# Graph Representation of Tables+Text and Compact Subgraph Retrieval for QA Tasks

Vishwajeet Kumar[1(✉)], Jaydeep Sen[1], Bhawna Chelani[2], and Soumen Chakrabarti[2]

[1] IBM Research, Gurgaon, India
{vishk024,jaydesen}@in.ibm.com
[2] IIT Bombay, Mumbai, India
soumen@cse.iitb.ac.in

**Abstract.** Large language models (LLMs) are increasingly used for table+text question answering (QA), but there is insufficient focus on unified table and text representation needed for effective retrieval guidance. In response, we present TabSegNet, comprised of (1) a fine-tuned LLM to decompose the question into segments that access individual tables;(2) a unified graph representation of tables+text, where retrieval amounts to identifying a compact node subset collectively covering the question segments; and (3) a novel graph neural network (GNN) whose messages are informed by the question and its segmentation. Experiments with existing and newly-created data sets demonstrate the promise of our approach, compared to sparse and dense nearest neighbor search, or using an LLM for retrieval.

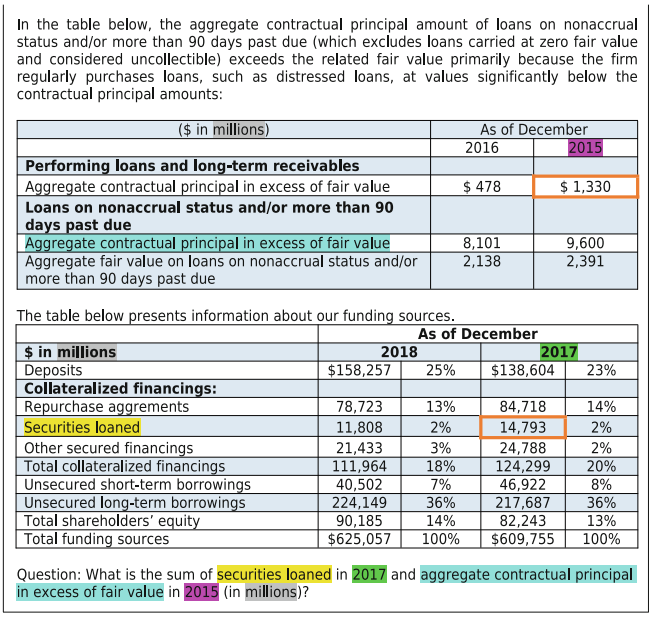
## 1 Introduction

Passage-based QA was among the earliest successful applications of large language models (LLMs), particularly when assisted with query-time retrieval. To satisfy the needs of many QA applications that benefit from tabular data, many systems converted tables into linear text, appended the question, encoded this sequence using an LLM, which then decoded the response [4,5,13]. Early LLMs were unable to support large tables, given their input length limits and quadratic attention structure. Even recent LLMs can get overwhelmed if the answer has to be inferred across multiple large tables.

A compelling enterprise scenario involves long documents (e.g., financial reports), with many embedded tables with captions [6,14,28]. Tables and their elements are referenced from text. Cells may have textual and numeric content. A complex question is asked in the context of one or few such documents, without specifying which tables are relevant. Elements or slices from relevant table(s) need to be accessed to answer the question, together with other text elements. These whole documents may be too large to present to even recent LLMs. Moreover, indiscriminate table linearization may lose vital structural clues.

Graphs provide a uniform representation of such multimodal objects. They have already been used to represent tables in isolation [22,26]. However, this


C. Hauff et al. (Eds.): ECIR 2025, LNCS 15572, pp. 164–180, 2025.
https://doi.org/10.1007/978-3-031-88708-6_11



In the table below, the aggregate contractual principal amount of loans on nonaccrual status and/or more than 90 days past due (which excludes loans carried at zero fair value and considered uncollectible) exceeds the related fair value primarily because the firm regularly purchases loans, such as distressed loans, at values significantly below the contractual principal amounts:

| ($ in millions) | As of December | |
|---|---|---|
| | 2016 | 2015 |
| **Performing loans and long-term receivables** | | |
| Aggregate contractual principal in excess of fair value | $ 478 | $ 1,330 |
| **Loans on nonaccrual status and/or more than 90 days past due** | | |
| Aggregate contractual principal in excess of fair value | 8,101 | 9,600 |
| Aggregate fair value on loans on nonaccrual status and/or more than 90 days past due | 2,138 | 2,391 |

The table below presents information about our funding sources.

| | As of December | | | |
|---|---|---|---|---|
| **$ in millions** | **2018** | | **2017** | |
| Deposits | $158,257 | 25% | $138,604 | 23% |
| **Collateralized financings:** | | | | |
| Repurchase aggrements | 78,723 | 13% | 84,718 | 14% |
| Securities loaned | 11,808 | 2% | 14,793 | 2% |
| Other secured financings | 21,433 | 3% | 24,788 | 2% |
| Total collateralized financings | 111,964 | 18% | 124,299 | 20% |
| Unsecured short-term borrowings | 40,502 | 7% | 46,922 | 8% |
| Unsecured long-term borrowings | 224,149 | 36% | 217,687 | 36% |
| Total shareholders' equity | 90,185 | 14% | 82,243 | 13% |
| Total funding sources | $625,057 | 100% | $609,755 | 100% |

Question: What is the sum of securities loaned in 2017 and aggregate contractual principal in excess of fair value in 2015 (in millions)?

