate the superiority of the AutoTGRL on various text datasets. Moreover, according to the findings shown in Table 10, AutoTGRL outperforms the current leading models in terms of precision, recall, and F1 scores for entity and relation extraction tasks on the ADE dataset.

The reason why AutoTGRL works well is twofold. First, the text-graph representation constructed by AutoTGRL can integrate syntactic and semantic information of token and document nodes, which can well guide the message passing between nodes. Second, AutoTGRL has the ability to design the optimal combinations of the TGRL model for the given text graph for different tasks. For more in-depth analysis, we observe that text-graph representation learning methods perform better than token embedding-based methods. The potential reason could be the characteristics of the graph structures. The network structure allows document and token nodes to acquire more precise embeddings by using diverse collocations.

<sup>1</sup> <https://nlp.stanford.edu/sentiment/code.html>.

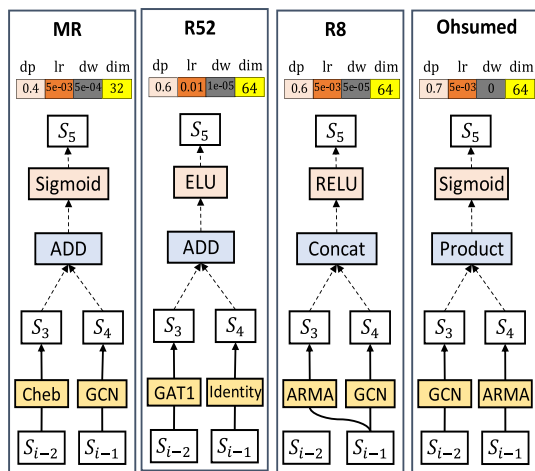


**Table 4** Summary statistics of the evaluation datasets for ASC task

Datasets	Division	# Negatives	# Positives	# Neutrals
Restaurant	Training set	807	2164	637
	Testing set	196	727	196
Laptop	Training set	851	976	455
	Testing set	128	337	167
Twitter	Training set	1528	1507	3016
	Testing set	169	172	336

**Table 5** Summary statistics of the evaluation datasets for ASTE task

Datasets	14res			14lap		
	Train	Dev	Test	Train	Dev	Test
Sentences	1300	323	496	920	228	339
Aspect-opinion pairs	2145	524	862	1265	337	490



**Fig. 4** An example of the DAGNN layer designed by AutoTGRl for text classification tasks

**4.4.2 NAS and large-scale pretrained models**

To demonstrate the efficacy of the proposed method, we also compare AutoTGRl under transductive learning settings with other NAS and large-scale pre-trained models on the text classification task. Table 11 illustrates the evaluation results of TGRl models designed by AutoTGRl against the baseline NAS and transformer models on SST datasets. We can notice that the TGRl models designed by AutoTGRl achieve competitive performance compared to the other network architectures designed automatically by the NAS approach. The experiment results demonstrate the effectiveness of our method and its capability to discover

the optimal TGRl models on different datasets. In addition, we compare AutoTGRl to the large-scale pretraining models. From the results, we can see that AutoTGRl outperforms the best baseline pre-trained models since they are used to initialize the document embedding based on the large-scale embedding strategy suitable for the dataset selected by the search algorithm, which makes our model more powerful. This enhancement is due to the ability of AutoTGRl to construct the best TGRl model for learning text representation.

**4.5 Average run-time and convergence speed**

In this section, the training time required by AutoTGRl models and the best baselines over R52, R8, and Ohsumed datasets are reported. In Tables 12 and 13, we report the training runtime of the designed TGRl models by AutoTGRl with transductive and inductive graph learning, respectively. We also compare TGRl models designed by AutoTGRl against the best baseline models, which achieved the highest performance in the literature. From the results, we observe that the models designed by AutoTGRl are faster than the baseline models. For example, using transductive graph learning, the AutoTGRl model completes a training phase in 125.356s, which is 9.894s faster than the baseline GCN-CNN-SB model and 67.244s faster than the Text-GCN model on the R8 dataset. Using inductive graph learning, the AutoTGRl model completes a training phase in 628.24s, which is 278.76s faster than the GFN model, 210.76s faster than TLGCN model, and 43.54s faster than the TextING-M model on the R8 dataset.

For more convergence analysis, we report the accuracy and loss values of the first 40 training epochs of the TGRl models designed by AutoTGRl to determine the convergence speed and accuracy performance of the proposed model on R8 and R52 datasets. We also compare the convergence speed and accuracy performance of the optimal models designed by AutoTGRl using transductive and inductive learning models to the best transductive and inductive baseline models. The epoch number and classification accuracy score are displayed in Fig. 5. From the results, we can notice that TGRl models designed by AutoTGRl outperform the baseline from the first epoch. For example, the baseline requires 20 epochs to improve accuracy on the R8 dataset, but the AutoTGRl model requires just 5 epochs. Since AutoTGRl converges faster than the baseline, it achieves optimum performance faster. The loss value decreasing trend in Fig. 6 confirms the aforementioned result. The loss curve of the AutoTGRl model decreases with a considerable gradient during the first epochs. In conclusion, the TGRl models designed by

**Table 6** Test accuracy comparison of models identified by AutoTGRL with transductive graph-based learning against the transductive-based learning baselines on text classification task over four benchmark datasets

