Compressed Heterogeneous Graph for Abstractive Multi-Document Summarization

Miao Li, Jianzhong Qi, Jey Han Lau
School of Computing and Information Systems,
The University of Melbourne
miao4@student.unimelb.edu.au, {jianzhong.qi, laujh}@unimelb.edu.au

Abstract

Multi-document summarization (MDS) aims to generate a summary for a number of related documents. We propose HGSUM — an MDS model that extends an encoder-decoder architecture to incorporate a *heterogeneous* graph to represent different semantic units (e.g., words and sentences) of the documents. This contrasts with existing MDS models which do not consider different edge types of graphs and as such do not capture the diversity of relationships in the documents. To preserve only key information and relationships of the documents in the heterogeneous graph, HGSUM uses graph pooling to compress the input graph. And to guide HGSUM to learn the compression, we introduce an additional objective that maximizes the similarity between the compressed graph and the graph constructed from the ground-truth summary during training. HGSUM is trained end-to-end with the graph similarity and standard cross-entropy objectives. Experimental results over MULTI-NEWS, WCEP-100, and ARXIV show that HGSUM outperforms state-of-the-art MDS models. The code for our model and experiments is available at: https://github.com/oaimli/HGSum.

Introduction

Multi-document summarization (MDS) aims to automatically generate a concise and informative summary for a cluster of topically related source documents (Ma et al. 2020; Radev, Hovy, and McKeown 2002). It has a wide range of applications such as creating news digests (Fabbri et al. 2019), product review summaries (Gerani et al. 2014), and summaries for scientific literature (Moro et al. 2022; Otmakhova et al. 2022). Our work targets *abstractive* MDS, which generates summaries with words that do not necessarily come from the source documents, resembling the summarization process of human beings.

State-of-the-art text summarization models use *pre-trained language models* (PLMs) including both general-purpose PLMs for text generation (Beltagy, Peters, and Cohan 2020; Lewis et al. 2020) and PLMs designed for text summarization (Zhang et al. 2020a; Xiao et al. 2022). When applied to the abstractive MDS task, these models take a flat concatenation of the (multiple) source documents, which may not capture cross-document relationships such as contradiction, redundancy, or complementary information very well (Radev 2000). Ma et al. (2020) argue that explicit modeling of cross-document relationships can potentially improve the quality of summaries. Following this, several recent studies (Li et al. 2020; Jin, Wang, and Wan 2020; Cui and Hu 2021) explore graphs to model source documents to improve abstractive MDS. However, these graphs are *homogeneous* in that the nodes or edges are not distinguished for different semantic units (e.g., words, sentences, and paragraphs) in the encoding process. This means these MDS models cannot capture the diverse cross-document relationships among different types of semantic units.

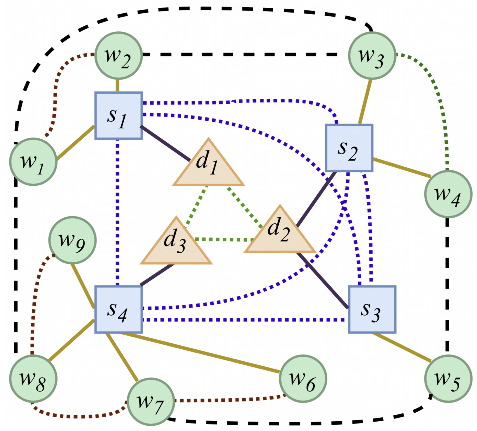


Figure 1: The structure of the heterogeneous graph given three documents in a document cluster: The orange triangles denote document nodes d, the blue quadrates denote sentence nodes s, the green circles denote word nodes w, and the line (or curve) segments between nodes denote edges. A detailed description of the graph is in the Preliminaries.

In this paper, we propose HGSUM — an MDS model that extends an encoder-decoder architecture to incorporate a heterogeneous graph to better capture the interaction between different semantic units in the documents. HGSUM's heterogeneous graph has different types of nodes and edges to model words, sentences, and documents, as shown in Figure 1. To facilitate HGSUM to learn cross-document relationships, we construct edges between sentences *across documents* based on the similarity of their sentence embeddings. We also explore compressing the graph with graph pooling to preserve only salient information (i.e., nodes and edges) that is helpful for summarization, before feeding signals from the compressed graph to the text decoder to generate the final summary. To guide HGSUM to learn this compression, we introduce an auxiliary objective that maximizes the similarity between the compressed graph and the graph derived from the ground-truth summary, in addition to the standard cross-entropy objective during training.

There are several challenges that we face. First, it is non-trivial to encode heterogeneous graphs with existing graph neural networks, as different types of nodes and edges should not be processed by the same function. To address this challenge, we propose multi-channel graph attention networks to encode heterogeneous graphs. Second, there are few graph compression or pooling methods proposed for heterogeneous graphs. Inspired by Lee, Lee, and Kang (2019), we introduce a compression method based on self-attentions to condense the heterogeneous graph. One novelty of our method is that it uses *soft masking* so that it does not break the differentiability of the network, allowing us to train HGSUM in an end-to-end manner.

To summarize, our contributions are given as follows:

- We propose HGSUM, an MDS model that extends the encoder-decoder architecture to incorporate a compressed graph to model the input documents. The graph is a heterogeneous graph that captures the diversity of semantic relationships in the documents, and it is compressed with a pooling method that helps preserve the most salient information for summarization.
- HGSUM is trained with two objectives that maximize the likelihood of generating the ground-truth summary and the similarity between the compressed graph and the graph constructed from the ground-truth summary.
- Experimental results over multiple datasets show that HGSUM outperforms state-of-the-art MDS models.

