HGT: Leveraging Heterogeneous Graph-enhanced Large Language Models for Few-shot Complex Table Understanding

Anonymous ACL submission

Abstract

Table understanding (TU) has achieved promising advancements, but it faces the challenges of the scarcity of manually labeled tables and the presence of complex table structures. To address these challenges, we propose HGT, a framework with a heterogeneous graph (HG)-enhanced large language model (LLM) to tackle few-shot TU tasks. It leverages the LLM by aligning the table semantics with the LLM's parametric knowledge through soft prompts and instruction turning and deals with complex tables by a multi-task pre-training scheme involving three novel multi-granularity self-supervised HG pre-training objectives. We empirically demonstrate the effectiveness of HGT, showing that it outperforms the SOTA for few-shot complex TU on several benchmarks.

1 Introduction

Table Understanding (TU) seeks to learn informative embeddings of tables containing inherently tabular semantics, often formatted in ways not easily interpretable by machines as shown in Fig. 1. This endeavor enhances machine performance across a range of table-related tasks, such as Table QA (Herzig et al., 2020) and Cell Type Classification (Ghasemi-Gol et al., 2019) and Table Type Classification (Wang et al., 2020b).

Yet, in real-world scenarios, TU faces the challenges of **a lack of sufficient human annotations** and the presence of **complex table structures**, which diminishes the effectiveness and applicability of existing frameworks. As for the first challenge, the data-hungry nature of existing frameworks results in diminished performance in few-shot TU scenarios, where only several samples are annotated. Although some studies (Yin et al., 2020; Herzig et al., 2020) have adopted pre-training with encoder-only architectures (Devlin et al., 2019) to alleviate the annotation burden, these solutions still require considerable amounts of labeled data for task-specific fine-tuning. As for the second challenge, while current frameworks attempt to capture structural information in tables through the use of position embeddings (Wang et al., 2020b) or by modeling tables as graphs (Wang et al., 2021), these methods are effective for simple tables but wane with complex tables. This shortfall arises because the cell-to-cell relationships in complex tables are more intricate than those encountered in simple tables as shown in Fig. 1.

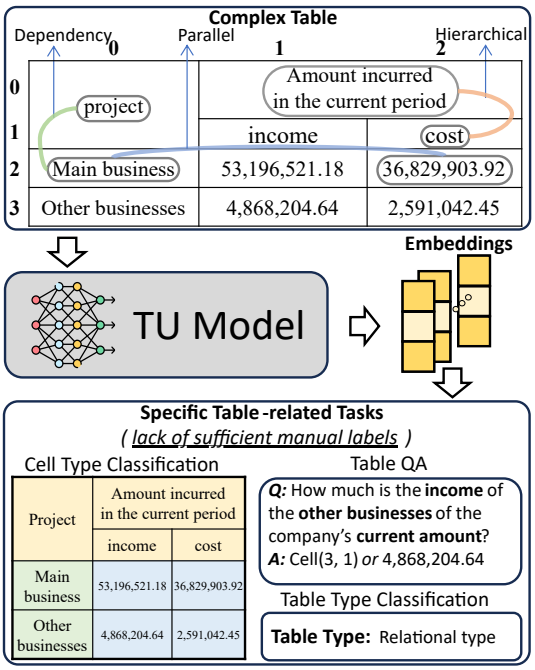


Figure 1: Few-shot complex Table Understanding. Complex tables contain intricate cell-to-cell relationships including dependency, hierarchical, and parallel ones.

Fortunately, recent advancements have introduced techniques that can be integrated into the TU framework to tackle the challenges: 1) Multimodal Large Language Models (LLMs) have shown remarkable effectiveness in managing few-shot tasks involving data from other modalities, such as vision (Liu et al., 2023) and time series (Jin et al., 2023a). This is achieved through instruction tuning

and soft prompts, which seek to align the semantic spaces of the other modality encoder with the LLM. 2) Self-supervised heterogeneous graph (HG) pre-training (Ma et al., 2023; Yang et al., 2022) empowers models to navigate multifaceted relationships within data from substantial volumes of unlabeled data.

In this work, we propose HGT, an HG-enhanced LLM framework for the few-shot complex TU. To enable HGT to model the intricate relationships within complex tables, HGT begins with modeling the tabular data with the HG and then processing it through a tabular graph encoder to generate vectors imbued with structural information. Additionally, to ensure the LLM achieves comparable performance in table tasks to its performance in natural language (NL) tasks, especially in few-shot scenarios, we align the representation spaces of the tabular graph encoder and the LLM via instruction tuning. Specifically, we incorporate table vectors containing structural information as soft prompts within the LLM inputs and innovatively design three multi-granularity self-supervised tasks tailored for both tables and LLMs for pre-training. After pre-training specific parameters of HGT via these self-supervised tasks, the HGT capitalizes on the LLM's exceptional generalization capability to adapt to downstream tasks with minimal data samples. To validate the performance of HGT in few-shot complex TU, we conduct extensive comparative experiments with existing powerful baselines on eight publicly available datasets for three specific table-related tasks in few-shot scenarios. The experimental results show that HGT exceeds the current SOTA for few-shot complex TU across multiple benchmark datasets.

In summary, our main contributions are:

- To the best of our knowledge, we are the first to propose a framework to align table modality with NLs to empower the LLM to perform on tables with the same few-shot proficiency as it does with NLs.
- To improve the framework's ability to complex TU, we propose a refined way to convert tables into HGs and design three novel multi-granularity self-supervised HG pre-training tasks tailored for tabular data and LLMs.
- We conduct extensive experiments on eight publicly available datasets, and the experimental results show that HGT exceeds the current SOTA for few-shot complex TU across multiple benchmark datasets.