Model	MR	R52	R8	Ohsumed
FastText	75.14 ± 0.20	92.81 ± 0.09	96.13 ± 0.21	57.70 ± 0.49
FastText (bigrams)	76.24 ± 0.12	90.99 ± 0.05	94.74 ± 0.11	55.69 ± 0.39
SWEM	76.65 ± 0.63	92.94 ± 0.24	95.32 ± 0.26	63.12 ± 0.55
LEAM	76.95 ± 0.45	91.84 ± 0.23	93.31 ± 0.24	58.58 ± 0.79
GraphCNN-C	77.22 ± 0.27	92.75 ± 0.22	96.99 ± 0.12	63.86 ± 0.53
GraphCNN-S	76.99 ± 0.14	92.74 ± 0.24	96.80 ± 0.20	62.82 ± 0.37
GraphCNN-F	76.74 ± 0.21	93.20 ± 0.04	96.89 ± 0.06	63.04 ± 0.77
TextGCN (transd)	76.74 ± 0.2	93.56 ± 0.18	97.07 ± 0.10	68.36 ± 0.56
SGC	75.9 ± 0.3	94.0 ± 0.2	97.2 ± 0.1	68.5 ± 0.30
TensorGCN	77.91 ± 0.07	95.05 ± 0.01	98.04 ± 0.08	70.11 ± 0.24
GCN-with-label	–	94.15 ± 0.10	97.37 ± 0.17	69.10 ± 0.20
HieGAT	78.04	94.54	97.83	69.84
GCN-CNN-SB	87.59	96.35	98.53	71.85
HINT	77.03 ± 0.12	95.02 ± 0.18	98.12 ± 0.09	68.79 ± 0.12
MP-GCN	77.92 ± 0.01	94.54 ± 0.08	97.85 ± 0.02	70.27 ± 0.12
TEGNN	80.83	95.91	98.26	71.33
AutoTGRL	<b>89.91 ± 0.14</b>	<b>97.39 ± 0.10</b>	<b>98.61 ± 0.10</b>	<b>73.25 ± 0.10</b>

**Table 7** Test accuracy comparison of models identified by AutoTGRL with inductive learning against the inductive-based learning baseline models on text classification task over four benchmark datasets

Model	MR	R52	R8	Ohsumed
Graph-CNN-C	77.22 ± 0.27	92.75 ± 0.22	96.99 ± 0.12	63.86 ± 0.53
Graph-CNN-S	76.99 ± 0.14	92.74 ± 0.24	96.80 ± 0.20	62.82 ± 0.37
Graph-CNN-F	76.74 ± 0.21	93.20 ± 0.04	96.89 ± 0.06	63.04 ± 0.77
HyperGAT	78.32 ± 0.27	94.98 ± 0.27	97.97 ± 0.23	69.90 ± 0.34
Text-GCN (induct.)	74.80 ± 0.25	88.20 ± 0.72	95.78 ± 0.29	57.70 ± 0.35
TextING-M	80.19 ± 0.31	95.68 ± 0.35	98.13 ± 0.12	70.84 ± 0.52
InducT-GCN	79.37 ± 0.38	95.23 ± 0.03	97.47 ± 0.51	70.21 ± 0.34
TIGCN	77.40 ± 0.10	94.4 ± 0.10	97.5 ± 0.10	66.30 ± 0.20
TLGNN	–	94.60 ± 0.30	97.80 ± 0.20	69.40 ± 0.60
GFN	78.04	95.31	98.22	70.20
AutoTGRL	<b>81.12 ± 0.10</b>	<b>96.20 ± 0.10</b>	<b>98.32 ± 0.10</b>	<b>71.60 ± 0.14</b>

**Table 8** Performance of ASC task by models identified by AutoTGRL and baseline models

Model	Laptop		Restaurant		Twitter	
	Acc	F1	ACC	F1	Acc	F1
BiGCN	74.59	71.84	81.97	73.48	74.16	73.35
DGEDT	76.80	72.30	83.90	75.10	74.80	73.40
kumaGCN	76.12	72.42	81.43	73.64	72.45	70.77
InterGCN	77.86	74.32	82.23	74.01	–	–
R-GAT	77.42	73.7	83.30	76.08	75.57	73.82
DualGCN+BERT	81.80	78.10	87.13	81.16	77.40	76.02
SSEGCN+BERT	81.01	77.96	87.31	81.09	77.40	76.02
AutoTGRL	<b>82.63</b>	<b>79.45</b>	<b>88.43</b>	<b>82.72</b>	<b>78.54</b>	<b>77.36</b>

the AutoTGRL model have faster convergence and greater performance.

We also presented the model size (number of parameters) of the best transductive and inductive-based learning baselines and compared them to AutoTGRL models with inductive and transductive learning. Table 14 shows that AutoTGRL models for R58 and R52 datasets have fewer parameters than the best baselines for transductive or inductive learning.

## 4.6 Ablation study

### 4.6.1 Ablation study of large-scale text embedding

This section investigates the significance of large-scale text embedding on R8, R52, MR, and Ohsumed datasets. We examine the influence of embeddings generated by this

**Table 9** Performance of ASTE task by models identified by AutoTGRL and baseline models

Model	14res			14lap		
	P.	R.	F1	P.	R.	F1
RINANTE	48.97	47.36	48.15	41.20	33.20	36.70
CMLA	67.80	73.69	70.62	54.70	59.20	56.90
WHW-ASTE	<b>76.60</b>	67.84	71.95	63.15	61.55	62.34
Li-unified	74.43	69.26	71.75	<b>68.01</b>	56.72	61.86
Li-unified-R	73.15	74.44	73.79	66.28	60.71	63.38
GTS-CNN	70.79	61.71	65.94	55.93	47.52	51.38
GTS-BiLSTM	67.28	61.91	64.49	59.42	45.13	51.30
GTS-BERT	70.92	69.49	70.20	57.52	51.92	54.58
IMN+IOG	59.57	63.88	61.65	49.21	46.23	47.68
AutoTGRL	74.90	<b>75.01</b>	<b>74.95</b>	67.64	<b>62.12</b>	<b>64.76</b>

**Table 10** Performance of entity and relation extraction task by AutoTGRL and other state-of-the-art models on the ADE dataset