**Fig. 1.** Example instance showing the need for matching collectively between the question, tables and linear text. Various colored spans in the question are best matched to correspondingly colored spans in the document, to extract the values (orange borders) needed to answer the question. Our approach to convert the document to a query-modulated graph network is expected to focus on these matches while tuning down other possible spurious matches (not shown). (Color figure online)

is not sufficient to handle situations where fragments of the query must match various sites in text and table elements that are collectively proximate, in some non-trivial sense, to the answer. In Fig. 1, the span `securities loaned` matches a row header and `2017` matches a column header in the lower table. The span `aggregate contractual principal in excess of fair value` matches a row header and `2015` matches a column header in the upper table. Not highlighted are partial matches between query spans and the unstructured text spans in the document (which may be useful or spurious). The spatial relations between these match locations have semantics that help us retrieve values from the two cells with orange borders, and then compute the response, possibly using a calculator tool or by invoking code. In this paper, we focus on the first *retrieval* step.

In this work, we model a document as a sequence of sentences, interspersed with, or connected with, tables with possibly complex layouts. Earlier work [22, 26] provides an idiom for converting standalone tables to graphs. We extend the graph by adding a node for each unstructured sentence, and connecting them up in a chain, passing through or linked to nodes that are placeholders for whole tables and connecting to its headers and cells. Edges in this graph are richly typed, depending on what kind of nodes they connect. The nodes contain text

from table cells or sentences, as the case may be. Another innovation is that the nodes and edges of this graph are featurized via early interaction with the question, or segments of the question inferred by an LLM.

Given a question, the table+text retrieval problem thus reduces to selecting a subset of nodes from the graph induced from tables and associated texts. Given the graph may have thousands to tens of thousands of nodes, this initial filtering or retrieval step must be fast and have high recall. Subsequent steps may refine these candidates down to the answer node(s) via computationally expensive operations such as cross-interaction LLMs.

***Contributions:*** We present a high-recall retrieval/filtration stage for text+table QA with the following salient combination of innovations.

**`Question segmentation`** —We use LLMs' vast pretrained world knowledge to segment the question into portions that target single tables, or a single connection between a pair of tables or sentences.

**`Segment-informed GNN construction`** —The question and its segments are used in conjunction with the uniform graph representation described above, to define a novel hybrid early-interaction graph neural network (GNN).

**`Compact subgraph retrieval`** —The GNN is trained to identify a subset of nodes that collectively cover all segments, and are connected by edges strongly triggered by the question.

The result is TabSegNet, a modular graph filter that achieves high recall of ground-truth answer-containing nodes, as evidenced by extensive experiments over several table+text QA benchmarks.

## 2 Related Work

***End-to-End LLMs for QA:*** Question answering over multi-modal documents including table+text presents a very practical use-case and therefore, has garnered much interest in the retrieval community. A common approach in many of the recent works on this topic has been to use LLMs to solve it as an end-to-end QA task. Typically these approaches [1,30] propose to linearize tables along with the question as inputs to an encoder, followed by a causal decoder to generate the answer. Although the maximum input context length of LLMs has rapidly increased, LLMs cannot yet exploit long contexts reliably [16,17,24].

Therefore, selecting relevant sentences and table slices becomes a key factor in ensuring LLMs efficacy, and yet, there is insufficient scrutiny of this critically important first step. Existing work focuses more on end-to-end QA accuracy than this filtering or retrieval stage, which is fraught with two serious dangers:(1) LLM-based end-to-end QA systems may elicit correct answers from pre-trained knowledge rather than deriving it from the retrieved table(s) and text, thus being vulnerable to make mistakes if the relevant text or tables change or gets updated, which is quite common in enterprise settings, especially with streaming data repository. (2) If retrieved supporting evidence is noisy, the LLM runs a greater risk of hallucinating incorrect responses.

***Table to Graph Representation:*** Given the heterogeneous nature of multimodal context, several recent systems [9,27] have recognized the versatility of graphs for developing a uniform representation for tables and text [23,26]. GTR [23] proposed a way to convert tables (even with nested row/column hierarchies) to graphs with typed edges, added edge-based biases [21] to a transformer network to define a GNN, and showed that table embeddings obtained from pooled node embeddings from this GNN are useful for whole-table retrieval. From this starting point, we make some critical enhancements. (1) We extend the representation to add unstuctured text nodes and connect them to nodes representing table elements. (2) Our GNN is defined as an early-interaction function of the question (and its decomposed segments) as well as the graph, not the graph alone. (3) We support fine-grained node filtering and retrieval, not whole-graph retrieval.

## 3 Baseline (Sub)graph Retrieval Approaches

We list baseline approaches for table+text QA and highlight potential drawbacks to justify TabSegNet, presented in Sect. 4. Some of these baselines will be used in our experiments.

***LLM as Retriever (LLM-R):*** For a fair comparison across retrieval techniques, we pass linearized tables and text, interspersed with unique, opaque[1] IDs (corresponding to graph nodes) into the LLM and prompt it to generate a ranked list of the most relevant IDs [20], which is used to evaluate recall@$K$, MAP, etc. Some fraction of instances overflowed the context capacity, and it took some prompt tweaking to reduce hallucination and elicit enough useful IDs from the LLM.

***Sparse and Dense Single-Vector Retrieval:*** Classic sparse and dense retrieval provide more baselines. For sparse retrieval we use vector space models **TFIDF** and **BM25** [18]. For dense passage retrieval (**DPR**), we encode the question and each node's text into single dense vectors and use the popular DiskANN package [19] with cosine similarity to rank graph nodes. Failure analysis shows that single-vector search is inadequate; evidence collection must happen across multiple table slices and text segments, also informed by graph proximity of matched nodes. TFIDF, BM25 and DPR all score nodes *in competition with* other nodes, failing to capture this collective scoring requirement.

