TrustUQA: A Trustful Framework for Unified Structured Data Question Answering

Wen Zhang[1,4], Long Jin[1], Yushan Zhu[1], Jiaoyan Chen[2], Zhiwei Huang[1], Junjie Wang[1], Yin Hua[1], Lei Liang[3], Huajun Chen[1,4,5*]

[1]Zhejiang University
[2]University of Manchester
[3]Ant Group
[4]ZJU-Ant Group Joint Lab of Knowledge Graph
[5]Zhejiang Key Laboratory of Big Data Intelligent Computing
{zhang.wen, longjin, yushanzhu, huajunsir}@zju.edu.cn

Abstract

Natural language question answering (QA) over structured data sources such as tables and knowledge graphs have been widely investigated, especially with Large Language Models (LLMs) in recent years. The main solutions include question to formal query parsing and retrieval-based answer generation. However, current methods of the former often suffer from weak generalization, failing to dealing with multi-types of sources, while the later is limited in trustfulness. In this paper, we propose TrustUQA, a trustful QA framework that can simultaneously support multiple types of structured data in a unified way. To this end, it adopts an LLM-friendly and unified knowledge representation method called *Condition Graph* (CG), and uses an LLM and demonstration-based two-level method for CG querying. For enhancement, it is also equipped with dynamic demonstration retrieval. We have evaluated TrustUQA with 5 benchmarks covering 3 types of structured data. It outperforms 2 existing unified structured data QA methods. In comparison with the baselines that are specific to one data type, it achieves state-of-the-art on 2 of the datasets. Further more, we have demonstrated the potential of our method for more general QA tasks, QA over mixed structured data and QA across structured data.

Code — https://github.com/zjukg/TrustUQA
Extended version — https://arxiv.org/abs/2406.18916

1 Introduction

Question answering (QA) seeking answers for a natural language question from structured data has attracted increasing attention in the past decade (2023a; 2023), leading to a few directions including QA over tables (2023), QA over knowledge graph (i.e., KGQA) (2018), QA over temporal KG (2023b) and so on. A straightforward and widely studied solution is to parse questions into formal queries that can be executed on data storage and reasoning engines (a.k.a. NL2Query). According to the data source types, these works are divided into NL2SQL (2020) for relational databases, NL2SPARQL (2020) for KGs, etc. However, current NL2Query methods are specifically developed for one query language corresponding to one data type.This significantly limits their generality and usage in real-world scenarios, especially when it is unknown in which data resource the answer lies, or the answer relies on multiple data sources.

With the development of LLMs and Retrieval-augmented Generation (RAG), another solution, which first retrieves relevant evidences of the question from data sources and then generates the answer, has become more and more popular (2023a; 2024). Using different retrieval techniques for different structured data, this solution can lead to more general methods. For example, StructGPT(2023a) iteratively retrieves evidences and feeds them into an LLM for answer generation, supporting KG, table and relational database. However, this solution suffers from some trustfulness issues: *(i)* the generated answer may be inconsistent with the original structured data due to the hallucination of LLMs (2023b), the insistence of LLMs' parametric knowledge (2023) and the irrelevant evidences that are retrieved; *(ii)* a list of evidences are exposed to the LLM, potentially leaking private data if a third-party LLM is used; *(iii)* the evidences can provide some plausible citations to the answer but it is hard to provide high quality explanations. In contrast, NL2Query can mostly avoid the issue of *(i)* and *(ii)* as only some meta data and prototypes need to be exposed to the LLM to generate the query instead of the answer, and the issue of *(iii)* since the query and its reasoning procedure are both accessible, providing logically rigorous explanation to the answer. Therefore, in this work, we adopt NL2Query, and propose **a trustful and unified framework named TrustUQA supporting different types of structured data simultaneously.**

There are two desiderata for TrustUQA. (1) It is expected to represent diverse structured data. Thus we propose **Condition Graph** (CG) and corresponding techniques for translating tables, KGs and temporal KGs into a CG. (2) It can support effective querying over the data representation. Thus we propose a method called **Lwo-layer Function-based CG Query**, which firstly uses LLMs to write basic queries (i.e., LLM queries) based on the question, and then uses predefined rules to transform these LLM queries to queries that can be executed on the CG (i.e., execution queries) for the eventual answer. Functions of the LLM query are designed with simple vocabularies that are more understandable by LLMs, and thus this method can get higher accuracy using no fine-tuning but few-shot prompting. We also propose a **dy-**

*Corresponding Author

namic demonstration retrieving method to further improve the prompt quality for higher accuracy.