Model	Entity			Relation		
	P	R	F1	P	R	F1
Neural Joint Model	82.7	86.7	84.60	67.5	75.8	71.40
Relation-Metric	86.16	88.08	87.11	77.36	77.25	77.29
Multihead	84.72	88.16	86.40	72.10	77.24	74.58
Multihead with AT	–	–	86.73	–	–	75.52
SpERT	88.99	89.59	89.28	77.77	79.96	78.84
KECI	–	–	90.67	–	–	81.74
SPAN <sub>Multi-Head</sub>	89.88	91.32	90.59	79.56	81.93	80.73
TabIERT	–	–	89.90	–	–	80.60
TabIERT-CNN	–	–	89.70	–	–	80.50
EINET	90.65	86.70	88.51	77.98	74.16	75.91
AutoTGRL	<b>91.22</b>	<b>91.85</b>	<b>91.53</b>	<b>82.52</b>	<b>83.21</b>	<b>82.86</b>

module, which are fed to transductive or inductive graph-based learning. We remove the large-scale text embedding module from AutoTGRL called AutoTGRL-NLTE and set the embedding of token nodes using pre-trained GloVe [35] model, while the embeddings of the document nodes are computed by the sum of token embeddings included in the document. We use AutoTGRL with transductive learning for this experiment. Considering the results shown in Table 15, we observe that the large-scale text embedding module has a pretty significant impact on our model over the datasets. We can notice that the results decline when removing the module from the AutoTGRL model.

**Table 11** Test accuracy comparison on SST datasets with baseline NAS methods and large-scale pre-trained models on text classification task

Model	SST-B	SST
ENAS-MACRO	88.90	51.55
ENAS-MICRO	87.52	47.00
DARTS	87.12	51.65
SMASH	85.94	46.65
ONE-SHOT	87.08	50.37
RANDOM SEARCH	87.15	49.20
TEXTNAS	90.33	52.51
ELMo BiLSTM	90.4	–
OpenAI GPT	91.3	–
BERT	93.39	53.2
RoBERTa	94.30	54.6
BertGCN	94.60	54.12
RoBERTaGCN	95.01	55.12
BART	92.70	–
AutoTGRL	<b>96.96</b>	<b>56.92</b>

**Table 12** Comparison of training duration between the designed TGRL models by AutoTGRL and best baselines using transductive learning

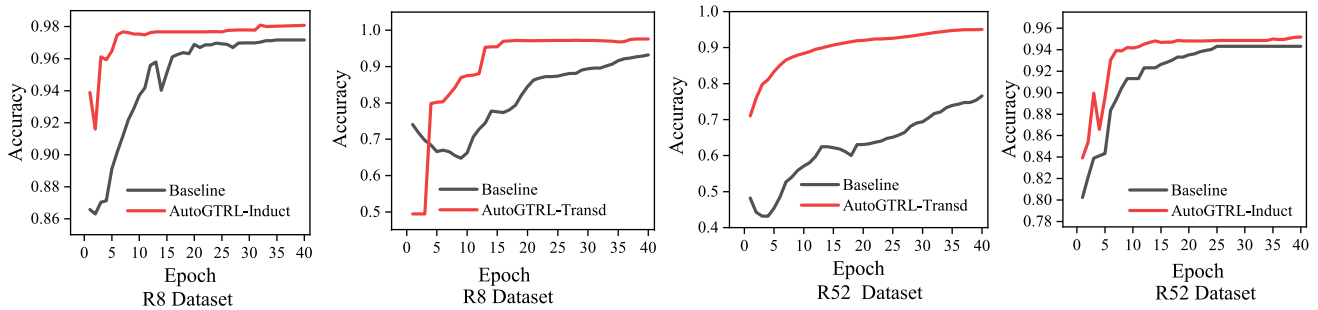
Datasets	AutoTGRL Training time	Text-GCN Training time	GCN-CNN-SB Training time
R52	223.815 s	246 s	233.45 s
R8	125.356 s	192.6	135.24 s
Ohsumed	140.791 s	252.4 s	154.52 s

#### 4.6.2 Ablation study of DAGNN layer

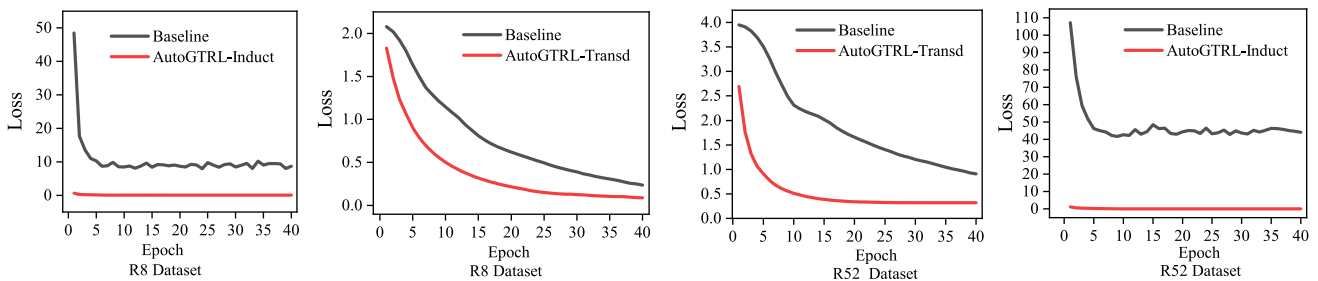
In this section, we verify the efficacy of the developed DAGNN layer by setting the number of intermediate states to 1 (equivalent to conventional GNN layer), 2, and 3 states. The GNN architecture of the TGRL model is then generated by the search algorithm depending on the number of intermediate states taken from the defined search space. From the results in Table 16, we can see that the best architecture discovered by AutoTGRL is when the intermediate states are set to 2. Additionally, it has been observed that there is a modest drop in performance when the number of intermediate states is increased to 3. This illustrates that deep GNN may indirectly collect information from unrelated nodes leading to a degradation in performance.