Preliminaries

Given a set of m related source documents $\mathcal{D} = \{d_0, d_1, \ldots, d_m\}$ (i.e., a document cluster), our aim is to generate a text summary $\hat{z} = \hat{w_0}, \hat{w_1}, \ldots, \hat{w_T}$ (composed of T words) that captures the essence of the source documents. As mentioned earlier, we generate the summary in an abstractive fashion, i.e., words in the generated summary can be words that are not found in the source documents. The generation of each word in the summary is modeled as:

$$\mathrm{p}(\hat{z}|\mathcal{D}) = \prod_{i=0}^{T} \mathrm{p}(\hat{w_i}|\mathcal{D}, \hat{w_0}, \hat{w_1}, \ldots, \hat{w_{i-1}}) \tag{1}$$

As heterogeneous graphs explicitly represent relationships among different semantic units (documents, sentences, and words), we construct a heterogeneous graph to represent a cluster of documents. We next explain how we construct the heterogeneous graph.

Heterogeneous Graph Construction

We denote the heterogeneous graph constructed to represent a cluster of documents as $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V}$ represents the set of nodes in the graph, and $\mathcal{E}$ the set of edges. As the example in Figure 1 shows, there are three types of nodes and six types of edges in $\mathcal{G}$. Specifically, $\mathcal{V} = \mathcal{V}_d \cup \mathcal{V}_s \cup \mathcal{V}_w$, where $\mathcal{V}_d$ is a set of document nodes: every document in the cluster corresponds to a node in $\mathcal{V}_d$ (orange triangles in Figure 1); $\mathcal{V}_s$ is a set of sentence nodes: every sentence in the documents corresponds to a node in $\mathcal{V}_s$ (blue quadrates in Figure 1); and $\mathcal{V}_t$ is a set of word nodes[1]: every word in the sentences corresponds to a node in $\mathcal{V}_w$ (green circles in Figure 1).

We next define the edges, which are all undirected:

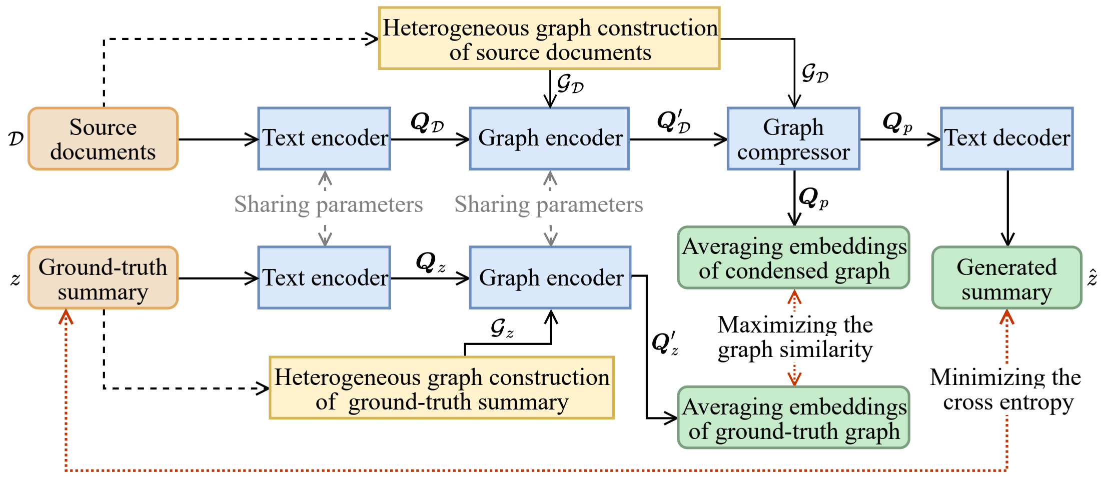
- The sets $\mathcal{E}_{we}$ and $\mathcal{E}_{wo}$ contain edges between word nodes (dash and dot lines between word nodes in Figure 1). Every edge in $\mathcal{E}_{we}$ connects two nodes corresponding to noun words (identified based on a dependency parser[2]).[3] The weight of an edge for a word pair in $\mathcal{E}_{we}$ is the cosine similarity of their embeddings. We use GloVe (Pennington, Socher, and Manning 2014) as the static word embeddings in this work. Edges in $\mathcal{E}_{wo}$, on the other hand, connect the nodes corresponding to every adjacent word pairs in a sentence. All edges in $\mathcal{E}_{wo}$ have a weight of 1.0.
- The set $\mathcal{E}_{ss}$ contains edges that connect every pair of sentences (dot lines between sentence nodes in Figure 1). The weight of an edge for a pair of sentences is the cosine similarity of their pre-trained sentence embeddings. We use Sentence-BERT (Reimers and Gurevych 2019) to compute the sentence embeddings, which is pre-trained based on the natural language inference task (Bowman et al. 2015).
- The set $\mathcal{E}_{dd}$ contains edges between document nodes (dot lines between document nodes in Figure 1). Every document is connected to all other documents in the cluster, and their edges are weighted using their n-gram overlap in terms of the average F1 value of ROUGE-1, ROUGE-2, and ROUGE-L (Lin and Hovy 2003).
- The sets $\mathcal{E}_{ds}$ and $\mathcal{E}_{st}$ contain edges that connect a document with its sentences (solid lines between document nodes and sentence nodes in Figure 1) and edges that connect a sentence with its words (solid lines between sentence nodes and words nodes in Figure 1). These edges are designed to preserve the hierarchical document-sentence and sentence-word structures. All the edge weights in these sets are set to 1.0.