2 Related Work

2.1 Graph-based Table Understanding

Many studies (Du et al., 2021; Jin et al., 2023b) have tried to convert tables into graphs and utilized graph encoders to capture tables' inherent topological information. These frameworks include employing homogeneous graphs or utilizing a basic node-linking strategy that connects cells exclusively to their adjacent counterparts. Consequently, such frameworks underperform when dealing with complex table structures.

2.2 LLM-based Table Understanding

Following the remarkable success of LLMs in NL tasks, some efforts (Zhang et al., 2023; Chen, 2022) have extended their application to table-related tasks. Although these frameworks leverage the exceptional generalization capacity of LLMs to achieve SOTA performance in some few-shot tasks, they resort to simply converting the table into a row-by-row NL format for input into LLMs. This process leads to a loss of the tables' intrinsic topological information.

2.3 Query Statement-based Table QA

Several studies (Cao et al., 2023; Cheng et al., 2022) have adapted semantic parsing techniques, traditionally applied to database tables, to general Table QA tasks. This involves transforming the table into a format interpretable by query languages, such as SQL, and subsequently utilizing Codex to generate a query statement to retrieve the answer. This methodology represents the current SOTA in Table QA. However, its architectural limitations restrict its applicability to other table-related tasks.

2.4 Bert-like Encoder-only Table Understanding

Since the rise in popularity of pre-training models like BERT (Devlin et al., 2019), there has been considerable effort devoted to designing specialized encoding methods for tabular data and unique pre-text objectives for pre-training (Herzig et al., 2020; Cheng et al., 2021a; Jia et al., 2023). Despite their utilization of self-supervised training, these methods still require a substantial amount of labeled data during fine-tuning for downstream tasks. Additionally, while they incorporate positional embedding into serialized tabular data, they do not effectively capture topological information.

3 Task Definition

Given a table $T = \{c_{i,j} | 0 \leq i < N, 0 \leq j < M\}$ where N is the number of rows, M is the number of columns, and $c_{i,j}$ is the cell located in the i^{th} row and j^{th} column. Merged cells, characterized by a row span or column span greater than 1, are prevalent in complex tables. For instance, the cell labeled "project" in the top-left corner of the table shown in Fig. 3 has a row span of 2. We assign the coordinates of such merged cells based on the position of the top-left cell before merging. Hence, the coordinate of the "project" cell is designated as (0,0).

Sub-TU tasks for Evaluation

Cell Type Classification (CTC) involves identifying the type y_c of each cell $c_{i,j}$ within a table T, where y_c can belong to a basic taxonomy $Y_c = \{header\ cell, data\ cell\}$ or a more complex one, varying across datasets.

Table Type Classification (TTC) is a table-level categorization task that requires models to classify the table T according to a specific taxonomy Y_t.

Table QA demands that the model produce an answer y_a in response to a natural language question q, with table T serving as the reference for deriving the answer y_a.

4 METHODOLOGY

This section explains the process in three phases: tabular HG construction and two tuning stages. An overview of HGT is shown in Fig. 2.

4.1 Tabular Heterogeneous Graph Construction

Given that HGs are more proficient at capturing diverse relationships compared to homogeneous graphs, we employ HGs in conjunction with a refined method that considers the semantic roles of cells to effectively model the structure of complex tables. The subsequent subsections detail the process of converting tabular data into HGs, including the creation of nodes and the heuristic rules for establishing edges between nodes. The process of creating a tabular HG is shown in Fig. 3.

4.1.1 Tabular Node Construction

Four Tabular Node Types. TABLE, ROW, Header CELL, and Data CELL nodes are denoted as green, red, yellow, and blue nodes in Fig. 3. 1) The TABLE node represents the content described by the table, facilitating the table-level tasks. 2) The ROW node signifies the information contained within a row, aiding in the prediction of the row's type during self-supervised training. 3) The Header CELL node denotes cells located in the header row, identifying column schemas or categories. 4) The Data CELL node represents cells in data rows, meaning the actual data entries of the table.

Creating Tabular Nodes. First, CELL nodes are created for each cell in the table, with each node denoted as $c_{i,j}$, where i and j represent the cell's coordinates in the original table. If a CELL node is located in the table's Top Header Row or is part of a merged cell spanning the entire table width, it is viewed as a Header Cell; otherwise, it is identified as a Data Cell. The method for determining the Top Header Row Number of a table is elaborated in Appendix B. Subsequently, an equivalent number of ROW nodes is created to correspond with the number of rows in the table, along with one TABLE node.

Initializing Tabular Node Embeddings. An initialization vector is required for each node in the HG. For Header and Data CELL nodes, we employ the output of Sentence-BERT (Reimers and Gurevych, 2019) applied to the text within each cell to obtain their initialization vectors. In the case of ROW nodes r_i, we initialize a vector by concatenating the text from the cells in the i^{th} row and inputting this concatenated text into the Sentence-BERT. TABLE nodes serve to represent the table's content itself. In a human's view, a table's content can be inferred by examining the cells in the headers. Consequently, we opt to concatenate the text from the cells in the headers and to obtain the embedding.