**Table 13** Comparison of training duration between the designed TGRL models by AutoTGRL models and best baselines using inductive learning

Datasets	AutoTGRL Training time	TLGCN Training time	GFN Training time	TextING-M Training time
Ohsumed	954.45 s	1685 s	1833 s	1018.678 s
R52	800.127 s	1175 s	1572 s	881.25 s
R8	628.24 s	839 s	907 s	671.78 s



**Fig. 5** Average accuracy of the AutoTGRL with transductive and inductive graph-based learning against the best baselines from 0 to 40 epochs



**Fig. 6** Loss value of the AutoTGRL with transductive (AutoTGRL-Transd) and inductive (AutoTGRL-Induct) graph-based learning against the best baselines from 0 to 40 epochs

**Table 14** Number of parameters (the model size) of the models identified by AutoTGRL under transductive and inductive graph-based learning against the best baseline models on R8 and R52 datasets. AutoTGRL-T means the AutoTGRL with transductive graph-based learning and AutoTGRL-I means the AutoTGRL with inductive graph-based learning

Datasets	Transductive learning		Inductive learning	
	AutoTGRL-T #Params	Baseline #Params	AutoTGRL-I #Params	Baseline #Params
R52	1,477,853	3,608,800	1,891,470	3,057,952
R8	1,469,009	3,074,000	1,189,010	2,722,908

**Table 15** The influence of large-scale text embedding module. We report the text classification accuracy after removing the large-scale text embedding module from AutoTGRL, namely, AutoTGRL-NLTE

Model	MR	R52	R8	Ohsumed
AutoTGRL	89.91	97.39	98.61	73.45
AutoTGRL-NLTE	85.31	95.32	97.56	70.45

**Table 16** AutoTGRL performance with varying number of intermediate states in the DAGNN layer over MR and R52 datasets

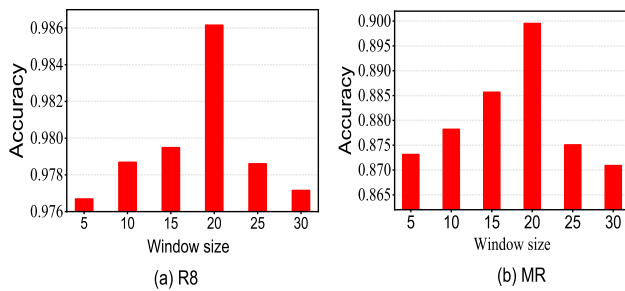
	One state	Two states	Three states
MR	87.40	<b>89.91</b>	86.2
R52	96.20	<b>97.39</b>	95.43

### 4.7 Parameter sensitivity

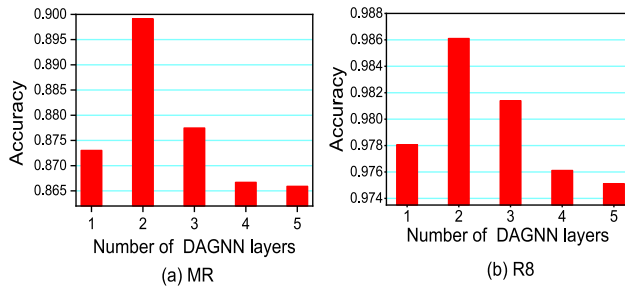
#### 4.7.1 Performance on different sliding window sizes

Figure 7 exhibits the performance of the models designed by AutoTGRL with a varying number of sliding window sizes while constructing text graphs on R8 and MR datasets

on the classification task. We can notice that choosing an appropriate window size plays an important role in test performance. We can see that as the size of the window



**Fig. 7** AutoTGRL test Accuracy by varying sliding window sizes on the classification task



**Fig. 8** AutoTGRL test accuracy with a different number of designed DAGNN layers on the classification task

gets bigger, the accuracy of the tests starts to get better. However, the average accuracy gets worse when the window size is 20 for both datasets. This suggests that small window sizes may not give enough information about how often global tokens occur together, while large window sizes may connect nodes that have nothing to do with each other.

#### 4.7.2 Performance on a different number of GNN layers

Figure 8 shows the classification performance on MR and R8 with varying numbers of designed GNN layers. We notice that two GNN layers provide the highest performance. Additionally, we see that the performance degrades as the number of layers increases. This demonstrates that deep GNN layers for text-graph representation may acquire information from a token or document nodes that are unrelated, resulting in performance reduction.

#### 4.7.3 Performance on different sizes of labeled training data

We also investigate the impact of labeled training data size. Different top-performing models are compared using different percentages of the whole dataset as training data. We present the results on R8 and MR in Fig. 9. In general, all evaluated methods achieve performance improvements with the growth of labeled training data. More surprisingly,

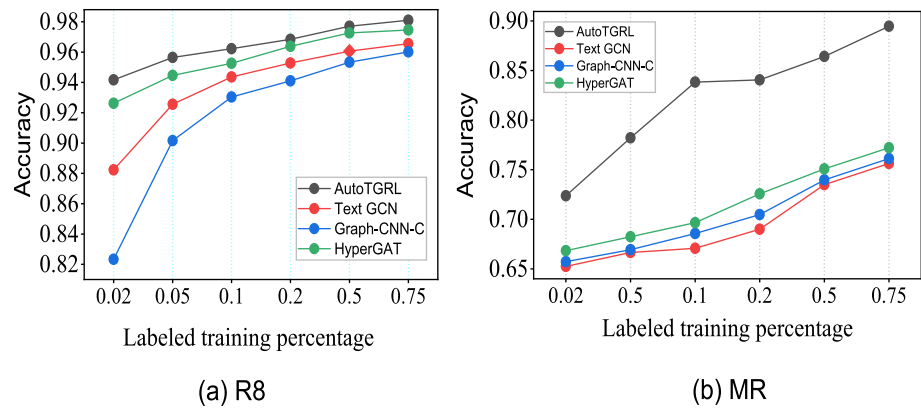
models made by AutoTGRL do much better than other baselines with less labeled data. This shows that the proposed method works well in real-world cases.