To summarize, we have $\mathcal{E} = \mathcal{E}_{we} \cup \mathcal{E}_{wo} \cup \mathcal{E}_{ss} \cup \mathcal{E}_{dd} \cup \mathcal{E}_{ds} \cup \mathcal{E}_{sw}$. These edges collectively create a connected graph over all three types of nodes (words, sentences, and documents). Note that the choice of pre-trained word/sentence embeddings is flexible in our architecture, and in future work it would be interesting to explore other pre-trained embeddings.

[1]Technically, these are *subword* nodes since we use subword tokenization, although most nodes map to full words in practice.

[2]https://spacy.io/

[3]Note that nodes that do not map to a full word will not have this type of edge, since they cannot be a noun.



Heterogeneous graph construction of source documents
$\mathcal{G}_\mathcal{D}$
$\mathcal{G}_\mathcal{D}$
$\mathcal{D}$
Source documents
Text encoder
$\boldsymbol{Q}_\mathcal{D}$
Graph encoder
$\boldsymbol{Q}'_\mathcal{D}$
Graph compressor
$\boldsymbol{Q}_p$
Text decoder
Sharing parameters
Sharing parameters
$\boldsymbol{Q}_p$
Averaging embeddings of condensed graph
z
Ground-truth summary
Text encoder
$\boldsymbol{Q}_z$
Graph encoder
Generated summary
$\hat{z}$
$\mathcal{G}_z$
$\boldsymbol{Q}'_z$
Maximizing the graph similarity
Minimizing the cross entropy
Heterogeneous graph construction of ground-truth summary
Averaging embeddings of ground-truth graph

Figure 2: The HGSUM architecture: There are four main components: (1) text encoder (initialised using PRIMERA weights); (2) graph encoder; (3) graph compressor; and (4) text decoder (initialised using PRIMERA weights).

The HGSUM Model

At its core, HGSUM extends a text encoder-decoder architecture (PRIMERA; Xiao et al. (2022)) to incorporate information from a compressed heterogeneous graph derived from the input source documents, as presented in Figure 2. HGSUM has four main components: (1) text encoder (initialized using PRIMERA weights), (2) graph encoder, (3) graph compressor, and (4) text decoder (initialized using PRIMERA weights).

During training, we first generate two heterogeneous graphs $\mathcal{G}_\mathcal{D}$ and $\mathcal{G}_z$ based on the input source documents $\mathcal{D}$ and the ground-truth summary z, respectively, following the graph construction procedure described in the previous section. The text of input source documents $\mathcal{D}$ and ground-truth summary z is processed by the text encoder to obtain contextual word embeddings $\boldsymbol{Q}_\mathcal{D}$ and $\boldsymbol{Q}_z$, respectively. These contextual word embeddings are then used by the graph encoder as the initial node embeddings of $\mathcal{G}_\mathcal{D}$ and $\mathcal{G}_z$, respectively. After processed by the graph encoder, we have the graph encodings $\boldsymbol{Q}'_\mathcal{D}$ and $\boldsymbol{Q}'_z$ respectively for the source documents and the ground-truth summary.[4] The graph encoding of the source documents ($\boldsymbol{Q}'_\mathcal{D}$) will be further processed by the graph compressor to produce compressed graph encoding $\boldsymbol{Q}_p$, and this will be used by the text decoder to generate the final summary $\hat{z}$. To train HGSUM, we minimize the cross entropy between the ground-truth summary z and generated summary $\hat{z}$ and maximize the similarity between the compressed graph encoding ($\boldsymbol{Q}_p$) and ground-truth summary graph encoding ($\boldsymbol{Q}'_z$).

Once the model is trained, we only use the text and graph encoders to encode the input source documents, the graph compressor to compress the document graph, and the text decoder to decode the summary, without using any ground-truth summary as input. We next detail these components.

[4]By graph encoding we mean the collective node embeddings in the graph.

Text Encoder

The text encoder follows the encoder architecture of PRIMERA — which uses the sparse attention of longformer (Beltagy, Peters, and Cohan 2020) to accommodate long text input — and is initialized with PRIMERA weights:

$$\boldsymbol{Q}_\mathcal{D} = \text{longformer}(\mathcal{D}) \tag{2}$$
$$\boldsymbol{Q}_z = \text{longformer}(z) \tag{3}$$

The text encoder takes as input a concatenated string containing all the words from the documents, and it produces contextualized embeddings for these words as the output ($\boldsymbol{Q}_\mathcal{D}$ for source documents and $\boldsymbol{Q}_z$ for the ground-truth summary). Note that we use special delimiters $\langle$*sent-sep*$\rangle$ and $\langle$*doc-sep*$\rangle$ to mark sentence and document boundaries, which allows us to extract sentence and document embeddings that we use as the initial sentence and document node embeddings in the graph encoder.

Graph Encoder

The graph encoder is responsible for learning node embeddings for the document graph $\mathcal{G}_\mathcal{D}$ and the ground-truth summary graph $\mathcal{G}_z$. We explain how the graph encoder works for the document graph below, but the same principle works for processing the ground-truth summary graph.

Node embeddings for the heterogeneous graph $\mathcal{G}_\mathcal{D}$ represent the words, sentences, and documents, and they are initialized using the contextual embeddings learned from the text encoder ($\boldsymbol{Q}_\mathcal{D}$). As standard graph neural networks (GNNs) based on message passing cannot be applied to the heterogeneous graphs directly, we propose *multi-channel graph attention networks* (MGAT) inspired by graph attention networks (GAT; Velickovic et al. (2018)) to encode the heterogeneous graph.