4.1.2 Adding Edges

To enhance the machine's understanding of the table's semantics, we analyze complex table data and develop the following heuristic method to link nodes: 1) A Table node, representing the whole table, should be linked to all CELL nodes to encapsulate the global semantics. 2) A Row node derives information from cells within the same row, so the Row node r_i should be linked to each Cell node c_{i*}. 3) A strong correlation exists between the semantics of data cells and their corresponding header cells within the same column. 4) Data cells located in the same column exhibit a stronger relational bond compared to those in different columns. Consequently, adjacent data cells within a column are interconnected. 5) The interpretation of rela-

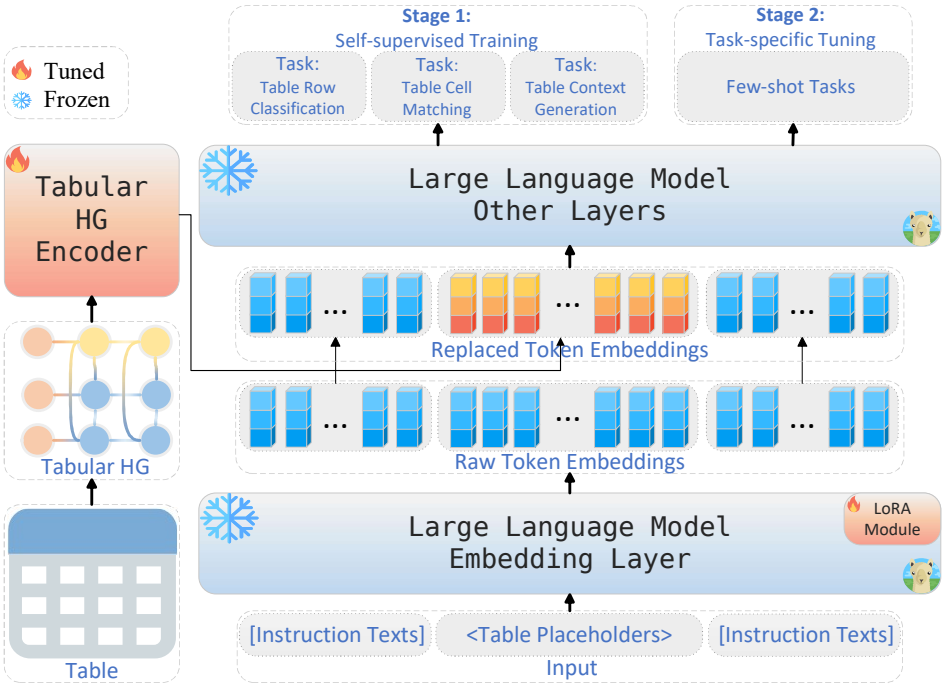


Figure 2: An overview of HGT framework. HGT processes <table, instruction> as an input. First, the table is converted into an HG and processed by a Tabular HG encoder to generate vectors for each tabular node, while the LLM transforms instruction texts into initial token embeddings. Subsequently, the HG encoder's outputs serve as soft prompts for the LLM, enabling the replacement of placeholder embeddings with actual tabular node vectors. The modified embedding sequence is then processed by the remaining LLM layers. Throughout both Stage 1 and Stage 2, only the weights of red components are tuned.

tionships between data cells across columns necessitates the semantic understanding of their respective header cells. For instance, as demonstrated in Fig. 3, determining the relationship between cell $c_{2,0}$ and cell $c_{2,1}$ requires an examination of their header cells. This examination reveals that "53,196,521.18" represents the income from the "main business" project.

Edge types are categorized based on the nodes they link, as follows: Table-Header, Table-Data, Header-Row, Data-Row, Header-Data, Data-Data, and Header-Header edges.

4.2 Stage 1: Self-supervised Instruction Tuning

Illustrated in Fig. 2, HGT utilizes the tabular HG encoder's output as soft prompts (Li and Liang, 2021), which form part of the LLM input. The weights of both modules are tuned through self-supervised instruction tuning to align the vector representation spaces of the two modules. This subsection provides a detailed description of the training process, which involves three granularity self-supervised tasks.

4.2.1 Tabular HG Encoder

Following the conversion of the table into a tabular HG, HGT introduces an RGNN (Wang et al., 2020a) as the encoder for tabular HG data. The encoder takes a tabular HG as input and generates vector representations for the tabular nodes as output. The RGNN employs a message-passing mechanism to collect semantic and topological information from neighboring cell nodes, considering various edge types. Typically, the RGNN comprises multiple graph aggregation layers, with each layer using the following aggregation formula for each cell node v:

$$h_v^{(l+1)} = \sigma\left(\sum_{e_t \in E_t} \sum_{u \in N_v^{e_t}} \frac{1}{|N_v^{e_t}|} W_{e_t}^{(l)} h_u^{(l)}\right), \quad (1)$$

where l is the l^{th} layer, $N_v^{e_r}$ means all neighbor cell nodes of v, E_r is the set of edge type, σ is an activation function and $W_{e_t}^{(l)}$ is trainable weights. The last hidden representation is used as part of the input for subsequent modules in the framework.

	0	1	2
0	project	Amount incurred in the current period	
1		income	cost
2	Main business	53,196,521.18	36,829,903.92
3	Other businesses	4,868,204.64	2,591,042.45

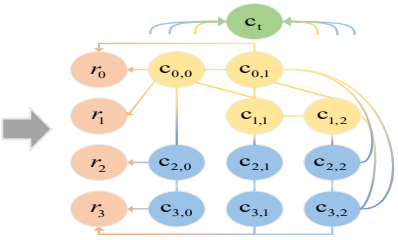


Figure 3: Converting a table into a heterogeneous graph. Cell nodes of varying types are depicted using distinct colors: Table nodes in green, Row nodes in red, Data Cell nodes in blue, and Header Cell nodes in yellow. Similarly, edges are differentiated by color to represent various types. Lines without arrows indicate bidirectional edges. For the sake of clarity, some edges have been omitted.