#### 4.8 Ablation study of the effect of balancing module error methods

In this section, conducted additional experiments to demonstrate the effectiveness of the two error balance methods: fine-tuning embedding method and performance-based feedback method. For the fine-tuning embedding method, we performed experiments where we fine-tune document embeddings based on truth labels involving training large-scale text embedding models on a given labeled dataset. Then, we compared the performance of this method against a baseline version of AutoTGRL, namely, AutoTGRL-FEM where no fine-tuning is applied. By measuring various evaluation metrics such as accuracy on text classification tasks, and F1-score on aspect sentiment classification tasks, we showcase how fine-tuning helps in achieving a better balance between errors across different modules. Similarly, for the performance-based feedback method, we conducted experiments that incorporate feedback from the model evaluation module to adjust the error weights of the controller of the reinforcement learning-based search algorithm in the automated TGRL model search module during training. We compared this method against a baseline version of AutoTGRL, namely, AutoTGRL-PFM where we use a random search algorithm [78]. The random search algorithm randomly selects the components of TGRL models from the TGRL search space and then ranks the chosen TGRL models based on their performances. Through these experiments, we aim to demonstrate how performance-based feedback used in reinforcement learning-based search algorithms can effectively balance errors and generate the TGRL model that performs better.

Table 17 of the revised manuscript (also shown below) shows the experiment results of the effect of the balancing module error methods. We report the accuracy scores on the text classification task and F1 scores on the ASC task after removing the fine-tuning embedding method, and performance-based feedback method when running AutoTGRL, namely, AutoTGRL-FEM and AutoTGRL-PFM. The results demonstrated that the both fine-tuning embedding method and the performance-based feedback method, namely, AutoTGRL have achieved better performance over text classification and ASC tasks. We also noticed that removing fine-tuning embedding method from our model achieved better performance than removing the performance-based feedback method, which indicates that the performance-based feedback method has a better impact on our model.

**Fig. 9** Test accuracy with varying training data proportions (2%, 5%, 10%, 20%, 50%, 75%) on R8 and MR datasets



**Table 17** The experiment results of the effect of the balancing module error methods

Model version	ASC task		Text classification task	
	Laptop F1	Restaurant F1	R52 ACC	R8 ACC
AutoTGRL-FEM	78.69	81.93	96.05	98.01
AutoTGRL-PFM	78.01	81.13	96.45	97.84
AutoTGRL	79.45	82.72	97.39	98.61

We report the accuracy scores on the text classification task and F1 scores on the ASC task after removing the fine-tuning embedding method, and performance-based feedback method when running AutoTGRL, namely, AutoTGRL-FEM, and AutoTGRL-PFM.

## 5 Conclusion

In this paper, we propose an automatic text-graph representation learning (TGRL) model based on the automatic search of the TGRL model, namely, AutoTGRL. The designed AutoTGRL framework consists of five main modules: large-scale text embedding, transductive graph-based learning, inductive graph-based learning, model evaluation, and automated TGRL model search modules. The automated TGRL model search module uses a search algorithm based on reinforcement learning with a policy gradient to generate the combination of the TGRL model from the TGRL search space and optimizes them over time. We apply the AutoTGRL model to text classification, aspect-based sentiment analysis, and entity and relation extraction tasks. The AutoTGR model can automatically identify the optimal large-scale text embedding strategy, text graph representation strategy, and GNN structure with hyperparameters to encode the given text for the downstream tasks. Another significant strength of our approach is its generality, which allows it to be applied to different text-graph representation learning tasks. Experiment results show that AutoTGRL can automatically design the optimal

TGRL models to obtain better performance over several datasets compared to the SOTA models. On the other hand, our approach has some limitations, such as the need for a predefined search space, which could limit the flexibility of the method to adapt to more complex tasks such as multiview multitask graph learning tasks. In addition, the approach does not consider multimodal tasks, which restricts its applicability in scenarios where the input data consists of multiple modalities such as images and videos. In future work, we plan to overcome these limitations by incorporating more flexible search spaces and search algorithms suitable for multiview multitask learning and multimodal learning tasks. Moreover, a more in-depth examination of how the approach would act with extremely multi-label datasets for various tasks might provide light on how the method would work for various tasks. To be able to apply AutoTGRL on a wider range of text-graph representation learning tasks, we need to adjust the candidate values of search space components and the prediction layer based on the given task.

**Author contributions** RA contributed to conceptualization, data curation, formal analysis, investigation, methodology, resources, software, validation, visualization, writing, and review. JG performed investigation, project administration, supervision, validation, and review. JC helped in data curation, editing, investigation, resources, validation, and review. BMO performed investigation, resources, editing, and review. TL performed investigation, validation, and review.

**Funding** The work is supported by the National Natural Science Foundation of China (No. 62272487).

**Data availability** All datasets used in the paper are publicly available.

**Code availability** Please email alsabiraeed@csu.edu.cn to request code for the proposed method.

## Declarations

**Conflict of interest** No conflicts of interest or competing interests.

**Ethical approval** Not applicable

**Consent to participate** The manuscript is approved by all authors for publication.

**Consent for publication** All authors who participated in this study give the publisher permission to publish this work.