Similar to GAT, MGAT is a multi-layer graph network. Intuitively, in each layer, MGAT aggregates embeddings of different channels (i.e., edge types) for each node. The computation of the l-th layer of MGAT is given as follows:

$$\boldsymbol{h}_i^{(l+1)} = \boldsymbol{U}\boldsymbol{H}_i^{(l)} \tag{4}$$

$$\boldsymbol{H}_i^{(l)} = \|_{c=1}^{C} \boldsymbol{h}_i^{(l),c} \tag{5}$$

where $\boldsymbol{h}_i^{(l+1)}$ is the output embedding of node i in the l-th layer, $\|$ is the concatenation operation, C is the number of channels (which equals to the number of edge types in the heterogeneous graph, six in our case), and $\boldsymbol{U}$ is the shared transformation matrix for different nodes. Intuitively, $\boldsymbol{h}_i^{(l),c}$ represents the embedding of node i in the c-th channel at the l-th layer, and $\boldsymbol{H}_i^{(l)}$ is the concatenation of node embeddings from all channels for node i in the l-th layer. Note that the input node embeddings of the first layer of any channel are the output contextual embeddings (words, sentences, and documents) of the text encoder, i.e., $\boldsymbol{h}_i^{(0)} = \boldsymbol{q}_i$ where $\boldsymbol{q}_i \in \boldsymbol{Q}_{\mathcal{D}}$. The graph encoding, $\boldsymbol{Q}'_{\mathcal{D}}$, consists of all updated node embeddings from the final layer, i.e., $\boldsymbol{Q}'_{\mathcal{D}} = \|_i \boldsymbol{h}_i^{(L)}$.

To compute $\boldsymbol{h}_i^{(l),c}$ in each channel:

$$\boldsymbol{h}_i^{(l),c} = \|_{m=1}^{M} \sigma\Big(\sum_{j\in\mathcal{N}_i^c} \alpha_{ij}^{m,c} \boldsymbol{W}^{m,c} \boldsymbol{h}_j^{(l),c}\Big) \tag{6}$$

where M is the number of attention heads. We can now see that $\boldsymbol{h}_i^{(l),c}$ is the concatenated representation of M independent attention heads with different weight matrices $\boldsymbol{W}^{m,c}$ and normalized attention weights $\alpha_{ij}^{m,c}$, with the latter computed as follows:

$$\alpha_{ij}^{m,c} = \frac{\exp(d_{ij}^{m,c})}{\sum_{k\in\mathcal{N}_i^c} \exp(d_{ik}^{m,c})} \tag{7}$$

where $\mathcal{N}_i^c$ denotes the set of nodes connected to node i by an edge of type c. The attention coefficient $d_{ij}^{m,c}$ represents the correlation between nodes, and is learned as follows:

$$d_{ij}^{m,c} = \sigma\big(e_{ij} \cdot \boldsymbol{w}_{m,c}^{\top}[\boldsymbol{W}^{m,c}\boldsymbol{h}_i^{(l),c} \| \boldsymbol{W}^{m,c}\boldsymbol{h}_j^{(l),c}]\big) \tag{8}$$

where e_{ij} is the edge weight between node i and node j (defined in the Preliminaries section).

To summarize, MGAT computes node embeddings by attending to neighbouring nodes just like GAT, but it does this for each edge type independently and then concatenates them together to produce the final node embeddings, and it repeats this for multiple layers/iterations to learn higher order connections. We note that HGSum has only one graph encoder, which is used to process both the source document graph $\mathcal{G}_{\mathcal{D}}$ to produce $\boldsymbol{Q}'_{\mathcal{D}}$ and the ground-truth summary graph $\mathcal{G}_z$ to produce $\boldsymbol{Q}'_z$.

Graph Compressor

Given $\mathcal{G}_{\mathcal{D}}$ and $\boldsymbol{Q}'_{\mathcal{D}}$ from the graph encoder, the graph compressor aims to "compress" the graph by selecting a subset of salient nodes and edges. Here we focus on filtering the sentence nodes, because we want to identify key sentences that help generate the summary. After the compression, all selected sentence nodes *and* their linked word and document nodes represent the compressed graph and their embeddings will be used by the text decoder for summary generation.

The graph compressor is inspired by Lee, Lee, and Kang (2019), and it works by computing the attention scores for all sentence nodes, filtering out nodes with the lowest scores, and then masking the rest using their attention scores. Firstly, attention scores of the sentence nodes are calculated based on the updated node embeddings from our proposed graph encoder $\mathrm{MGAT}(\boldsymbol{Q}_{\mathcal{D}}, \mathcal{G}_{\mathcal{D}})$:

$$\boldsymbol{t} = \mathrm{softmax}(\mathrm{MGAT}(\boldsymbol{Q}_{\mathcal{D}}, \mathcal{G}_{\mathcal{D}}) \cdot \boldsymbol{r}) \tag{9}$$

where $\boldsymbol{r}$ is the only trainable parameter of the graph compressor which transforms the updated node embedding into a scalar. Then, based on these scores, we select sentence nodes with the highest scores:

$$\mathcal{I}_s = \text{top-k}(\boldsymbol{t}, k, \mathcal{G}_{\mathcal{D}}) \tag{10}$$

$$\mathcal{I} = \mathrm{extend}(\mathcal{I}_s, \mathcal{G}_{\mathcal{D}}) \tag{11}$$

where top-k is a function that selects top-ranked sentence nodes in $\mathcal{G}_{\mathcal{D}}$ based on $\boldsymbol{t}$, $k \in (0, 1]$ is a hyper-parameter that determines the ratio of sentence nodes to be kept, $\mathcal{I}_s$ is the set of selected sentence nodes, and extend is a function that extends the selected sentence nodes in $\mathcal{I}_s$ to include word and document nodes that they link to (and so $\mathcal{I}$ includes word, sentence and document nodes). Lastly, we mask all the selected nodes using their attention scores, producing the encoding of the compressed graph, $\boldsymbol{Q}_p$:

$$\boldsymbol{Q}_p = \|_i^{\mathcal{I}} \boldsymbol{q}'_i \times \boldsymbol{t}_i, \boldsymbol{q}'_i \in \boldsymbol{Q}'_{\mathcal{D}}. \tag{12}$$