To evaluate the effectiveness of TrustUQA, we experiment it on 5 benchmarks covering 3 kinds of structured data, including WikiSQL (2017) and WTQ (2015) for table QA, WebQSP (2016) and MetaQA (2018) for KG, and CronQuestion (2021) for temporal KG. We compare its results with existing RAG-based unified QA methods (2023a; 2024). Results show that TrustUQA outperforms these RAG-based methods on WikiSQL and WebQSP. It also achieves state-of-the-art on WebQSP and CronQuestions compared to the models that support only one type of structured data.

Besides the comprehensive model analysis, we further demonstrate the generality of TrustUQA by questions whose answers potentially rely on different data sources — tables, KG and temporal KG. It includes two tasks: *QA over mixed structured data* where each answer relies on one of the given data sources but which source it relies on is not specified, and *QA across structured data* where answering the question relies on data from more than one sources. These tasks are close to real-world scenarios, but have not been explored.

In summary, our contributions lie in three aspects: (a) We propose a trustful framework TrustUQA for unified QA over multiple kinds of structured data. (b) We conduct comprehensives experiments over 5 benchmarks covering 3 types of structured data, proving the effectiveness of TrustUQA. (c) We demonstrate the generality of TrustUQA on two practical tasks that have never been explored before — QA over mixed structured data and QA across structured data.

2 Related Works

NL2Query. For table QA, the query language is SQL and the task is also known as NL2SQL. NL2SQL methods include schema-based approaches that build the query based on the schema of the database and the data indices (2016; 2011), parsing-based approaches that parse questions through grammatical structures (2017; 2017), and neural-machine translation-based approaches that model NL2SQL as a language translation problem (2019; 2020). Methods based on pre-trained language models (PLMs) fine-tune PLMs to generate SQL queries (2022; 2023a), complete the SQL query sketch (2023), address the sub-queries (2023). We refer (2023) for a comprehensive survey on NL2SQL.

For KGQA, the formal query language is SPARQL, thus the task is also called NL2SPARQL. The methods usually include three steps: question understanding, linking and filtering (2023). In question understanding, some works (2018; 2021) use natural language parsers such as dependency parsing while the others (2023; 2021) use sequence to sequence generation to get graph patterns. The linking step maps the mentions in questions to the entities and relations in the KG, where mapping dictionary (2012), indexing system (2019; 2018), embedding generation system (2020) are usually applied. In filtering step, the type constraint is applied to filter answers. LLMs are also applied for this task, in which generate-then-retrieve paradigm are usually applied that firstly generate the logical form and then bind the mentions to elements in KGs (2023b; 2023).

These NL2Query methods are developed towards one specific query language or even for one specific dataset, lacking generality for supporting different types of structured data.

NL2Answer. NL2Answer methods directly generate the answer, skipping the formal query. For example, KGQA can be modeled as multi-hop reasoning (2018). Some works (2023c; 2018) trains a text encoder to encode the question and a reasoning module to perform multi-hop reasoning. The others (2023b) use a unified model for text encoding and reasoning. Since these models need to be trained, they are often only applicable to some specific datasets after training. To address this problem, more general pre-training solutions that work across datasets are proposed. For example, TableGPT (2023) is a unified framework that enables LLMs to understand and operate on tables following natural languages. It pre-trains a general table encoder encoding tables into embeddings, and prompt-tuning the LLM with table embedding as input over a vast corpus. These methods are applicable to multiple datasets but cannot support different types of structured data.