## References

- Zhang Z, Zhou Z, Wang Y (2022) SSEGCN: syntactic and semantic enhanced graph convolutional network for aspect-based sentiment analysis. In: Proceedings of the 2022 conference of the North American chapter of the association for computational Linguistics: human language technologies, pp 4916–4925. <https://doi.org/10.18653/v1/2022.naacl-main.362>
- Li X, Bing L, Li P, Lam W (2019) A unified model for opinion target extraction and target sentiment prediction. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 6714–6721. <https://doi.org/10.1609/aaai.v33i01.33016714>
- Song R, Giunchiglia F, Zhao K, Tian M, Xu H (2022) Graph topology enhancement for text classification. *Appl Intell* 52(13):15091–15104. <https://doi.org/10.1007/s10489-021-03113-8>
- Tran T, Kavuluru R (2019) Neural metric learning for fast end-to-end relation extraction. arXiv preprint. [arXiv:1905.07458](https://arxiv.org/abs/1905.07458)
- Bekoulis G, Deleu J, Demeester T, Develder C (2018) Adversarial training for multi-context joint entity and relation extraction. In: Proceedings of the 2018 conference on empirical methods in natural language processing, pp 2830–2836. <https://doi.org/10.18653/v1/d18-1307>
- Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th international conference on learning representations
- Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: Proceedings of the 6th international conference on learning representations
- Liu X, You X, Zhang X, Wu J, Lv P (2020) Tensor graph convolutional networks for text classification. In: Proceedings of the AAAI conference on artificial intelligence, vol 34, pp 8409–8416. <https://doi.org/10.1609/aaai.v34i05.6359>
- Xint Y, Xu L, Guo J, Li J, Sheng X, Zhou Y (2020) Label incorporated graph neural networks for text classification. In: Proceedings of the 25th international conference on pattern recognition (ICPR), pp 8892–8898. <https://doi.org/10.1109/ICPR48806.2021.9413086>
- Osman AH, Barukub OM (2020) Graph-based text representation and matching: a review of the state of the art and future challenges. *IEEE Access* 8:87562–87583. <https://doi.org/10.1109/ACCESS.2020.2993191>
- Yao L, Mao C, Luo Y (2019) Graph convolutional networks for text classification. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 7370–7377. <https://doi.org/10.1609/aaai.v33i01.33017370>
- Huang L, Ma D, Li S, Zhang X, Wang H (2019) Text level graph neural network for text classification. In: Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing, pp 3442–3448. <https://doi.org/10.18653/v1/D19-1345>
- Zhang Y, Yu X, Cui Z, Wu S, Wen Z, Wang L (2020) Every document owns its structure: inductive text classification via graph neural networks. In: Proceedings of the 58th annual meeting of the association for computational linguistics, pp 334–339. <https://doi.org/10.18653/v1/2020.acl-main.31>
- Oloulade BM, Gao J, Chen J, Lyu T, Al-Sabri R (2021) Graph neural architecture search: a survey. *Tsinghua Sci Technol* 27(4):692–708. <https://doi.org/10.26599/TST.2021.9010057>
- Wu F, Jr AHS, Zhang T, Fifty C, Yu T, Weinberger KQ (2019) Simplifying graph convolutional networks. In: Proceedings of the 36th international conference on machine learning, vol 97, pp 6861–6871
- Ding K, Wang J, Li J, Li D, Liu H (2020) Be more with less: hypergraph attention networks for inductive text classification. In: Proceedings of the 2020 conference on empirical methods in natural language processing, pp 4927–4936. <https://doi.org/10.18653/v1/2020.emnlp-main.399>
- Liang B, Yin R, Gui L, Du J, Xu R (2020) Jointly learning aspect-focused and inter-aspect relations with graph convolutional networks for aspect sentiment analysis. In: Proceedings of the 28th international conference on computational linguistics, pp 150–161. <https://doi.org/10.18653/v1/2020.coling-main.13>
- Zhao H, Xie J, Wang H (2022) Graph convolutional network based on multi-head pooling for short text classification. *IEEE Access* 10:11947–11956. <https://doi.org/10.1109/ACCESS.2022.3146303>
- Li R, Chen H, Feng F, Ma Z, Wang X, Hovy EH (2021) Dual graph convolutional networks for aspect-based sentiment analysis. In: Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing, vol 1, pp 6319–6329. <https://doi.org/10.18653/v1/2021.acl-long.494>
- Lai TM, Ji H, Zhai C, Tran QH (2021) Joint biomedical entity and relation extraction with knowledge-enhanced collective inference. In: Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing, vol 1, pp 6248–6260. <https://doi.org/10.18653/v1/2021.acl-long.488>
- Zhang M, Qian T (2020) Convolution over hierarchical syntactic and lexical graphs for aspect level sentiment analysis. In: Proceedings of the 2020 conference on empirical methods in natural language processing, pp 3540–3549. <https://doi.org/10.18653/v1/2020.emnlp-main.286>
- Chen H, Zhai Z, Feng F, Li R, Wang X (2022) Enhanced multi-channel graph convolutional network for aspect sentiment triplet extraction. In: Proceedings of the 60th annual meeting of the association for computational linguistics, vol 1, pp 2974–2985. <https://doi.org/10.18653/v1/2022.acl-long.212>
- Elsken T, Metzen JH, Hutter F (2019) Neural architecture search: a survey. *J Mach Learn Res* 20(1):1997–2017
- Wang Y, Yang Y, Chen Y, Bai J, Zhang C, Su G, Kou X, Tong Y, Yang M, Zhou L (2020) TextNAS: a neural architecture search space tailored for text representation. In: Proceedings of the AAAI conference on artificial intelligence, pp 9242–9249. <https://doi.org/10.1609/aaai.v34i05.6462>
- Pham H, Guan M, Zoph B, Le Q, Dean J (2018) Efficient neural architecture search via parameters sharing. In: Proceedings of the 35th international conference on machine learning, vol 80, pp 4095–4104
- Liu H, Simonyan K, Yang Y (2018) DARTS: differentiable architecture search. arXiv preprint. [arXiv:1806.09055](https://arxiv.org/abs/1806.09055)
- Brock A, Lim T, Ritchie JM, Weston N (2018) SMASH: one-shot model architecture search through hypernetworks. In: Proceedings of the 6th international conference on learning representations
- Gao Y, Yang H, Zhang P, Zhou C, Hu Y (2020) Graph neural architecture search. In: Proceedings of the 29th international joint conference on artificial intelligence, pp 1403–1409. <https://doi.org/10.24963/ijcai.2020/195>