Text Decoder

The text decoder follows the same architecture as a decoder Transformer (which uses masked attention to prevent attention to future words), is initialized with PRIMERA weights, and takes $\boldsymbol{Q}_p$ as input to generate the summary:

$$\hat{z} = \mathrm{transformer}(\boldsymbol{Q}_p) \tag{13}$$

Note that the node embeddings in $\boldsymbol{Q}_p$ retain the original word index in the source documents $\mathcal{D}$, and as such positional embeddings are added to them following standard transformer architecture.

Multi-Task Training

HGSum is trained with two objectives: maximizing the likelihood of generating the ground-truth summary z and the graph similarity between the compressed graph encoding $\boldsymbol{Q}_p$ and ground-truth summary graph encoding $\boldsymbol{Q}'_z$.

To maximize the likelihood of generating the ground-truth summary, we minimize the cross entropy over the ground-truth summary and the generated summary with conventional teacher forcing.

$$\mathcal{L}_{ce} = -\frac{1}{T}\sum_{i=1}^{T} w_i \log \hat{w}_i \tag{14}$$

where w_i is the i-th word in the ground-truth summary, while $\hat{w}_i$ is the i-th word in the generated summary.

To maximize the graph similarity, we compute the cosine similarity of the average node embeddings from the compressed graph and the ground-truth summary graph:

$$\mathcal{L}_{gs} = -\,\mathrm{sim}(\mathrm{avg}(\boldsymbol{Q}_p), \mathrm{avg}(\boldsymbol{Q}'_z)) \tag{15}$$

Dataset	#c	#d/c	#w/d	#w/s
MULTI-NEWS	56,216	2.79	690.97	241.61
WCEP-100	10,200	63.38	439.24	30.53
ARXIV	215,913	5.63	978.17	251.07

Table 1: Dataset statistics. "c" = cluster; "d" = document; "w" = word; and "s" = summary. "#" denotes "the number of" and "/" denotes "in each".

Model	#parameters	Len-in	Len-out
PEGASUS	568M	1,024	512
LED	459M	16,384	512
PRIMERA	447M	4,096	512
MGSum	129M	2,000	400
GraphSum	463M	4,050	300
HGSUM	501M	4,096	512

Table 2: Model parameter sizes. Len-in and Len-out denote the maximum lengths of the model input and the model output, respectively.

The final loss function of HGSUM is the sum of $\mathcal{L}_{ce}$ and $\mathcal{L}_{gs}$ weighted by hyper-parameter $\beta \in (0, 1)$.

$$\mathcal{L} = \beta\mathcal{L}_{ce} + (1 - \beta)\mathcal{L}_{gs} \tag{16}$$

Experiments

We test our proposed model HGSUM and compare it against state-of-the-art abstractive MDS models over several datasets. We also report the results of an ablation study to show the effectiveness of the components of HGSUM.

Experimental Setup

Datasets We use MULTI-NEWS (Fabbri et al. 2019), WCEP-100 (Ghalandari et al. 2020), and ARXIV (Cohan et al. 2018) as benchmark English datasets. These datasets come from different domains including news, Wikipedia, and scientific domains. MULTI-NEWS contains clusters of news articles plus a summary corresponding to each cluster written by professional editors. WCEP-100 contains human-written summaries of different news events from Wikipedia. In ARXIV, each cluster corresponds to a research paper in the scientific domain, where the paper abstract is used as the summary, while sections of the paper are used as the source documents in each cluster. Table 1 summarizes statistics of these datasets.

Competitors We compare our model with two groups of state-of-the-art abstractive MDS models: *PLM-based* and *graph-based*. (1) The PLM-based models include **PEGASUS** (Zhang et al. 2020a), **LED** (Beltagy, Peters, and Cohan 2020), and **PRIMERA** (Xiao et al. 2022). LED is a general-purpose PLM that introduces the longformer architecture which uses sparse self-attention to allow it to process much longer input than previous models. LED is pre-trained by reconstructing documents from their corrupted input in the same way as BART (Lewis et al. 2020). In contrast, PEGASUS and PRIMERA are pre-trained models designed for summarization (the former for single-document and the latter multi-document summarization). Specifically, PEGASUS is pre-trained by generating pseudo summaries for documents, where the pseudo summaries are composed of gap sentences extracted from a document based on ROUGE scores. PRIMERA is similarly pre-trained to generate pseudo summaries, but their pseudo summaries are extracted based on the salience of entities which correspond to their document frequency. For these PLM-based models, we take their off-the-shelf models and fine-tune them on our datasets. We follow the standard approach where we concatenate documents from the same cluster to form a long and flat input string. (2) For the graph-based models, we compare against **MGSum** (Jin, Wang, and Wan 2020)[5] and **GraphSum** (Li et al. 2020)[6]. To model cross-document relationships in MDS, MGSum (Jin, Wang, and Wan 2020) uses a three-level hierarchical graph to represent source documents, including different levels of nodes (documents, sentences, and words). It learns semantics with a multi-level interaction network. Although there are different types of nodes in this hierarchical graph, all of its edges are of the same type (i.e., it is a homogeneous graph).[7] GraphSum (Li et al. 2020) uses a similarity graph over paragraphs to capture cross-document relationships, and it uses pre-trained RoBERTa (Liu et al. 2019) as its encoder. Just like MGSum, its graph is homogeneous.