Unified QA. With LLMs, some unified QA frameworks that can support different types of structured data have been proposed. StructGPT (2023a) is an iterative reading-then-reasoning framework. In reading, it collects evidence from the structured data through specialized functions for different kinds of structured data. In reasoning, it uses LLMs to generate the answer or the next reasoning step based on the collected evidence. Readi (2024) is a reasoning-path-editing framework. Given a question, it first generates a reasoning path, and edits the path according to the feedback from instantiating the path over the structured data. It collects the KG evidence based on the edited reasoning paths and uses LLMs to generate the answer based on the evidence and the question. Though these two methods could be applied to different types of structured data, they are not unified enough, because the functions in StructGPT and the reasoning paths in Readi are specific to data types. More importantly, they generate the answer based on evidence retrieval, and thus are not as trustful as NL2Query methods, as we discussed in Introduction.

3 Methodology

TrustUQA uses Condition Graph (CG) for general and expressive data representation. It can represent simple relationships, complex temporal facts, and rules. For composing execution queries over CG, we design execution query function e.g., $search\_node()$ and $search\_condition()$. As directly writing execution queries by LLMs is challenging due to their complexity and novelty, we propose a querying method called *Two-layer Function-based CG Query*. In the first layer, it uses LLMs to write simplified functions like $get\_information()$, whose vocabularies (head entity, tail entity, relation, key and value) are general and more understandable to LLMs, and applies such LLM functions for composing LLM queries. In the second layer, it translates each LLM query into an execution query according to predefined rules. To augment LLMs for writing LLM queries, we apply few-shot prompting with a few demonstrations of question and LLM query pairs, and

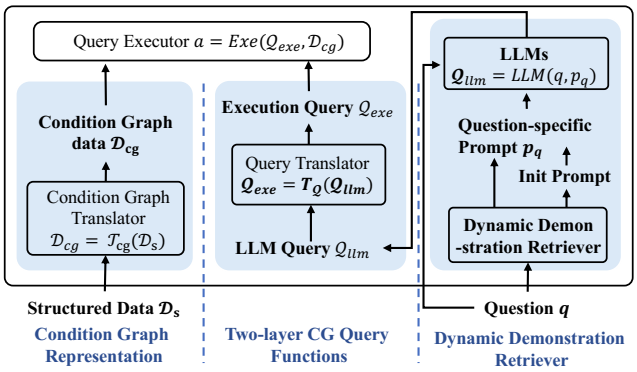

Figure 1: Overview of the TrustUQA framework

propose a dynamic demonstration retriever, since the optimal demonstration varies with each question.

Briefly, TrustUQA consists of 3 main modules, as shown in Figure 1: (1) Condition Graph Translator (T_{cg}) transforming input structured data into a CG, (2) Query Translator (T_q) translating LLM queries to execution queries over the CG, (3) Dynamic Demonstration Retriever (R) selecting the most similar examples from the training data as demonstrations for writing LLM queries.

3.1 Condition Graph Translator T_{cg}

Given structured data $\mathcal{D}_s$, T_{cg} translates it into a condition graph which is formally defined below.

Definition 1. ***Condition Graph*** *is a labeled directed graph represented as $\mathcal{CG} = \{\mathcal{N}, \mathcal{T}\}$. $\mathcal{N}$ is the set of nodes in CG. Each node is combined with a string to represent the semantic meaning, which could be entities such as* `Earth`*, relationships such as* `has friends`*, properties such as* `time`*, or numerical values such as* `2024`*. $\mathcal{T} = \{(node_1, node_2, condition) | node_1 \in \mathcal{N}, node_2 \in \mathcal{N}\}$ is a collection of condition triples representing an edge that $node_1$ is connected to $node_2$ under the condition, where $condition = [node_a, node_b, ...]$ is a list of nodes in $\mathcal{N}$. The condition can be empty ([]) indicating that $node_1$ is connected to $node_2$ without any conditions.*

Condition graph is expressive. The triple in the CG can represent (1) simple relationships such as (`Born In`, `Ulm`, [`Einstein`]), (2) complex facts such as (`date`, `14 March 1879`, [`Einstein`, `Born In`, `Ulm`]), (3) common rules such as (`Person`, `Has Parents`, []), meaning that each person has parents.