4.2.2 Multi-granularity Self-supervised Instruction Tasks

Drawing inspiration from the methodologies presented in LLaVA (Liu et al., 2023), Time-LLM(Jin et al., 2023a), and GraphGPT (Tang et al., 2023), we develop a method to effectively align the vector spaces of two distinct modalities of data: tables and NLs. We introduce the soft prompts technique as a bridge between the two encoders and focus on fine-tuning only the tabular graph encoder and the LoRA module within the LLM. This process, designed to be lightweight, enables the LLM to grasp the topological information of tables through semantic instruction learning.

Instruction learning (Wang et al., 2022; Ouyang et al., 2022a), a technique that merges fine-tuning with prompt learning, significantly enhances the generalization capabilities of LLMs. Therefore, before the model-tuning process, we pre-format the self-supervised training data into an instruction-based QA format. Examples of training data for the three tasks—Table Row Classification, Table Cell Matching, and Table Context Generation—are illustrated in Fig. 4.

To align the LLM with the HG encoder, the HGT incorporates additional tokens into the vocabulary: <tabular_node>, <table_start>, and <table_end>. The token <tabular_node> serves as a placeholder for table tasks within the instruction text, allowing for the substitution with actual table node vectors post-processing by the LLM's Embedding layer. The quantity of these placeholders matches the number of table nodes relevant to the current task. For instance, since there are 10 cell tokens in Fig. 4, the <tabular_node> is replicated 10 times. The tokens <table_start> and <table_end> signify the beginning and ending delimiters of the table placeholders.

The forward propagation in HGT begins with the input of a <Table, Instruction> pair, where the instruction text passes through the LLM's Embedding layer, and the table is transformed into a Tabular HG before being processed by the RGNN. The LLM Embedding layer assigns embeddings to each token, creating a sequence, while the RGNN's output provides aggregated vector representations for each table node. Subsequently, HGT replaces the embeddings corresponding to the table placeholders within the sequence with the actual node vectors. This adjusted embedding sequence is then fed into the LLM's remaining layers.

To enhance the HGT understanding of tables, we innovatively devise three tasks tailored for the graph-enhanced LLM, each varying in granularity: Table Row Classification, Table Cell Matching, and Table Context Generation. The complexity of these tasks increases progressively, aiming to elevate the model's capability to comprehend semantic information across different levels. This tiered method is designed to systematically broaden the model's semantic awareness. Below is the detailed descriptions of each task.

Table Row Classification (TRC). The objective of this task is to train the model to accurately identify the category of each table row. Utilizing the vector representations of Row nodes provided by the HG encoder, the model discerns whether a node corresponds to a Header Row or a Data Row. The categorization of Row nodes is automated by an algorithm elaborated in Appendix. B, eliminating the need for manual intervention and thus qualifying as a self-supervised training process. In the dataset we utilize, the labeling accuracy of the algorithm reaches 97.6%. This task facilitates the model's initial grasp of coarse-grained information about the structure of table rows.

Table Cell Matching (TCM). The task involves supplying the model with vector representations

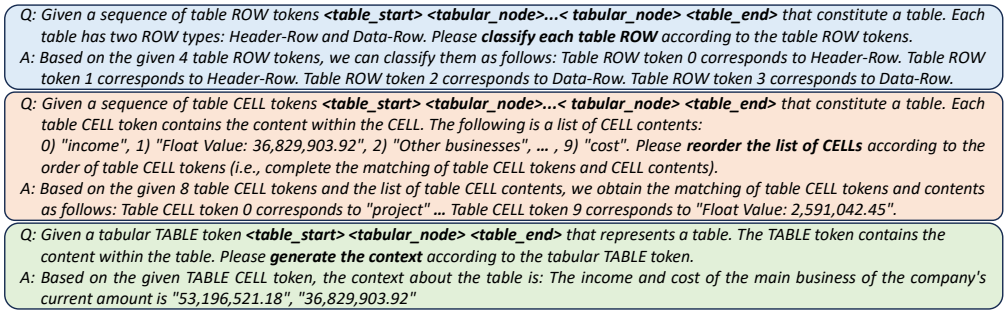


Figure 4: Examples of three self-supervised instruction tuning datasets, each tailored for distinct tasks: Table Row Classification, Table Cell Matching, and Table Context Generation.

of each cell node alongside a list of shuffled cell contents. The model's objective is to correctly pair each cell node vector with its corresponding original text. For instance, referring to the table depicted in Fig. 3, the model needs to align the node vector of $c_{1,1}$ with the string "income" within the list correctly. This training task enables the model to discern the semantic information encapsulated within the cell contents based on the graph node vectors. Essentially, it aligns the semantic space of the HG encoder and the LLM.

Table Context Generation (TCG). The objective of this task is to enable the model to generate context surrounding table data, utilizing the vector representations from the Table node. This task facilitates the model's learning of the table's global semantic information, proving beneficial for tasks requiring a comprehensive understanding of the table as an entity.

Examples of data sets in instruction format for the three tasks are shown in Fig. 4.

4.3 Task-specific Instruction Tuning

Following the completion of the self-supervised task in Stage 1, the HGT has successfully aligned the representation spaces of the HG encoder and the LLM. This alignment, combined with the intrinsic capabilities of the LLM, empowers HGT to more effectively comprehend the topological nuances of complex tables. Consequently, when applied to a specific downstream task, HGT requires only a small number of training samples to quickly grasp the expected answer format for the current task and reorganize its pre-existing knowledge into the suitable output format. When fine-tuning in Stage 2, the model adopts the same weight-tuning strategy as in Stage 1, with all parameters frozen except for those of the HG Encoder.

5 Experiments

5.1 Datasets

To validate the effectiveness of our model, we selected a variety of datasets that are both widely studied and easy to parse, including *TURL* (Deng et al., 2020), *WCC* (Ghasemi-Gol and Szekely, 2018), *IM-TQA* (Zheng et al., 2023), *HiTab* (Cheng et al., 2021b), *WTQ (Flatten)* (Pasupat and Liang, 2015), *WTQ (Raw)* (Pasupat and Liang, 2015), each pertinent to CTC, TTC, or Table QA, respectively.