Implementation Details For the PLMs, we use the large version of the models which roughly have the same number of parameters (Table 2).[8] For the graph-based models, we use open-source code from the original authors and train them on our datasets, following their recommended hyper-parameters and configurations. As Table 2 shows, most models are trained to generate a maximum length of 512 subwords ("Len-out") for the summary (exception: MGSum and GraphSum where we follow the original output length). Note though that the maximum input lengths ("Len-in") of these models range from 1K-16K subwords, depending on the architecture of their encoder.

For HGSUM, the text encoder and decoder are initialized with PRIMERA weights. To alleviate overfitting, we apply label smoothing during training with a smoothing factor of 0.1. We use beam search decoding with beam width 5 to generate the summary. The hyper-parameter β is set to 0.5 to balance two loss functions. All other hyper-parameters are tuned based on the development set.

All experiments are run on Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz with NVIDIA Tesla A100 GPU (40G).

Overall Results

We report the average F1 of ROUGE-1 (R-1), ROUGE-2 (R-2) and ROUGE-L (R-L) (Lin and Hovy 2003). Note that we

[5]https://github.com/zhongxia96/MGSum

[6]https://github.com/PaddlePaddle/Research/tree/master/NLP

[7]For fair comparison we use the abstractive variant of MGSum.

[8]PLM-based models are implemented using the HuggingFace library: https://huggingface.co/

Model	MULTI-NEWS			WCEP-100			ARXIV		
	R-1	R-2	R-L	R-1	R-2	R-L	R-1	R-2	R-L
PEGASUS	47.70	18.36	43.62	42.43	17.33	32.35	44.21	16.95	38.87
LED	47.68	19.72	43.83	43.05	20.94	34.99	46.50	18.96	41.87
PRIMERA	49.40	20.51	45.35	43.11	**21.85**	35.89	47.24	20.24	42.61
MGSum	45.63	16.71	40.92	38.88	14.22	23.37	40.58	11.22	29.93
GraphSum	45.71	17.12	41.99	39.56	14.38	29.41	42.98	16.55	37.01
HGSUM (our model)	**50.64**†	**21.69**†	**45.90**†	**44.21**†	21.81	**36.21**†	**49.32**†	**21.30**†	**44.50**†
Performance gain	+2.51%	+5.75%	+1.21%	+2.55%	-0.18%	+0.89%	+4.40%	+5.24%	+4.44%

Table 3: Model performance on summarizing MULTI-NEWS, WCEP-100, and ARXIV in terms of F1 of ROUGE scores. The best performance results are in boldface, while the second best is underlined. †: significantly better than others (p-value < 0.05).

Doc 1	*. . . Parents are risking their babies' health because of a surge in the popularity of swaddling . . .*
Doc 2	*There has been a recent resurgence of swaddling because of . . .*
Doc 3	*. . . Swaddling babies is on the rise: Add it to the long list of mixed messages new parents get about infant care . . .*
Generated summary	*The trend of swaddling babies is on the rise, but an orthopaedic surgeon . . . is warning parents against the practice.*

Table 4: An example of a generated summary in MULTI-NEWS by HGSUM.

use the summary-level R-L,[9] and each summary is split into sentences using NLTK[10].

Table 3 reports the performance of all models over all datasets. HGSUM outperforms most of the benchmark systems, demonstrating the effectiveness of incorporating a compressed heterogeneous graph for text summarization. Interestingly, the PLMs (PEGASUS, LED, PRIMERA, and HGSUM) also seem to be consistently better than graph-based models (MGSum and GraphSum). This shows that using graph-based document representations does not necessarily lead to better MDS results, thus confirming the advantage of our heterogeneous graph-based model design. We give an example of generated summary by HGSUM in MULTI-NEWS in Table 4.

Ablation Study

To show the effectiveness of the HGSUM components, we conduct an ablation study and compare it with three model variants: (1) HGSUM **w/o MGAT**, which replaces MGAT with the vanilla GAT model that treats all graph nodes and edges as being the same type, (2) HGSUM **w/o graph compressor**, which drops the graph compressor from HGSUM and uses the output from the graph encoder directly as the input for the text decoder, and (3) HGSUM **w/o multi-task training**, which replaces the multi-task objective using only the cross entropy objective.

Model	R-1	R-2	R-L	BScore
HGSUM	50.64	21.69	45.90	87.38
w/o MGAT	48.87	20.32	43.21	87.08
w/o graph compressor	49.00	20.38	45.01	86.92
w/o multi-task training	48.10	20.30	44.24	86.85

Table 5: Results of ablation study on MULTI-NEWS.

Initialized by	R-1	R-2	R-L	BScore
random weights	18.99	27.86	16.88	79.32
LED	48.36	19.99	44.25	86.73
PRIMERA	50.64	21.69	45.90	87.38

Table 6: Summarization results of HGSUM with different initialization on MULTI-NEWS.

For the ablation results, we also present the performance in terms of BERTScore ("BScore"; Zhang et al. (2020b)), which measures the semantic similarity between the ground-truth and generated summary based on BERT embeddings. Table 5 shows the ablation results on the test set of MULTI-NEWS.[11] We see that removing the heterogeneous graph encoder, graph compressor, or the multi-task objective result in a performance drop over all metrics, confirming the effectiveness of these components. In particular, dropping the multi-task objective leads to the largest degradation in model performance, suggesting that this auxiliary task is essential to help HGSUM learn how to compress the graph for summarization.