Condition graph is a unified representation. Next, we introduce how to translate three typical structured data — tables, KGs, and temporal KGs, into a CG.

Translating Tables. A table as $\mathcal{D}_s$ is represented as $Table = \{R, C, V\}$ which contains m rows $R = \{row_i\}_{i=1}^{m}$ and n columns $C = \{col_j\}_{j=1}^{n}$. The header row_1 records the semantic meaning of each column. $V = \{v_{i,j} | i \in [1, m], j \in [1, n]\}$ represents the cell values of the table. For example, a table with *name* | *born in city* | *Time* as the header and *Einstein* | *Ulm* | *14 March 1879* as the record. During transformation, the translator T_{cg} first inserts a column c_0 in front of the first column that $C := \{c_0, col_1, ..., col_n\}$. The values of the inserted column cells are strings indicating the order of the corresponding row, i.e. for the ith row, the inserted value is $[line\_i]$. For example, after insertion, the previous example table becomes *[line_1]| name | born in city | Time* as the header and *[line_2] | Einstein | Ulm | 14 March 1879* as the record. After inserting, starting from the second row and the second column, the translator will generate two condition triples for each cell. For example, for $v_{2,2}$ = *Einstein* in the example table, there will be ($[line\_2]$, `name`, []) represents the entity $[line\_2]$ has name, and (`name`, `Einstein`, $[line\_2]$) represents the name of entity $[line\_2]$ is Einstein. Formally, after translation, $\mathcal{D}_{cg}^{table} = \{\mathcal{N}_{table}, \mathcal{T}_{table}\}$, where $\mathcal{N}_{table} = V \cup \{line\_1, line\_2, ..., line\_m\}$, $\mathcal{T}_{table} = \{([line\_i], v_{1,j}, []), (v_{1,j}, v_{i,j}, [line\_i]) | i \in [2, m], j \in [2, n+1]\}$.

Translating Knowledge Graph. A KG as $\mathcal{D}_s$ is represented as $\mathcal{KG} = \{\mathcal{E}, \mathcal{R}, \mathcal{F}\}$, where $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{F}$ are the set of entities, relations and facts. $\mathcal{F} = \{(h, r, t) | \{h, t\} \in \mathcal{E}, r \in \mathcal{R}\}$ is a set of triples representing the relations between entities. Translator T_{cg} transforms a $\mathcal{KG}$ by generating two condition triples for each knowledge graph fact. For example, given a triple *(Einstein, born in city, Ulm)*, one condition triple (`Einstein`, `born in`, []) denotes Einstein has the property *born in*, another condition triple (`born In`, `Ulm`, [`Einstein`]) means if it is for Einstein, the value of *born in* is Ulm. Thus after translation, $\mathcal{D}_{cg}^{kg} = \{\mathcal{N}_{kg}, \mathcal{T}_{kg}\}$, where $\mathcal{N}_{kg} = \mathcal{E} \cup \mathcal{R}$, and $\mathcal{T}_{kg} = \{(h, r, []), (r, t, [h]) | (h, r, t) \in \mathcal{F}\}$.