***ColBERT Multi-vector Dense Retrieval:*** [15] avoid premature aggregation of question and node texts into fixed-size vectors, while preserving some contextualization. ColBERT is known to work better than single-vector DPR. Each question word finds its best 'partner' word in the text of a candidate node. Our system uses ColBERT's scoring function, but, used by itself, it may also suffer from not modeling *collective* satisfaction of query words by a candidate subgraph.

[1] I.e., unlikely to collide with meaningful tokens in the LLM's vocabulary.

## 4 Proposed Method: TabSegNet

***Preliminaries:*** We are given a query $\boldsymbol{q}$ which is a sequence of tokens. Along with the query, we are given a document $D$ containing spans of text interleaved with tables $T_1, \ldots, T_N$. The document text is segmented into sentence-sized spans. Table $T_n$ has a caption text. It is composed of cell elements $T_n[i, j]$. Our goal is to retrieve a ranked list of most relevant table elements and sentences.

In the following sections, we describe the essential components of TabSegNet: (1) question segmentation using a tuned LLM, (2) construction of a feature-rich graph from the question segments and the document, and (3) formulation of a GNN-based subgraph selection method.

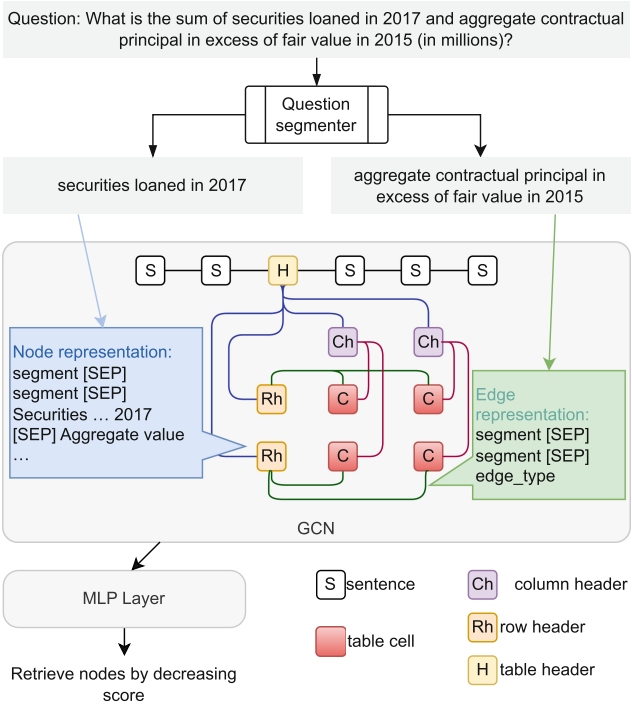



**Fig. 2.** TabSegNet block diagram. A fine-tuned LLM segments the query. Text interspersed with tables are converted into a uniform graph representation. Nodes and edges are featurized based on their edge types and the query segments. Then a graph convolutional network (GCN/GNN) followed by a MLP layer score nodes for retrieval. The parameters in these stages are trained. Some cell-to-cell edges are elided for clarity.

### 4.1 Question Segmentation

Many simplistic retrieval systems score and rank individual nodes against the whole question $\boldsymbol{q}$. As established in Fig. 1, if the question entails combining

information across multiple nodes, it stands to reason that various spans of the question match portions of text in these nodes, but no single node 'covers' the whole question.

```
Instruction: Find segments of the given question which are needed to answer
it. (separate segments by SEP)
Query: What's the difference of Securities America, Inc.-3(4) between 2010
and 2009 for Actual Capital? (in million)
Predicted Segments: Securities America, Inc.-3(4) Actual Capital 2010 (in
millions) SEP Securities America, Inc.-3(4) Actual Capital 2009 (in
millions)
```

**Fig. 3.** Segmenting a question. Black text shows encoder input and teal text shows expected decoder output.

To address this problem, we employ an encoder-decoder LLM to generate *segments* $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_M\}$ from the question. Graph nodes will compete to 'cover' each segment, but, in the end, we will choose a *set* of nodes to cover all segments well. We fine-tune a pre-trained LLM for the task of query segmentation. An example input and output are shown in Fig. 3. For fine-tuning, ideally, we need manually segmented queries, but such data is hard to come by. Instead, we use the available supervision data at the sentence and table cells to derive the desired segments of the question. To finetune, we concatenated the instruction, the question, then the text of each segment separated by **SEP**. Because we fine-tune, no in-context few-shot examples are used. Note that, when the LLM segmenter is deployed, our segments are arbitrary text generated by the LLM, not necessarily contiguous token spans from the question, or text from table or corpus text elements.

Recently, [2] used an LLM to identify, from a natural language question, tables and columns in a relational schema that should appear in a SQL translation. They do not *segment* the question in our sense, or match these segments against unstructured text. The items ranked are whole tables and their columns, not individual cells.