More Analysis

Impact of Text Encoder and Decoder Initialization Our text encoder and decoder can be initialized by any pre-trained Transformer models. Here we make a comparison

[9]We note that prior studies use a mixture of summary-level and sentence-level R-L, and for more details about their differences, we refer the reader to: https://pypi.org/project/rouge-score/

[10]https://www.nltk.org/

[11]We found similar results for different datasets, and present only MULTI-NEWS here in light of space.

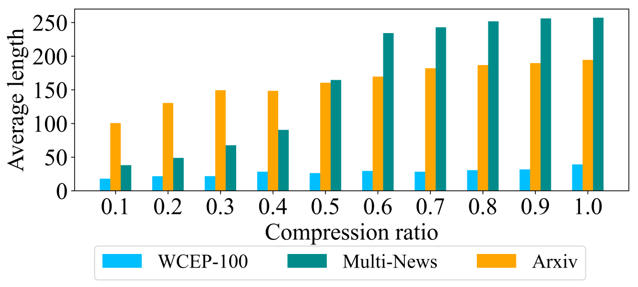


Figure 3: Average lengths of generated summaries for different datasets when the compression ratio k is set to different values.

on initialization using PRIMERA, the large version of LED and random weights. Table 6 shows results using such initialization strategies on the test set of MULTI-NEWS. We see that initialization with random weights has much worse performance than initialization using pre-trained PLMs, which is expected. Using PRIMERA leads to better empirical performance than using the LED, consistent with prior findings.

Impact of the Graph Compression Ratio k The hyperparameter k in the heterogeneous graph pooling is to control the proportion of sentence nodes to be retained in the compressed graph. To understand how much k affects the generated summary length, we present average lengths of generated summaries for different datasets when the compression ration k is set to different values in Figure 3. Interestingly, we see that larger k generally produces longer summary, and this effect is strongest for MULTI-NEWS.

Related Work

Abstractive Multi-Document Summarization

PLM-Based Models Recent PLM-based models have shown strong performance for abstractive text summarization tasks. These models follow a Transformer-based (Vaswani et al. 2017) encoder-decoder architecture. For example, general-purpose PLMs such as T5 (Raffel et al. 2020), BART (Lewis et al. 2020), and LED (Beltagy, Peters, and Cohan 2020) can be fine-tuned for abstractive text summarization. PEGASUS (Zhang et al. 2020a) is a strong PLM-based model pre-trained with an objective that predicts gap sentences as a pseudo summary. These models can be used for MDS by concatenating the source documents into a single document. PRIMERA (Xiao et al. 2022) has the same architecture as LED, but is designed for MDS specifically in that it is pre-trained to generate pseudo summaries — text spans that are automatically extracted based on the entity salience. Although these models show impressive performances and can even handle zero-shot cases, they use a flat concatenation of the input documents, which limits their capability in learning the cross-document relationships among different semantic units.

Graph-Based Models Although graphs are commonly used to boost text summarization (Wu et al. 2021b; You et al. 2022; Song and King 2022), there are only a handful of models which have been proposed to use graphs to encode the documents in abstractive MDS (Li et al. 2020; Jin, Wang, and Wan 2020; Li and Zhuge 2021; Cui and Hu 2021). Most of these models only leverage homogeneous graphs as they do not consider different edge types of graphs. For example, MGSum (Jin, Wang, and Wan 2020) constructs a three-level (i.e., document, sentence, and word levels) hierarchical graph and learns semantics with a multi-level interaction network. GraphSum (Li et al. 2020) constructs a similarity graph over the paragraphs. It learns a graph representation for the paragraphs and uses a hierarchical graph attention mechanism to guide the summary generation process. The graphs constructed in these models are in fact homogeneous, in that GraphSum only consider paragraph nodes, and MGSum uses the same edge type to connect the graph nodes.

Graph Neural Networks

Graph Modeling GNNs have yielded strong performance for modeling documents (Wu et al. 2021a), e.g., to model relationships among text spans for MDS. Graph convolutional networks (GCN; Kipf and Welling (2017)) and graph attention networks (GAT; Velickovic et al. (2018)) are two representative GNN models, which are frequently used in modeling graph-structured data composed of nodes and edges. GAT is based on the attention mechanism (Vaswani et al. 2017), while GCN is based on Laplacian transformation on the adjacency matrix. Another difference between these two is that edge weights of GCNs (i.e., the adjacency matrix) are fixed in training but those of GAT (i.e., the attentions) can be updated, although both of them perform message passing (Gilmer et al. 2017) on graphs.

Graph Pooling Graph pooling (Liu et al. 2022) aggregates node embeddings to obtain compressed graph representations. Existing graph pooling methods can be largely grouped into two categories: *global pooling* and *hierarchical pooling*. Global pooling generates the graph representation with a mean- or sum-pooling over the node embeddings. This method does not preserve the hierarchical structure of graphs. Hierarchical pooling, in contrast, considers the graph structure by compressing an input graph into smaller graphs iteratively, through node clustering (Bianchi, Grattarola, and Alippi 2020) or node dropping (Lee, Lee, and Kang 2019). Our graph compressor follows the idea of the hierarchical pooling, and condenses the graph by removing nodes to generate a small-sized graph.

Conclusion

We propose HGSUM, an extended encoder-decoder model that builds on PLMs to incorporate a compressed heterogeneous graph for abstractive multi-document summarization. HGSUM is novel in that it captures the heterogeneity between words, sentences, and document units in the constructed graph for source documents, and it also learns to compress the heterogeneous graph by 'mimicking' the ground-truth summary graph during training. Experimental results over multiple datasets show that HGSUM outperforms current state-of-the-art MDS systems.