Statistics for these datasets are presented in Table 1, illustrating variations in the types of annotations, the primary domains covered, and the proportion of complex tables. Given the focus on few-shot TU scenarios, we only list the number of tables within the test sets. Comprehensive details of the dataset annotations and pre-processing are available in Appendix A.

5.2 Baselines

We compare HGT with eight strong baselines to verify its effectiveness. The eight baselines can be categorized into four groups according to their frameworks. ForTap(Cheng et al., 2021a) and GetPt(Jia et al., 2023) emulate BERT, devising specific pretext tasks tailored for tabular data to pre-train the Transformer encoder(Devlin et al., 2019). TabularNet (Du et al., 2021) and TabPrompt(Jin et al., 2023b) incorporate a graph encoder. Binder(Cheng et al., 2022) and AugCodex(Cao et al., 2023) rely on Codex to generate query statements to retrieve the answer. TableLlama(Zhang et al., 2023) and GPT-3.5(Ouyang et al., 2022b) are both LLMs renowned for their exceptional performance in few-shot scenarios.

5.3 Implementation Details

General Setup Across All Tasks. We employ Vicuna-7B-v1.5(Chiang et al., 2023) as the base

Datasets	Annotation Type			Table Info			
	CTC	TTC	Table QA	# Test Tables	% Complex Tables	# QA pairs	Main Domains
IM-TQA	✓	✓	✓	153	47.71	627	Manufacturing
WCC		✓		371	–	–	General
HiTab	✓		✓	3597	92.88	1584	Crime, Health
WTQ (Flatten)			✓	2108	0.00	4344	General
WTQ (Raw)			✓	2108	14.80	4344	General

Table 1: Dataset Statistics. "✓" indicates the type of annotation in the dataset that has this task in it. "# Test Tables", "% Complex Tables" and "# QA pairs" columns show the number of tables in the test set, the percentage of complex tables and the numbers of QA pairs, respectively.

Model Type	Models	CTC (Macro–F1)		TTC (Macro–F1)		Table QA (Accuracy)			
		HiTab	IM–TQA	WCC	IM–TQA	WTQ (Flat)	WTQ (Raw)	HiTab	IM–TQA
Bert-like Encoder-based	ForTap	55.74	46.32	45.13	52.34	26.31	25.48	23.80	28.07
	GetPt	57.45	48.61	47.53	55.42	22.63	22.33	22.29	21.05
Graph-based	TabularNet	53.11	44.53	44.97	55.32	18.12	17.52	21.34	18.98
	TabPrompt	64.44	45.75	49.24	53.32	16.62	13.72	19.19	13.08
Query Statement-based	Binder	–	–	–	–	**50.09**	42.63	38.07	39.07
	Aug-Codex	–	–	–	–	41.39	38.42	53.09	43.54
LLM-based	TableLlama	58.58	52.67	46.31	54.69	37.85	33.32	–	47.05
	GPT-3.5	62.46	55.63	52.32	59.10	39.25	37.51	42.05	51.52
HGT		**66.24**	**60.11**	**56.10**	**59.58**	45.00	**43.11**	**54.61**	**52.95**

Table 2: Overall evaluation results on three TU tasks with best **bolded** and runner-up underlined. '–' indicates that the current framework cannot handle the current task due to some limitations. [1]

model for LLM and a 2-layer RGAT(Wang et al., 2020a) as our tabular HG encoder whose hidden dimension is set as 1048. The initial vectors of nodes in HG are obtained by Sentence-BERT (Reimers and Gurevych, 2019), whose dimension is 768. We integrate a LoRA(Hu et al., 2021) module to the embedding layer of the LLM. At any tuning stage, HGT exclusively tunes the parameters of the RGAT and the LoRA module. This lightweight configuration allows our model to be efficiently trained on a single 4090 GPU. We assemble the tables from training and validation sets of the datasets mentioned above along with a selection of high-quality tables from the TURL as a comprehensive dataset consisting of 100k tables for use in the pre-training stage. Moreover, recognizing GPT's limited sensitivity to numerical data, we prefix cells containing only numbers with "Float Value:" and so on, which can be achieved by regular expression. This enhancement is aimed at bolstering the semantic information conveyed by these cells.

General Setup across Evaluation Tasks. Given our aim to validate the HGT's performance on few-shot TU, we randomly generate five 3-shot tasks for both training and validation on each dataset (i.e., 3 train/dev tables for CTC, 3 train/dev tables per class for TTC and 3 train/dev QA pairs for Tale QA). Each training task, paired with a corresponding validation task, is utilized to fine-tune the models optimally for subsequent testing. Table 2 presents the average performance of a framework on the test set across these five tasks. In addition, to align the experiments more closely with real-world scenarios, we adhere to the setup outlined in Zheng et al. (2023) and refrain from explicitly providing the model with the hierarchical structure information in the header row / columns, despite some datasets being explicitly labeled with this detail. The detailed setup of baselines is shown in Appendix C.

Evaluation Metrics. For CTC and TTC, we adopt Macro-F1 as the evaluation metric. Considering that our framework, along with some baselines, employs a generative architecture, relying solely on exact match might inaccurately categorize some correct responses (e.g., generating an answer with an additional period at the end). Therefore, we incorporate the semantic-match (Cheng et al., 2022) and human evaluation as the evaluation metric of Table QA.

[1]For a fair comparison with our framework, we use 3-shot prompting for Binder and Aug-Codex, which decreases performance compared to the results in their paper. More details are shown in Appendix C.