### 4.2 Table+Text Graph Construction

Given a document $D$ containing tables passages, we create a **hybrid graph** $(V, E)$ by extending the paradigm of [22,26], and others. Figure 2 shows the typical topology of the graph obtained by conversion from text and tables. Each table, table element and table caption becomes a node in the graph. Some of the table elements are headers. Table-derived nodes are connected similar to GTR [23] (however, GTR provides only whole-table retrieval and not subsets of table elements and text spans like us). Each sentence in the document becomes a node in the graph. Successive sentence nodes are connected with an edge. Documents

vary with regard to meaningful placement of tables. If the tables are placed deliberately and manually, we connect the table header node to adjacent sentence nodes. Some typesetting tools may migrate tables and figures for improved layout. In such cases, we can detect references from the text to a table, and insert edges in the text chain from those reference sites. This is left for future work. If $(u, v)$ is an edge, $t(u, v) \in [1, T]$ will denote its type index. We list in Fig. 4 some of the edge types we use in our table+text to graph representation. Depending on the application, we can add make every edge bidirected, adding distinct edge types in the two directions.

| | |
|---|---|
| passage to another passage | table caption to previous passage |
| passage to table caption | table caption to next table caption |
| table caption to row header | row header to table caption |
| table caption to column header | column header to table caption |
| table caption to all table cells | table cell to table caption |
| table cell to its row header | table cell to its column header |
| table cell to next cell in same row | table cell to previous cell in same row |
| table cell to next cell in same column | table cell to previous cell in same column |

**Fig. 4.** Sample edge types introduced into the unified graph.

### 4.3 Subgraph Selection via GNN

In a broad departure from earlier table QA methods, we bring together the question, its segments, and the hybrid graph (in a process of early interaction) to define a GNN, which is then used to select a subgraph as retrieved for the question.

`CLS` vectors output by a pretrained BERT [8] model is used to obtain the base node and query representations. Suppose $\text{text}(u)$ is the token sequence associated with node $u \in V$ in the graph. Then we write

$$\boldsymbol{q} = \text{BERT}(\boldsymbol{q})[\texttt{CLS}] \text{ and} \tag{1}$$

$$\boldsymbol{u} = \text{BERT}(\text{text}(u))[\texttt{CLS}]. \tag{2}$$

Similarly, if $\boldsymbol{s}_m$ is a segment, we write

$$\boldsymbol{s}_m = \text{BERT}(\boldsymbol{s}_m)[\texttt{CLS}]. \tag{3}$$

At this point, most GNN-based QA systems would initialize the first layer node embeddings as $\boldsymbol{x}^{(0)}(u) \leftarrow \boldsymbol{u}$ and then iterate

$$\boldsymbol{x}^{(\ell+1)}(u) \leftarrow \text{comb}\left(\boldsymbol{x}^{(\ell)}(u), \sigma\big(\bigoplus_{(u,v)\in E} \boldsymbol{W}\boldsymbol{x}^{(\ell)}(v)\big)\right), \tag{4}$$

where $\bigoplus$ is a symmetric aggregator and $\sigma(\cdot)$ is some activation function. After the last $L$th layer, they would obtain node representations $\boldsymbol{x}^{(L)}(u)$ and compute

the score of node $u$ as the similarity $\text{sim}(\boldsymbol{x}^{(L)}(u), \boldsymbol{q})$. The intuition behind such use of a GNN is that, if nodes $u$ and $v$ collectively cover the question well, their last-layer embeddings will resemble $\boldsymbol{q}$ only if they are nearby in the graph.

***Question-Driven Edge Modulation:*** The above intuition might work if all edges convey relations of comparable importance, which is definitely not our situation. Initial experiments were unimpressive, leading to the insight that although $E$ contains many types of edges, only a few of them are important from the point of view of a given question; the other edges might well not exist.

To implement this intuition, we provide an embedding vector for each edge (relation) type in the form of matrix $\boldsymbol{R} \in \mathbb{R}^{T \times d'}$, with row $\boldsymbol{R}[t(u,v), \star]$ being the base embedding of edge $(u,v) \in E$. The question-dependent importance or 'strength' of edge $(u,v)$ is computed as the scalar value

$$s((u,v)|\boldsymbol{q}) = \boldsymbol{q} \cdot \tanh\left(\boldsymbol{R}[t(u,v), \star]\boldsymbol{A} + \boldsymbol{b}\right) \tag{5}$$

where $\boldsymbol{A} \in \mathbb{R}^{d' \times dim\_in\_channel}, \boldsymbol{b} \in \mathbb{R}^{dim\_in\_channel}$ are global trainable model weights, with $dim\_in\_channel$ being the input channels dimension of GNN

***Question-Informed Node Initialization:*** As with edges, a question-independent node initialization fails to capture early the affinity of the question to various nodes. To rectify this, we modify node initialization to

$$\boldsymbol{x}^{(0)}(u) \leftarrow \tanh\big([\boldsymbol{u}, \boldsymbol{q}]\boldsymbol{C} + \boldsymbol{d}\big), \tag{6}$$

where $[\boldsymbol{u}, \boldsymbol{q}]$ is the concatenation of node and question embeddings, and $\boldsymbol{C}, \boldsymbol{d}$ are more global mode weights to learn.

***Segment-Informed Node Initialization:*** Ideally, we wish to inform node initialization from not only the whole query but also the segment best covering the node. To reflect this, we modify $[\boldsymbol{u}, \boldsymbol{q}]$ to also append the embedding $\boldsymbol{s}^*(u)$ of the segment $\boldsymbol{s}^*(u)$ that best covers node $u$, writing this as $[\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{s}^*(u)]$, and compute

$$\boldsymbol{x}^{(0)}(u) \leftarrow \tanh\big([\boldsymbol{u}, \boldsymbol{q}, \boldsymbol{s}^*(u)]\boldsymbol{C} + \boldsymbol{d}\big), \tag{7}$$

adjusting the shape of $\boldsymbol{C}$ suitably.