Acknowledgements

We would like to thank the anonymous reviewers for their great comments. This work is supported by China Scholarship Council (CSC).

References

Beltagy, I.; Peters, M. E.; and Cohan, A. 2020. Longformer: The Long-Document Transformer. *CoRR*, abs/2004.05150.

Bianchi, F. M.; Grattarola, D.; and Alippi, C. 2020. Spectral Clustering with Graph Neural Networks for Graph Pooling. In *ICML*, 874–883.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*, 632–642.

Cohan, A.; Dernoncourt, F.; Kim, D. S.; Bui, T.; Kim, S.; Chang, W.; and Goharian, N. 2018. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. In *NAACL-HLT*, 615–621.

Cui, P.; and Hu, L. 2021. Topic-Guided Abstractive Multi-Document Summarization. In *Findings of EMNLP*, 1463–1472.

Fabbri, A. R.; Li, I.; She, T.; Li, S.; and Radev, D. R. 2019. Multi-News: A Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model. In *ACL*, 1074–1084.

Gerani, S.; Mehdad, Y.; Carenini, G.; Ng, R. T.; and Nejat, B. 2014. Abstractive Summarization of Product Reviews Using Discourse Structure. In *EMNLP*, 1602–1613.

Ghalandari, D. G.; Hokamp, C.; Pham, N. T.; Glover, J.; and Ifrim, G. 2020. A Large-Scale Multi-Document Summarization Dataset from the Wikipedia Current Events Portal. In *ACL*, 1302–1308.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*, 1263–1272.

Jin, H.; Wang, T.; and Wan, X. 2020. Multi-Granularity Interaction Network for Extractive and Abstractive Multi-Document Summarization. In *ACL*, 6244–6254.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-Attention Graph Pooling. In *ICML*, 3734–3743.

Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; and Zettlemoyer, L. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *ACL*, 7871–7880.

Li, W.; Xiao, X.; Liu, J.; Wu, H.; Wang, H.; and Du, J. 2020. Leveraging Graph to Improve Abstractive Multi-Document Summarization. In *ACL*, 6232–6243.

Li, W.; and Zhuge, H. 2021. Abstractive Multi-Document Summarization Based on Semantic Link Network. *TKDE*, 33(1): 43–54.

Lin, C.; and Hovy, E. H. 2003. Automatic Evaluation of Summaries Using N-gram Co-occurrence Statistics. In *HLT-NAACL*, 71–78.

Liu, C.; Zhan, Y.; Li, C.; Du, B.; Wu, J.; Hu, W.; Liu, T.; and Tao, D. 2022. Graph Pooling for Graph Neural Networks: Progress, Challenges, and Opportunities. *CoRR*, abs/2204.07321.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Ma, C.; Zhang, W. E.; Guo, M.; Wang, H.; and Sheng, Q. Z. 2020. Multi-document Summarization via Deep Learning Techniques: A Survey. *CoRR*, abs/2011.04843.

Moro, G.; Ragazzi, L.; Valgimigli, L.; and Freddi, D. 2022. Discriminative Marginalized Probabilistic Neural Method for Multi-Document Summarization of Medical Literature. In *ACL*, 180–189.

Otmakhova, Y.; Verspoor, K.; Baldwin, T.; and Lau, J. H. 2022. The Patient Is More Dead than Alive: Exploring the Current State of the Multi-document Summarisation of the Biomedical Literature. In *ACL*, 5098–5111.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*, 1532–1543.

Radev, D. R. 2000. A Common Theory of Information Fusion from Multiple Text Sources Step One: Cross-Document Structure. In *SIGDIAL Workshop*, 74–83.

Radev, D. R.; Hovy, E. H.; and McKeown, K. R. 2002. Introduction to the Special Issue on Summarization. *Computational Linguistics*, 28(4): 399–408.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR*, 21: 140:1–140:67.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*, 3980–3990.

Song, Z.; and King, I. 2022. Hierarchical Heterogeneous Graph Attention Network for Syntax-Aware Summarization. In *AAAI*, 11340–11348.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *NeurIPS*, 5998–6008.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *ICLR*.

Wu, L.; Chen, Y.; Shen, K.; Guo, X.; Gao, H.; Li, S.; Pei, J.; and Long, B. 2021a. Graph Neural Networks for Natural Language Processing: A Survey. *CoRR*, abs/2106.06090.

Wu, W.; Li, W.; Xiao, X.; Liu, J.; Cao, Z.; Li, S.; Wu, H.; and Wang, H. 2021b. BASS: Boosting Abstractive Summarization with Unified Semantic Graph. In *ACL*, 6052–6067.

Xiao, W.; Beltagy, I.; Carenini, G.; and Cohan, A. 2022. PRIMERA: Pyramid-based Masked Sentence Pre-training for Multi-document Summarization. In *ACL*, 5245–5263.

You, J.; Li, D.; Kamigaito, H.; Funakoshi, K.; and Okumura, M. 2022. Joint Learning-based Heterogeneous Graph Attention Network for Timeline Summarization. In *NAACL*, 4091–4104.

Zhang, J.; Zhao, Y.; Saleh, M.; and Liu, P. J. 2020a. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. In *ICML*, 11328–11339.

Zhang, T.; Kishore, V.; Wu, F.; Weinberger, K. Q.; and Artzi, Y. 2020b. BERTScore: Evaluating Text Generation with BERT. In *ICLR*.