5.4 Results and Analysis

The experimental results presented in Table 2 shows that HGT achieves the best performance on 7 out of 8 datasets, with the WTQ (Raw) dataset being the sole exception where Binder outperforms HGT. The following analysis sheds light on these outcomes: 1) While Query Statement-based frameworks demonstrate strong capabilities in Table QA, their architectural design limits their ability to perform other tabular tasks. Compared to other end-to-end frameworks, their generality is somewhat lacking. 2) WTQ (Raw) represents the unmodified version of WTQ (Flat), retaining the complexity of hierarchical headers within tables. A visual depiction of the difference between these versions is provided in Fig. 8. The noticeable drop in performance of other methods on WTQ (Raw) underscores HGT's capability to effectively process complex table structures. 3) The fact that other LLM-based frameworks underperform compared to HGT underscores the efficacy of our framework in mitigating the drawbacks associated with linearized table representations which locks the full capabilities of LLMs. 4) Without sufficient data for fine-tuning the downstream tasks, pre-trained models fail to reach their full potential. 5) On the IM-TQA dataset, the performance of Query Statement-based frameworks falls short compared to the LLM-based frameworks. We surmise this discrepancy is likely due to the dataset featuring particularly complex tables and look-up type questions, while Statement-based frameworks excel in different problem types, such as arithmetic queries. The results of the experiments exploring HGT at different numbers of training samples are detailed in Appendix. D.

5.5 Ablation Study

We conduct ablation experiments on a single dataset IM-TQA, the only dataset containing three types of labels, to validate the effectiveness of each component and self-supervised objective, the results of which are shown in Table 3.

We first conduct ablation studies w/o three self-training tasks respectively. We pre-train three models without TRC, TCM, and TCG respectively to assess their individual impact. These three pre-training tasks prove beneficial across all table evaluation tasks, with TCM offering the most substantial enhancement across three evaluation tasks—enhancements of 5.54%, 3.53%, and 6.70%.

Modules	CTC	TTC	Table QA
Full HGT	**66.24**	**59.58**	**52.95**
w/o TRC	61.71	58.03	47.05
w/o TCM	60.70	56.05	46.25
w/o TCG	65.38	57.46	50.56
w/o HG	62.42	58.55	49.44
w/o hl	63.09	58.87	48.96

Table 3: Ablation results on IM-TQA dataset. The "w/o TRC", "w/o TCM", and "w/o TCG": HGT pre-trained without the specified objective. The "w/o HG": HGT pre-trained with homogeneous graph. The "w/o hl": HGT pre-trained without the heuristic linking strategy described earlier, instead linking all neighboring cells directly.

This improvement is akin to how a human can only effectively organize disordered cells by understanding the table's semantics and the interrelations between cells. The contribution of TRC is particularly noticeable in CTC and Table QA compared to TTC, with improvements of 4.53% and 5.95% respectively. as it aids the model in recognizing that the semantics of cells in header rows and data rows differ, even when their content is identical. TCG is especially advantageous for TTC, more so than the other tasks, aligning with our expectations.

We then conduct ablation studies about our method to construct tabular graphs. This involves converting tables into homogeneous graphs rather than heterogeneous ones and replacing the heuristic linking strategy with a raw method linking all neighboring cells. Both variants result in substantial performance declines in CTC and Table QA, with at least 3.15% and 3.51% reduction on these two tasks, respectively. It indicates that our conversion method is more effective than existing methods for modeling complex tables.

6 Conclusion

In this paper, we introduce a novel framework, HGT, tailored for few-shot complex TU. The effectiveness of HGT is validated across multiple datasets for CTC, TTC, and Table QA, accompanied by an in-depth ablation study to examine the impact of each component.

In future work, we plan to expand HGT's applicability to tables featuring more diverse layouts and to further improve HGT's performance in Table QA tasks by implementing techniques that enhance the model's inference capabilities.

Limitations

Firstly, HGT's performance notably declines when dealing with tables that have irregular layouts, such as those containing sub-titles or images, or forms filled with personalized information. Secondly, HGT tends to consume excessive GPU memory when processing larger tables. Lastly, when faced with Excel-type forms, it is needed to use the API to convert them to html-format tables first.

References

Yihan Cao, Shuyi Chen, Ryan Liu, Zhiruo Wang, and Daniel Fried. 2023. Api-assisted code generation for question answering on varied table structures. *ArXiv*, abs/2310.14687.

Wenhu Chen. 2022. Large language models are few(1)-shot table reasoners. *ArXiv*, abs/2210.06710.

Zhoujun Cheng, Haoyu Dong, Fan Cheng, Ran Jia, Pengfei Wu, Shi Han, and Dongmei Zhang. 2021a. Fortap: Using formulas for numerical-reasoning-aware table pretraining. In *Annual Meeting of the Association for Computational Linguistics*.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2021b. Hitab: A hierarchical table dataset for question answering and natural language generation. *ArXiv*, abs/2108.06712.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, R.K. Nadkarni, Yushi Hu, Caiming Xiong, Dragomir R. Radev, Marilyn Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2022. Binding language models in symbolic languages. *ArXiv*, abs/2210.02875.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Eric Crestan and Patrick Pantel. 2011. Web-scale table census and classification. In *Web Search and Data Mining*.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. Turl: Table understanding through representation learning. *SIGMOD Rec.*, 51:33–40.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805.

Lun Du, Fei Gao, Xu Chen, Ran Jia, Junshan Wang, Shi Han, and Dongmei Zhang. 2021. Tabularnet: A neural network architecture for understanding semantic structures of tabular data. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

Majid Ghasemi-Gol, Jay Pujara, and Pedro A. Szekely. 2019. Tabular cell classification using pre-trained cell embeddings. *2019 IEEE International Conference on Data Mining (ICDM)*, pages 230–239.