***GNN:*** At this point we are ready to launch the modified GNN, with layer updates

$$\boldsymbol{x}^{(\ell+1)}(u) \leftarrow \text{comb}\left(\boldsymbol{x}^{(\ell)}(u), \sigma\big(\bigoplus_{(u,v)\in E} s((u,v)|\boldsymbol{q})\, \boldsymbol{x}^{(\ell)}(v)\big)\right), \tag{8}$$

where $\boldsymbol{W}$ has been replaced with edge importance $s((u,v)|\boldsymbol{q})$. The score of node $u$ is calculated as

$$\text{score}(u|D, \boldsymbol{q}) = \text{MLP}_\theta\left(\boldsymbol{x}^{(L)}(u)\right), \tag{9}$$

where $\text{MLP}_\theta$ is a single layer neural network with parameters $\theta$. We sort the nodes by decreasing $\text{score}(u|D, \boldsymbol{q})$.

***Training Protocol:*** Training instances provide $\boldsymbol{q}$ and $D$, together with the set of 'gold' nodes $\{u_{\oplus}\}$. Let all other nodes form the set $\{u_{\ominus}\} = V \setminus \{u_{\oplus}\}$ of 'worse' nodes. We use a standard pairwise hinge loss [12] to encourage the scores of gold nodes to exceed the scores of worse nodes:

$$\sum_{u_{\oplus}, u_{\ominus}} \text{ReLU}\left(\delta + \text{score}(u_{\ominus}|D, \boldsymbol{q}) - \text{score}(u_{\oplus}|D, \boldsymbol{q})\right), \tag{10}$$

where $\delta$ is a margin hyperparameter. The trainable weights of TabSegNet consist of $\theta, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{C}, \boldsymbol{d}$, which makes for a fairly frugal model with fast training speeds.

### 4.4 Training the Retriever

We use a sentence transformer as the encoder to get initial representations for both node and query representations. For each node in the graph, we extract its text and use encoder to get its representation. Similarly, we use encoder to get representations for the query and its segments. After assigning initial representations for node, query and its segments, the representations are updated via the GCN layer during each iteration of training. Each iteration refines the node's representation by combining it with the representations of its neighboring nodes in the graph. At the end of training we get refined node representations for all the nodes and we score the nodes based on their similarity between node and query representations. The basic idea is that nodes close to each other in the graph and relevant to the query will have similar representations.

***Node Representation*** Each node in the graph is represented by concatenating the node's data embedding with the embeddings of the query and all its segments. This combined representation is then compressed using a linear layer followed by a tanh activation function. This approach allows the node representation to capture interactions with the query and segments, enabling the model to learn the relationship between the query and nodes and score the nodes based on their relevance to the query. To better capture the relevance of the query to different nodes, we modify the initial node embeddings. Instead of using only the node's encoder representation, we also include the query's representation. This helps the GCN to start with an understanding of how each node relates to the query.

***Edge Type and Edge Representation*** Each edge type in the graph is represented as the element-wise dot product of the query embedding and the relation type embedding (similar to a base word embedding matrix). This design allows the edge representation to adapt to the query, learning the relationship between the query and different edge types. For example, for an aggregate query, the model learns to emphasize edges connecting column headers to cells, which are crucial for aggregating information. We assign an embedding to each type of edge and compute its importance using the query. This ensures that only the most relevant edges are emphasized during the GCN's propagation process.

The GNN processes the node representations and aggregates information from their neighbors based on the graph structure. The output from the GCN

is fed into a single Multi-Layer Perceptron (MLP) layer. This MLP layer is responsible for scoring each node, determining its relevance to the query. Training involves using a set of training instances where each instance includes a query, a document, and the set of 'gold' nodes (the most relevant nodes). We train the model to ensure that the scores of these gold nodes are higher than those of other nodes. We use a hinge loss function to achieve this, which encourages the model to maximize the difference in scores between relevant and irrelevant nodes. The model parameters are adjusted during training to improve its performance, leading to an efficient model that trains quickly.

## 5 Experimental Setup

In this section we detail our experimental setup including the benchmarks, baselines, implementation details and finally a thorough analysis of results and discuss key takeaways.

### 5.1 Benchmarks

Although our problem setting is well-motivated, public domain data sets exactly satisfying our requirements are not plentiful. Our requirements are that, to answer a question successfully, the system must collate evidence from a combination of tables and passages from a document having several passages interspersed with tables. Other popular table-QA data sets like HybridQA [5], TaT-QA [31], OTT-QA [3] or AIT-QA [14] do not satisfy all our requirements.

We use the MultiHierTT [28] data set as the latest and among the most challenging data sets in the table+text QA space. We also extend the FinQA [6] data set to fit our requirements. For both data sets, our focus will be on retrieving relevant evidence, rather than inferring the final response. We will make our extended data sets publicly available.

***MultiHierTT:*** The MultiHierTT [29] data set consists of 7830 training instances and 1044 dev instances. Each instance contains (1) a context document consisting of passages interspersed with structured tables with complex layouts, (2) a natural language question that can be answered using the context document, (3) manual annotation of ground-truth evidences (table elements and text spans) required to answer the query, and (4) the ground-truth answer. The context documents contain 70 passages and 4 tables on average. In general, evidences from multiple passages and tables are required to answer the questions. The manual annotations are used in two ways: (1) text from ground-truth evidence and answer are used to fine-tune our LLM segmenter, and (2) ground-truth evidence and answer are mapped to gold nodes $\{u_{\oplus}\}$, used to supervise the GNN (10).