29. Gao Y, Yang H, Zhang P, Zhou C, Hu Y (2019) GraphNAS: graph neural architecture search with reinforcement learning. arXiv preprint. [arXiv:1904.09981](https://arxiv.org/abs/1904.09981)
30. Lu Z, Du P, Nie J (2020) VGCN-BERT: augmenting BERT with graph embedding for text classification. *Adv Inf Retr* 12035:369–382. [https://doi.org/10.1007/978-3-030-45439-5\\_25](https://doi.org/10.1007/978-3-030-45439-5_25)
31. Devlin J, Chang M, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, vol 1, pp 4171–4186. <https://doi.org/10.18653/v1/n19-1423>
32. Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V (2019) RoBERTa: a robustly optimized BERT pretraining approach. arXiv preprint. [arXiv:1907.11692](https://arxiv.org/abs/1907.11692)
33. Iacobacci I, Navigli R (2019) LSTMEmbed: learning word and sense representations from a large semantically annotated corpus with long short-term memories. In: Proceedings of the 57th conference of the association for computational linguistics, vol 1, pp 1685–1695. <https://doi.org/10.18653/v1/p19-1165>
34. Thost V, Chen J (2021) Directed acyclic graph neural networks. In: Proceedings of the 9th international conference on learning representations
35. Pennington J, Socher R, Manning CD (2014) Glove: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing, pp 1532–1543. <https://doi.org/10.3115/v1/d14-1162>
36. Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks?. In: Proceedings of the 7th international conference on learning representations
37. Lee JB, Rossi RA, Kim S, Ahmed NK, Koh E (2019) Attention models in graphs: a survey. *ACM Trans Knowl Discov Data* 13(6):62–16225. <https://doi.org/10.1145/3363574>
38. Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: *Advances in neural information processing systems*, vol 29, pp 3837–3845
39. Bianchi FM, Grattarola D, Livi L, Alippi C (2022) Graph neural networks with convolutional ARMA filters. *IEEE Trans Pattern Anal Mach Intell* 44(7):3496–3507. <https://doi.org/10.1109/TPAMI.2021.3054830>
40. Hamilton WL, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. *Adv Neural Inf Process Syst* 30:1024–1034
41. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res* 15(1):1929–1958. <https://doi.org/10.5555/2627435.2670313>
42. Zhang W, Li X, Deng Y, Bing L, Lam W (2021) Towards generative aspect-based sentiment analysis. In: Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing, vol 2, pp 504–510. <https://doi.org/10.18653/v1/2021.acl-short.64>
43. Wu Z, Ying C, Zhao F, Fan Z, Dai X, Xia R (2020) Grid tagging scheme for aspect-oriented fine-grained opinion extraction. arXiv preprint. [arXiv:2010.04640](https://arxiv.org/abs/2010.04640)
44. Peng H, Xu L, Bing L, Huang F, Lu W, Si L (2020) Knowing what, how and why: a near complete solution for aspect-based sentiment analysis. In: Proceedings of the AAAI conference on artificial intelligence, vol 34, pp 8600–8607. <https://doi.org/10.1609/aaai.v34i05.6383>
45. Ma Y, Hiraoka T, Okazaki N (2022) Named entity recognition and relation extraction using enhanced table filling by contextualized representations. *J Nat Lang Process* 29(1):187–223. <https://doi.org/10.5715/jnlp.29.187>
46. Ma Y, Hiraoka T, Okazaki N (2022) Joint entity and relation extraction based on table labeling using convolutional neural networks. In: Proceedings of the sixth workshop on structured prediction for NLP, pp 11–21. <https://doi.org/10.18653/v1/2022.spnlp-1.2>
47. Beltagy I, Lo K, Cohan A (2019) SciBERT: a pretrained language model for scientific text. In: Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing, pp 3613–3618. <https://doi.org/10.18653/v1/D19-1371>
48. Lin Y, Meng Y, Sun X, Han Q, Kuang K, Li J, Wu F (2021) BertGCN: transductive text classification by combining GNN and BERT. In: Findings of the association for computational linguistics: ACL-IJCNLP 2021, pp 1456–1462. <https://doi.org/10.18653/v1/2021.findings-acl.126>
49. Joulin A, Grave E, Bojanowski P, Mikolov T (2017) Bag of tricks for efficient text classification. In: Proceedings of the 15th conference of the European chapter of the association for computational linguistics, vol 2, pp 427–431. <https://doi.org/10.18653/v1/e17-2068>
50. Shen D, Wang G, Wang W, Min MR, Su Q, Zhang Y, Li C, Henao R, Carin L (2018) Baseline needs more love: on simple word-embedding-based models and associated pooling mechanisms. In: Proceedings of the 56th annual meeting of the association for computational linguistics, vol 1, pp 440–450. <https://doi.org/10.18653/v1/P18-1041>
51. Peng H, Li J, He Y, Liu Y, Bao M, Wang L, Song Y, Yang Q (2018) Large-scale hierarchical text classification with recursively regularized deep graph-CNN. In: Proceedings of the 2018 world wide web conference, pp 1063–1072. <https://doi.org/10.1145/3178876.3186005>
52. Hua S, Li X, Jing Y, Liu Q (2022) A semantic hierarchical graph neural network for text classification. arXiv preprint. [arXiv:2209.07031](https://arxiv.org/abs/2209.07031)
53. Zeng F, Chen N, Yang D, Meng Z (2022) Simplified-boosting ensemble convolutional network for text classification. *Neural Process Lett* 54(6):4971–4986. <https://doi.org/10.1007/s11063-022-10843-4>
54. Zhang C, Zhu H, Peng X, Wu J, Xu K (2022) Hierarchical information matters: text classification via tree based graph neural network. In: Proceedings of the 29th international conference on computational linguistics, pp 950–959
55. Henaff M, Bruna J, LeCun Y (2015) Deep convolutional networks on graph-structured data. arXiv preprint. [arXiv:1506.05163](https://arxiv.org/abs/1506.05163)
56. Bruna J, Zaremba W, Szlam A, LeCun Y (2014) Spectral networks and locally connected networks on graphs. In: Proceedings of the 2nd international conference on learning representations
57. Wang K, Han S.C, Poon J (2022) InducT-GCN: inductive graph convolutional networks for text classification. In: Proceedings of the 26th international conference on pattern recognition, pp 1243–1249. <https://doi.org/10.1109/ICPR56361.2022.9956075>
58. Shi J, Wu X, Liu X, Lu W, Li S (2022) Inductive light graph convolution network for text classification based on word-label graph. In: Proceedings of the international conference on intelligent information processing, vol 643, pp 42–55. [https://doi.org/10.1007/978-3-031-03948-5\\_4](https://doi.org/10.1007/978-3-031-03948-5_4)
59. Dai Y, Shou L, Gong M, Xia X, Kang Z, Xu Z, Jiang D (2022) Graph fusion network for text classification. *Knowl-Based Syst* 236:107659. <https://doi.org/10.1016/j.knsys.2021.107659>
60. Wang K, Shen W, Yang Y, Quan X, Wang R (2020) Relational graph attention network for aspect-based sentiment analysis. In: Proceedings of the 58th annual meeting of the association for