Translating Temporal Knowledge Graph. A temporal KG as the $\mathcal{D}_s$ is represented as $\mathcal{TKG} = \{\mathcal{E}, \mathcal{R}, \tau, \mathcal{Q}\}$, where $\mathcal{E}, \mathcal{R}$ and τ are the entity, relation, and time sets, and $\mathcal{Q} = \{(h, r, t, \tau_s, \tau_e)\}$ is the quintuple set where $h, t \in \mathcal{E}$, $r \in \mathcal{R}$, and $\tau_s, \tau_e \in \tau$ denote the start time and end time of the fact (h, r, t), respectively. After translation, $\mathcal{T}_{tkg}$ collects all condition triples translated from each TKG triple. Specifically, for each $(h, r, t, \tau_s, \tau_e)$ in $\mathcal{Q}$, following condition triples are generated $(h, r, [])$, $(r, t, [h])$, (`start time`, τ_s, $[h, r, t]$), (`end time`, τ_e, $[h, r, t]$), $\{(\texttt{time}, \tau', [h, r, t]) | \tau' \in \{\tau_s, \tau_s + 1, , \tau_s + 2, ..., \tau_e\}\}$, where `start time`, `end time` and `time` are three built-in terms introduced during translation and τ' are the integer time stamps between the start time and the end time. Thus after translation, $\mathcal{D}_{cg}^{tkg} = \{\mathcal{N}_{tkg}, \mathcal{T}_{tkg}\}$. $\mathcal{N}_{tkg} = \mathcal{E} \cup \mathcal{R} \cup \tau \cup \{\texttt{start time}, \texttt{end time}, \texttt{time}\} \cup \{\tau'\}$.

3.2 Query Translator T_Q

The query translator T_Q translates the LLM query $\mathcal{Q}_{llm}$ into the execution query $\mathcal{Q}_{exe}$ that can be executed over the CG.

LLM Query $\mathcal{Q}_{llm}$. The LLM query is generated by an LLM corresponding to the question. LLMs know the common vocabularies of head entity, relation, tail entity from KGs, as well as key, value used in tables and property graphs well. Thus we design the following *searching function* to search information from the graph:

$$get\_information(head\_entity, relation, tail\_entity, key, value) \quad (1)$$

LLM Query Function	Execution Query Function
$gi(h = H)$	$sn(n1 = H)$
$gi(r = R)$	$sn(n1 = R)$
$gi(k = K)$	$sn(n1 = K)$
$gi(h = H, r = R)$	$sn(n1 = R, c = H)$
$gi(h = H, k = K)$	$sn(n1 = K, c = H)$
$gi(r = R, t\text{“=”}T)$	$sc(n1 = R, n2 = T, op =\text{“=”})$
$gi(k = K, v\text{“=”}V)$	$sc(n1 = K, n2 = V, op =\text{“=”})$
$gi(r = R, t\text{“=”}T, k = K, v\text{“=”}V)$	$output\_of\_query_1 = sc(n1 = R, n2 = T, op =\text{“=”})$ $output\_of\_query_2 = sc(n1 = K, n2 = V, op =\text{“=”})$ $set\_interaction(set1 = output\_of\_query_1, set2 = output\_of\_query_2)$
$gi(r = R, t\text{“=”}T, k = K)$	$output\_of\_query_1 = sc(n1 = R, n2 = T, op =\text{“=”})$ $sn(n1 = K, s = output\_of\_query_1)$
$gi(r = R, k = K, v\text{“=”}V)$	$output\_of\_query_1 = sc(n1 = K, n2 = V, op =\text{“=”})$ $sn(n1 = R, s = output\_of\_query_1)$

Table 1: Rules for translating LLM query functions to execution query functions. gi, h, r, t, k, v denote $get\_information$, $head\_entity$, $relation$, $tail\_entity$, key, and $value$, sn, sc, $n1$, $n2$, c, and s denote $search\_node$, $searh\_condition$, $node_1$, $node_2$, $condition$ and $scope$. For the parameters t and v of $get\_information$, op is one of $\{>, <, =, \geq, \leq\}$ and is "=" in the table as example.

where the variables $head\_entity$, $relation$, $tail\_entity$, key and $value$ are set to $None$ by default. This function is able to represent complex queries. For example, $get\_information(head\_entity$=$None, relation$=Won, $tail\_entity$=Nobel Prize, key=Year, $value > 2000)$ represents "Who are the Nobel Prize Winners after 2000?" $get\_information(head\_entity$=Einstein, $relation$=Won, $tail\_entity$=Nobel Prize, key=Year, $value$=$None)$ corresponds to "In which year did Einstein won the Nobel Prize?"