***FinQA:*** The FinQA[2] [6] data set consists of 6251 training instances and 883 dev instances. On average, in this data set, the context for every query contains

[2] Processed data and scripts are available here https://github.com/vishwajeetkumar93/tabsegnet_data_scripts.

23 passages and a single table. To stress the ability of TabSegNet to retrieve the correct table out of competitive choices in a more realistic setting, we add a set of distractor [25] tables to the context. We first tokenize all tables in the data set and prepare a sparse index. Then, for each instance, we tokenize the table containing the ground-truth answer, tokenize it, and look it up in the index using the BM25 similarity score. The three most similar tables are artificially introduced into the context document at randomly selected positions.

### 5.2 Baselines

We provide the configuration details for the baselines as mentioned in Sect. 3 .

**TFIDF:** Scikit implementation with default settings.
**BM25:** Okapi BM25 library with default settings.
**DPR:** Facebook DPR library with default settings in zero-shot mode. Our benchmarks have been publicly available for years, so fine-tuning is unlikely to result in large improvements. Tables are linearized and table element texts are concatenated before passing into DPR's encoder.
**ColBERT:** We used the code and model checkpoint from existing public code repository. Tables are linearized similar to DPR. No fine-tuning is done for the same reason as in DPR.
**Splade:** We also use learned sparse retriever Splade_v2 [10] as a baseline here, reusing the code and model checkpoints publicly available from the authors.

### 5.3 Implementation Details

All experiments ran on a single NVIDIA A100_80GB GPU. Some implementation details of TabSegNet follow. For **query segmentation**, we fine-tuned FLAN-T5-XL [7] using AdamW with learning rate $2{\times}10^{-5}$. For **graph representation**, we used the NetworkX library. To derive initial **node and edge embeddings**, we used pre-trained SentenceTransformers to get a 768 dimensional vector for the `[CLS]` embedding to represent the query and each of the nodes. For **GNN computations**, we used GraphConv, with $L = 1$ layer ($L > 1$ made minimal difference) and max as the aggregator function $\bigoplus$ in Eqn. 8, 120 input and 150 output channels. $\text{MLP}_\theta$ has one layer. Model weights were initialized with Xavier Uniform [11] and were optimized using Adam, setting learning rate to 0.001. Models were trained for 20 epochs on the train set and the best checkpoint was selected based on the lowest loss on the dev set.

**Table 1.** TabSegNet overall retrieval comparison with baselines.

| Model | FinQA | | MultiHierTT | |
|---|---|---|---|---|
| | R@10 | MAP | R@10 | MAP |
| BM25 | 0.610 | 0.47 | 0.472 | 0.35 |
| TFIDF | 0.700 | 0.56 | 0.535 | 0.40 |
| DPR | 0.640 | 0.52 | 0.491 | 0.36 |
| ColBERT | 0.714 | 0.60 | 0.566 | 0.44 |
| SPLADE | 0.732 | **0.61** | 0.591 | 0.46 |
| TabNet_No_Etype | 0.745 | 0.42 | 0.651 | 0.43 |
| TabNet | 0.798 | 0.46 | 0.634 | 0.41 |
| TabSegNet | **0.836** | 0.54 | **0.675** | **0.47** |

### 5.4 Evaluation Metrics

Our data sets identified ground-truth answer text spans and table elements. These are easily mapped to nodes in the graph we build, which we call 'gold' nodes. Any system being evaluated is expected to return a ranked list of nodes. From this ranking, we compute recall@10 (R@10) and mean average precision (MAP), which reduces to mean reciprocal rank (MRR) in case only one gold node exists. (The MultiHierTT leaderboard and FinQA leaderboard are online, but they report end-to-end answer accuracy, which includes arithmetic abilities beyond retrieval, which is outside our scope. Most chart-toppers focus, and are evaluated, on the correctness of the *program* computing the answer, not *retrieval*.)

## 6 Results and Analysis

We first describe key research questions that we want to address —**RQ1:** Does TabSegNet improve retrieval accuracy for complex multimodal QA where a query needs multiple tablular and text segments? We also want to investigate the effect of key components in TabSegNet design. **RQ2:** How important is the segmentation step for TabSegNet performance? **RQ3:** How important are having edge type distinctions?

### 6.1 TabSegNet vs. baselines

Table 1 shows that, in terms of R@10, TabSegNet is better than all other retriever baselines. Among existing baselines, ColBERT does better than DPR or (untrained) sparse retrievers, which is expected given its ability to allow multi-vector interaction between query and document. The difference in performance between TabSegNet and sparse retrievers is even higher, given that the sparse retrievers rely on keyword matches irrespective of their table/text positions.

SPLADE is the best baseline method. It relies on learned keyword expansions which acts like semantic synonyms of important terms in the document. Moreover, SPLADE and ColBERT also achieves better MAP scores because they work at token level matches for relevance scoring. However, for downstream applications using LLMs over the retrieved context, a better recall becomes the necessary criterion where TabSegNet improves on both ColBERT and SPLADE in R@10, supporting RQ1. Both TabSegNet and TabNet gets better recall than baselines. This validates the need for adopting a graph structure unifying tabular and text data, plus graph-enabled learning to score nodes.

### 6.2 Benefit of Segmentation

We also see in Table 1 that, without segmentation, there is a considerable drop in both recall and MAP. Question segmentation helps in finding focused localized matches and also differentiates between a compact sub-graph covering important question words, versus a set of unrelated nodes that match parts of the query. This result answers RQ2 in the affirmative, i.e., segmentation is indeed needed to boost accuracy of table+text retrieval. Summarizing, TabSegNet with question segmentation, graph representation, and learning to score nodes via GNN training can effectively induce the much-needed notion of compact subgraph retrieval.