- computational linguistics, pp 3229–3238. <https://doi.org/10.18653/v1/2020.acl-main.295>
61. Tang H, Ji D, Li C, Zhou Q (2020) Dependency graph enhanced dual-transformer structure for aspect-based sentiment classification. In: Proceedings of the 58th annual meeting of the association for computational linguistics, pp 6578–6588. <https://doi.org/10.18653/v1/2020.acl-main.588>
  62. Chen C, Teng Z, Zhang Y (2020) Inducing target-specific latent structures for aspect sentiment classification. In: Proceedings of the 2020 conference on empirical methods in natural language processing, pp 5596–5607. <https://doi.org/10.18653/v1/2020.emnlp-main.451>
  63. Dai H, Song Y (2019) Neural aspect and opinion term extraction with mined rules as weak supervision. In: Proceedings of the 57th conference of the association for computational linguistics, vol 1, pp 5268–5277. <https://doi.org/10.18653/v1/p19-1520>
  64. Wang W, Pan SJ, Dahlmeier D, Xiao X (2017) Coupled multi-layer attentions for co-extraction of aspect and opinion terms. In: Proceedings of the AAAI conference on artificial intelligence, vol 31, pp 3316–3322. <https://doi.org/10.1609/aaai.v31i1.10974>
  65. Bekoulis G, Deleu J, Demeester T, Develder C (2018) Joint entity recognition and relation extraction as a multi-head selection problem. *Expert Syst Appl* 114:34–45. <https://doi.org/10.1016/j.eswa.2018.07.032>
  66. Eberts M, Ülges A (2020) Span-based joint entity and relation extraction with transformer pre-training. In: Proceedings of the 24th European conference on artificial intelligence, vol 325, pp 2006–2013. <https://doi.org/10.3233/FAIA200321>
  67. Ji B, Yu J, Li S, Ma J, Wu Q, Tan Y, Liu H (2020) Span-based joint entity and relation extraction with attention-based span-specific and contextual semantic representations. In: Proceedings of the 28th international conference on computational linguistics, pp 88–99. <https://doi.org/10.18653/v1/2020.coling-main.8>
  68. Li F, Zhang M, Fu G, Ji D (2017) A neural joint model for entity and relation extraction from biomedical text. *BMC Bioinform* 18(1):198–119811. <https://doi.org/10.1186/s12859-017-1609-9>
  69. Wu H, Huang J (2022) Joint entity and relation extraction network with enhanced explicit and implicit semantic information. *Appl Sci* 12(12):6231–6247. <https://doi.org/10.3390/app12126231>
  70. Bender G, Kindermans P, Zoph B, Vasudevan V, Le QV (2018) Understanding and simplifying one-shot architecture search. In: Proceedings of the 35th international conference on machine learning, vol 80, pp 549–558
  71. Li L, Talwalkar A (2020) Random search and reproducibility for neural architecture search. In: Proceedings of the 35th conference on uncertainty in artificial intelligence. Proceedings of machine learning research, vol 115, pp 367–377
  72. Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. In: Proceedings of the 2018 conference of the North American chapter of the association for computational linguistics: human language technologies, vol 1, pp 2227–2237. <https://doi.org/10.18653/v1/n18-1202>
  73. Radford A, Narasimhan K, Salimans T, Sutskever I et al. (2018) Improving language understanding by generative pre-training. Technical Report, OpenAI
  74. Pontiki M, Galanis D, Pavlopoulos J, Papageorgiou H, Androutsopoulos I, Manandhar S (2014) Semeval-2014 task 4: aspect based sentiment analysis. In: Proceedings of the 8th international workshop on semantic evaluation, pp 27–35. <https://doi.org/10.3115/v1/s14-2004>
  75. Dong L, Wei F, Tan C, Tang D, Zhou M, Xu K (2014) Adaptive recursive neural network for target-dependent twitter sentiment classification. In: Proceedings of the 52nd annual meeting of the association for computational linguistics, vol 2, pp 49–54. <https://doi.org/10.3115/v1/p14-2009>
  76. Fan Z, Wu Z, Dai X, Huang S, Chen J (2019) Target-oriented opinion words extraction with target-fused neural sequence labeling. In: Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, vol 1, pp 2509–2518. <https://doi.org/10.18653/v1/n19-1259>
  77. Gurulingappa H, Rajput AM, Roberts A, Fluck J, Hofmann-Apitius M, Toldo L (2012) Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *J Biomed Inform* 45(5):885–892. <https://doi.org/10.1016/j.jbi.2012.04.008>
  78. You J, Ying Z, Leskovec J (2020) Design space for graph neural networks. *Adv Neural Inf Process Syst* 33:17009–17021

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.