Majid Ghasemi-Gol and Pedro A. Szekely. 2018. Tabvec: Table vectors for classification of web tables. *ArXiv*, abs/1802.06290.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *ArXiv*, abs/2004.02349.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Ran Jia, Haoming Guo, Xiaoyu Jin, Chao Yan, Lun Du, Xiaojun Ma, Tamara Stankovic, Marko Lozajic, Goran Zoranovic, Igor Ilic, Shi Han, and Dongmei Zhang. 2023. Getpt: Graph-enhanced general table pre-training with alternate attention network. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiao Long Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. 2023a. Time-llm: Time series forecasting by reprogramming large language models. *ArXiv*, abs/2310.01728.

Rihui Jin, Jianan Wang, Wei Tan, Yongrui Chen, Guilin Qi, and Wang Hao. 2023b. Tabprompt: Graph-based pre-training and prompting for few-shot table understanding. In *Conference on Empirical Methods in Natural Language Processing*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *ArXiv*, abs/2304.08485.

Shuai Ma, Jian-wei Liu, and Xin Zuo. 2023. Self-supervised learning for heterogeneous graph via structure information based on metapath. *Applied Soft Computing*, 143:110388.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton,

Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022a. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022b. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Annual Meeting of the Association for Computational Linguistics*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2023. Graphgpt: Graph instruction tuning for large language models. *ArXiv*, abs/2310.13023.

Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro Szekely. 2021. 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*, pages 1472–1482.

Kai Wang, Weizhou Shen, Yunyi Yang, Xiaojun Quan, and Rui Wang. 2020a. Relational graph attention network for aspect-based sentiment analysis. In *ACL*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. In *Annual Meeting of the Association for Computational Linguistics*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2020b. Tuta: Tree-based transformers for generally structured table pre-training. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

Yaming Yang, Ziyu Guan, Zhe Wang, Wei Zhao, Cai Xu, Weigang Lu, and Jianbin Huang. 2022. Self-supervised heterogeneous graph pre-training based on structural clustering. *Advances in Neural Information Processing Systems*, 35:16962–16974.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *ArXiv*, abs/2005.08314.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023. Tablellama: Towards open large generalist models for tables. *ArXiv*, abs/2311.09206.

Mingyu Zheng, Yang Hao, Wen-Jie Jiang, Zheng Lin, Yajuan Lyu, Qiaoqiao She, and Weiping Wang. 2023. Im-tqa: A chinese table question answering dataset with implicit and multi-type table structures. In *Annual Meeting of the Association for Computational Linguistics*.

A Datasets and Pre-processing

is located in left header columns	is located in top header rows	Cell Type
✓	✓	column attribute
✓	×	column attribute
×	✓	row attribute
×	×	value

Figure 5: Annotation correspondence of HiTab.

cells stretching across rows in the first row
cells stretching across rows in the first column
float type cells in the first row
float type cells in the first column
Average distance in vector space for the first row of cells
Average distance in vector space for the first column of cells
rows of the table
columns of the table
variance of cell vector space distance in a row-wise way
variance of cell vector space distance in a column-wise way

Figure 6: Manual Features for Table Orientation Classification.

IM-TQA (Zheng et al., 2023) consists of tables in Chinese, gathered from encyclopedias and reports, and translated into English. These tables are annotated for the three tasks: CTC, TTC, and Table QA. Due to our method of converting tables into HGs not being suitable for Vertical tables—where header cells do not align with the table rows—we employ a feature engineering method to pre-process and convert Vertical tables into Horizontal tables. This method has demonstrated a 98 % accuracy rate, with the relevant features depicted in Fig. 6. *HiTab* (Cheng et al., 2021b) features hierarchical matrix tables sourced from statistical reports and Wikipedia articles, annotated for row and column granularity as well as Table QA. We also transform these broad-granularity annotations into cell-level annotations, as shown in Fig. 5. *WTQ (Flatten)* and *WTQ (Raw)* originate from the same work (Pasupat and Liang, 2015) and consist of tables sourced from Wikipedia, accompanied by questions that involve basic arithmetic operations like sum and max, and sometimes necessitate compositional reasoning. To simplify the

Table QA task, the authors converted the hierarchical structure of header rows into a single row in the *WTQ (Raw)* dataset, resulting in the *WTQ (Flatten)* version. An illustration of this transformation is presented in Fig. 8. *WCC* (Ghasemi-Gol and Szekely, 2018), a subset of the *July 2015 Common Crawl*, is annotated for TTC using the web table taxonomy introduced by Crestan and Pantel 2011. It is not explicitly divided into training and test sets, so we use the 353 tables that were not sampled as the test set. Due to the GPU memory limit, we exclude the tables containing more than 150 cells.

B Top Header Row Number

The heuristic rule of acquiring the Top Header Row Number is shown in Alg. 1. Alg. 2 aims to determine the type of each row of a table.