### 6.3 Benefit from Edge Types and Weights

Table 1 further shows that if we remove distinct edge types, recall and MAP significantly decreases for FinQA, although not so much for MultiHierTT. With designated, informative edge types, the GNN can better learn associations between the question and valuable edges, as well as differentiate between promising versus unrelated neighbourhood nodes. Such learning translates to better evidence retrieval during inference. FinQA numbers answer our third research question RQ3 in the affirmative, i.e. edge types play a vital role.

**Table 2.** TabSegNet

| Model | FinQA | | MultiHierTT | |
|---|---|---|---|---|
| | Recall@10 | MAP | Recall@10 | MAP |
| TabSegNet_large | 0.836 | 0.542 | 0.675 | 0.468 |
| TabSegNet_base | 0.790 | 0.462 | 0.648 | 0.440 |
| TabNet_large | 0.798 | 0.462 | 0.634 | 0.407 |
| TabNet_base | 0.731 | 0.396 | 0.542 | 0.347 |

### 6.4 Ablation on GNN Model Size

Results in Table 2 explores the effect of changing the number of GNN parameters. Here 'large' models have GNN input width of 128 as opposed to 120 in 'base' models and the output node representation with is 256 in 'large' models, as opposed to 150 in 'base'. Furthermore, the 'large' model uses an edge embedding size of 128 whereas it is 10 for 'base'.

As we can see in Table 2 the change in representation sizes in GCN nodes and edge embedding impacts recall and MAP in predictable ways. The 'large' configuration has more representation power and is able to learn better associations between the question and relevant graph nodes and edges, resulting in better performance. Also, the effects of having a larger variation of GNN params impacts both TabSegNet and TabNet, once again validating the fact that a robust GNN helps the overall system—with or without segmentation.

### 6.5 Discussion and Anecdotes

***Segmenter Errors.*** Unlike MultiHierTT, which gold evidence nodes accompanying each instance, for FinQA we derive those using gold program annotations and hence call them silver segments. We do a quality check of these silver segments and perform necessary cleaning to keep FinQA data relevant for our use-case. We see two key failure cases - pertaining to segmenter and the silver segments.

**Incorrect labeled data** —Careful inspection revealed spurious program annotations in FinQA, which renders the silver segments spurious and uncorrelated to the question. Such error cases can not be recovered from any segmentation method. One example is shown:

Question
```
what was jpmorgan chase & co's common equity tier 1 ( cet1 ) ratio
in 2008?
```

Silver segments
```
2008 total tier 1 capital ( a ) [SEP] 2008 risk-weighted assets
```

Inferred segments
```
2008 common equity tier 1 ( cet1 ) [SEP]
2007 common equity tier 1 ( cet1 )
```

Although the question specifically refers to `common equity tier 1 (cet1)`, the gold program annotation refers to a unrelated (incorrect) table with column header `total tier 1 capital`. This leads to the silver segments from this annotation include tokens such as `total`, which is not in the original query itself. Our trained segmenter does a satisfactory job of predicting `2008 common equity tier 1 ( cet1 )` and `2007 common equity tier 1 ( cet1 )` as possible query segments which seem plausible. We purposefully exclude such misleading instances to report on a sanitized FinQA.

**Only textual evidence** —There are many instances in FinQA where the answer is directly contained in a single text span. Such instances are not suitable for our use-case where the aim is to evaluate systems on their ability to discover *connections* between multiple tables and text spans for accurate evidence retrieval. In fact, for such examples, the silver segments are empty, again misleading the training of our LLM-based segmenter. There are a total of 235 instances where the derived silver segment is empty. We remove those from our evaluation.

## 7 Conclusion

We presented TabSegNet, a retrieval and filtering stage for table+text QA. In TabSegNet, we use a uniform graph representation for table+text, upon which we define a GNN whose node and edge weights are carefully informed by question segments, which are obtained from a fine-tuned LLM. Experiments show that TabSegNet has superior recall of ground-truth text spans and table elements compared to recent competitive dense retrieval methods.

## References

1. Andrejczuk, E., Eisenschlos, J., Piccinno, F., Krichene, S., Altun, Y.: Table-to-text generation and pre-training with TabT5. In: Goldberg, Y., Kozareva, Z., Zhang, Y. (eds.) Findings of the Association for Computational Linguistics: EMNLP 2022, pp. 6758–6766. Association for Computational Linguistics, Abu Dhabi (2022). https://doi.org/10.18653/v1/2022.findings-emnlp.503. https://aclanthology.org/2022.findings-emnlp.503
2. Chen, P.B., Zhang, Y., Roth, D.: Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In: ACL (2024). https://arxiv.org/abs/2404.09889
3. Chen, W., Chang, M.W., Schlinger, E., Wang, W., Cohen, W.W.: Open question answering over tables and text (2021)
4. Chen, W., et al.: Tabfact: a large-scale dataset for table-based fact verification. CoRR arxiv:1909.02164 (2019)
5. Chen, W., Zha, H., Chen, Z., Xiong, W., Wang, H., Wang, W.Y.: HybridQA: a dataset of multi-hop question answering over tabular and textual data. In: Cohn, T., He, Y., Liu, Y. (eds.) Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1026–1036. Association for Computational Linguistics, Online (2020). https://doi.org/10.18653/v1/2020.findings-emnlp.91. https://aclanthology.org/2020.findings-emnlp.91
6. Chen, Z., et al.: Finqa: a dataset of numerical reasoning over financial data (2022)
7. Chung, H.W., et al.: Scaling instruction-finetuned language models (2022)
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL Conference (2019). https://www.aclweb.org/anthology/N19-1423.pdf
9. Du, L., et al.: TabularNet: a neural network architecture for understanding semantic structures of tabular data. In: Proceedings of the 27th ACM SIGKDD Conference. KDD 2021. ACM (2021). https://doi.org/10.1145/3447548.3467228