Algorithm 1 Acquisition of Top Header Row Number

function $getThrn$(Table X)
    TopHeadRowNumber $thrn_X \leftarrow 0$
    for Cell x in $X.firstRow$ **do**:
        // row span is an attribute of each cell
        RowSpan $rs \leftarrow$ x.getRowSpan()
        $thrn \leftarrow max(rs, thrn)$
    end for
    Return $thrn$
end function

Algorithm 2 Identify Row Type

function $identifyRowType$(Table X, Row R)
    $thrn_X \leftarrow getThrn(X)$
    $r_{th} \leftarrow getRowLocationNumber(X)$
    // r_{th} represents R located in the r_{th} row of table X
    Cell $x \leftarrow R.getFirstCell()$
    // Cell x is the first cell located in row R
    if $r_{th} < thrn_X$ **then**
        Return Header-Row
    else if $x.getColumnSpan() = X.getColumnNumber$ **then**
        Return Header-Row
    else
        Return Data-Row
    end if
end function

C Setup of Baselines

This section introduces the setup of baselines. GetPt and Aug-Codex have not made their source code publicly available as of yet, leading us to replicate the framework based on the details provided in their paper. TableLlama, which does not utilize the Cell Type Classification (CTC) and Table Type Classification (TTC) datasets for training, exhibits suboptimal performance when inputs are presented alongside in-context prompts during testing. To address this, we fine-tuned the model's parameters using few-shot training data prior to testing. Additionally, given that Tablellama was trained using a substantial volume of data from the HiTab dataset, its test results are not indicative of performance in few-shot scenarios. For GPT-3.5, we employed API calls for the testing, where the training data was integrated as in-context prompts during the test phase.

D Results of varied training shots on HiTab

We further explore the performance of HGT under different numbers of training sampleswith the results depicted in Fig. 7. We choose Aug-codex, Fortap, and Tablellama as the baselines for comparison. In Fig. 7, the vertical axis represents the accuracy, and the horizontal axis represents the number of training samples, which are 1, 3, 5, and full, respectively. The full setting refers to the use of any amount of data from the training set until optimal results are achieved by the model. It is also noteworthy that Tablellama's accuracy is only reported in full settings, as it incorporates HiTab training data during its training phase.

As you can see from the graph, HGT performs best with a limited number of samples and maintains competitiveness with the state-of-the-art Aug-Codex even in full settings.

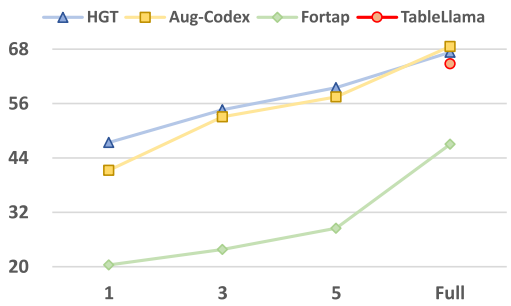
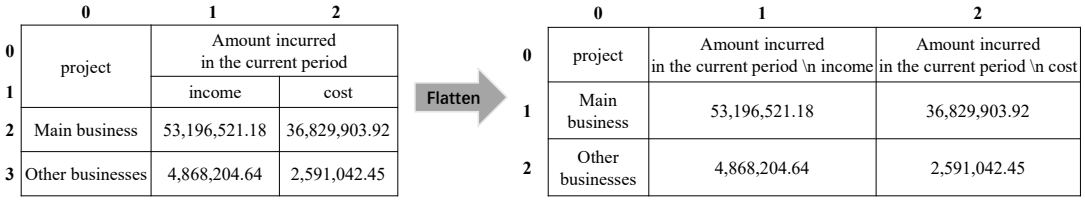


Figure 7: Impact of varied shots on HiTab for Table QA.



	0	1	2
0	project	Amount incurred in the current period	
1		income	cost
2	Main business	53,196,521.18	36,829,903.92
3	Other businesses	4,868,204.64	2,591,042.45

	0	1	2
0	project	Amount incurred in the current period \n income	Amount incurred in the current period \n cost
1	Main business	53,196,521.18	36,829,903.92
2	Other businesses	4,868,204.64	2,591,042.45

Figure 8: Comparison of a table before and after being flattened

Q: Given a sequence of Table CELL tokens ***<table_start> <tabular_node>...< tabular_node> <table_end>*** *that constitute a table. Each table CELL token contains the content within the CELL. Tabular CELLs can be divided into five types: row-index, column-index, row-attribute, column-attribute, value-CELL. Please* ***classify each CELL*** *according to the table CELL tokens.*
A: Based on the given 10 Table CELL tokens, we can classify them as follows: The type of Table CELL token 0 is column-attribute. The type of Table CELL token 1 is column-attribute. The type of Table CELL token 2 is column-attribute. The type of Table CELL token 3 is column-attribute. The type of Table CELL token 4 is row-index. The type of Table CELL token 5 is value-cell. The type of Table CELL token 6 is value-cell. The type of Table CELL token 7 is row-index. The type of Table CELL token 8 is value-cell. The type of Table CELL token 9 is value-cell.

Q: Given a tabular TABLE token ***<table_start> <tabular_node> <table_end>*** *that represents a table. The TABLE token contains the content within the table. Please* ***classify the table*** *according to the tabular TABLE token.*
A: Based on the given Table CELL token, we can classify the table as hierarchical.

Q: Given a sequence of Table CELL tokens ***<table_start> <tabular_node>...< tabular_node> <table_end>*** *that constitute a table. Each table cell token contains the content within the cell. The table is as follows, each cell is separated by "| ":*
project | Amount incurred in the current period | Amount incurred in the current period |
project | income | cost |
Main business | Float Value: 53,196,521.18 | Float Value: 36,829,903.92 |
Other businesses | Float Value: 4,868,204.64 | Float Value: 2,591,042.45 |
Please ***answer the following question*** *according to the table cell tokens:*
Question: How much are the income and cost of the other businesses of the company's current amount?
A: Based on the given table cells and their tokens, the corresponding Answer: 4,868,204.64, 2,591,042.45.'

Figure 9: Examples of three downstream table-related tasks presented as instructions include: Cell Type Classification, Table Type Classification, and Table QA.

E Downstream Table-related Task Examples

Sample inputs for CTC, TTC, and Table QA are shown in Fig. 9.