10. Formal, T., Piwowarski, B., Clinchant, S.: Splade: sparse lexical and expansion model for first stage ranking. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 2288–2292 (2021)
11. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 9, pp. 249–256. PMLR, Chia Laguna Resort (2010). https://proceedings.mlr.press/v9/glorot10a.html
12. Joachims, T.: Optimizing search engines using clickthrough data. In: SIGKDD Conference, pp. 133–142. ACM (2002). http://www.cs.cornell.edu/People/tj/publications/joachims_02c.pdf
13. Katsis, Y., et al.: AIT-QA: Question answering dataset over complex tables in the airline industry. In: Loukina, A., Gangadharaiah, R., Min, B. (eds.) Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track, pp. 305–314. Association for Computational Linguistics, Hybrid: Seattle, Washington + Online (2022). https://doi.org/10.18653/v1/2022.naacl-industry.34. https://aclanthology.org/2022.naacl-industry.34
14. Katsis, Y., et al.: AIT-QA: question answering dataset over complex tables in the airline industry. In: NAACL Conference (2022). https://openreview.net/forum?id=VkY8WURpggN
15. Khattab, O., Zaharia, M.: Colbert: efficient and effective passage search via contextualized late interaction over bert. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 39–48 (2020)
16. Li, T., Zhang, G., Do, Q.D., Yue, X., Chen, W.: Long-context llms struggle with long in-context learning (2024)
17. Liu, N.F., et al.: Lost in the middle: how language models use long contexts. Trans. Assoc. Comput. Linguist. **12**, 157–173 (2024). https://doi.org/10.1162/tacl_a_00638
18. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge (1999). https://nlp.stanford.edu/fsnlp/
19. Simhadri, H.V., et al.: DiskANN: graph-structured indices for scalable, fast, fresh and filtered approximate nearest neighbor search (2023). https://github.com/Microsoft/DiskANN
20. Tay, Y., et al.: Transformer memory as a differentiable search index. In: Oh, A.H., Agarwal, A., Belgrave, D., Cho, K. (eds.) Advances in Neural Information Processing Systems (2022). https://openreview.net/forum?id=Vu-B0clPfq
21. Wang, B., Shin, R., Liu, X., Polozov, O., Richardson, M.: Rat-sql: relation-aware schema encoding and linking for text-to-sql parsers (2021)
22. Wang, F., Sun, K., Chen, M., Pujara, J., Szekely, P.: Retrieving complex tables with multi-granular graph representation learning. In: SIGIR Conference, pp. 1472–1482 (2021). https://dl.acm.org/doi/pdf/10.1145/3404835.3462909
23. Wang, F., Sun, K., Chen, M., Pujara, J., Szekely, P.A.: Retrieving complex tables with multi-granular graph representation learning. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (2021). https://api.semanticscholar.org/CorpusID:233739808
24. Xiang, Y., Yan, H., Gui, L., He, Y.: Addressing order sensitivity of in-context demonstration examples in causal language models. ACL Findings (2024). https://arxiv.org/abs/2402.15637

25. Xiao, Y., et al.: Dynamically fused graph network for multi-hop reasoning. arXiv preprint arXiv:1905.06933 (2019)
26. Zayats, V., Toutanova, K., Ostendorf, M.: Representations for question answering from documents with tables and text. In: EACL Conference (2021). https://www.aclweb.org/anthology/2021.eacl-main.253.pdf
27. Zhang, X., Shou, L., Pei, J., Gong, M., Wen, L., Jiang, D.: A graph representation of semi-structured data for web question answering. In: COLING (2020). https://arxiv.org/abs/2010.06801
28. Zhao, Y., Li, Y., Li, C., Zhang, R.: Multihiertt: numerical reasoning over multi hierarchical tabular and textual data. arXiv preprint arXiv:2206.01347 (2022)
29. Zhao, Y., Li, Y., Li, C., Zhang, R.: MultiHiertt: Numerical reasoning over multi hierarchical tabular and textual data. In: Muresan, S., Nakov, P., Villavicencio, A. (eds.) Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, vol. 1: Long Papers, pp. 6588–6600. Association for Computational Linguistics, Dublin (20). https://doi.org/10.18653/v1/2022.acl-long.454. https://aclanthology.org/2022.acl-long.454
30. Zhou, Y., Bao, J., Duan, C., Wu, Y., He, X., Zhao, T.: UniRPG: unified discrete reasoning over table and text as program generation. In: Goldberg, Y., Kozareva, Z., Zhang, Y. (eds.) Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pp. 7494–7507. Association for Computational Linguistics, Abu Dhabi (2022). https://doi.org/10.18653/v1/2022.emnlp-main.508. https://aclanthology.org/2022.emnlp-main.508
31. Zhu, F., et al.: TAT-QA: a question answering benchmark on a hybrid of tabular and textual content in finance. In: Zong, C., Xia, F., Li, W., Navigli, R. (eds.) Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, vol. 1: Long Papers, pp. 3277–3287. Association for Computational Linguistics, Online (2021). https://doi.org/10.18653/v1/2021.acl-long.254. https://aclanthology.org/2021.acl-long.254