Apart from $get\_information()$, we also design a set of *reasoning functions* as follows:

- Set operations: $set\_intersection(set_1, set_2)$, $set\_union(set_1, set_2)$, $set\_difference(set_1, set_2)$, $set\_negation(set_1, set_2)$, and $keep(set, value)$[1]
- Simple calculations: $mean()$, $max()$, $min()$, $count()$.

With these functions, multiple (nested) searching and reasoning steps can be expressed by LLM queries. We give examples of these reasoning functions in Appendix.

Execution Query $\mathcal{Q}_{exe}$. The execution query is executed over the CG, and is composed of a set of *execution query functions* which include *searching functions* and *reasoning functions*. The reasoning functions are the same as the LLM reasoning functions. The search functions are used to obtain information from the CG following constrains:

- $search\_node(node_1, condition, scope)$: return $node_2$ of the condition triple $(node_1, ?, [condition])$ from the condition triple set denoted as $scope$. $condition$=$None$ means condition list is empty. $scope$=all by default means including all the condition triples in $\mathcal{D}_{cg}$. For example $search_node(node$1=Born In, $condition$=Einstein, $scope$=$all)$ means getting the born-in information of Einstein based on all condition triples.
- $search\_condition(node_1, node_2\_value, op)$: return $condition$ in the condition triple $(node_1, node_2, ?)$ such that $node_2$ satisfies the operation op with respect to $node_2\_value$. For example, if op is '>', the function return $condition$ where $node_2 > node_2\_value$. Specifically, $search\_condition(node_1$=Born, $node_2\_value$=2020, op=$' >')$ means searching entities borning after 2020. To determine if the op constrain meets, $compare(op, val_1, val_2)$ will be called.
- $compare(op, val_1, val_2)$: return $ture$ if val_1 $\{op\}$ val_2, otherwise return $false$, where op is one of the comparison symbols $\{>, <, =, \geq, \leq\}$. For example, $compare(op$=$' >'$, val_1=2024, val_2=2020) will return $true$.

Given a CG $\mathcal{D}_{cg}$ and complex execution queries based on these functions $\mathcal{Q}_{exe}$, the query executor Exe can automatically execute $\mathcal{Q}_{exe}$ over $\mathcal{D}_{cg}$ to get the answer.

Translating $\mathcal{Q}_{llm}$ to $\mathcal{Q}_{exe}$. There are semantic mapping and syntax mapping steps for T_Q to translate $\mathcal{Q}_{llm}$ to $\mathcal{Q}_{exe}$. (1) *Semantic mapping*: mapping the variable values in $\mathcal{Q}_{llm}$ to the nodes in the $\mathcal{D}_{cg}$. To achieve this, we use a dense text encoder E, such as SentenceBert (Reimers and Gurevych 2019), to encode the nodes and variable values into vectors and map each value to the most similar node. We replace the value by the string of the mapped node in the function, resulting $\mathcal{Q}'_{llm}$. (2) *Syntax mapping:* translating $\mathcal{Q}'_{llm}$ to $\mathcal{Q}_{exe}$ following a fixed set of rules. We summarize the translation rules in Table 1. There is a chance that the generated LLM queries are composed of undefined LLM query functions. For example, LLM might generate $compare($*A's score*, *B's score*$)$ for question "*For A and B, who has a larger score?*". For such LLM query, the translator T_Q keeps the query as it is during translation. And during execution, the *LLM function* is called to use LLM to get the output. Specifically, we include the function name and parameters as input to LLM to generate output. The *LLM function* prompt is shown in Appendix.

[1]An example is $keep(set = \{1, 2, 3\}, value < 2)$

3.3 Dynamic Demonstration Retriever R

Given a question q, we use an LLM to generate the LLM query, denoted as $\mathcal{Q}_{llm} = LLM(q, p_q)$ where p_q is another input acting as the question-specific prompt. p_q includes a few question-query examples, called demonstrations. The initial demonstrations are manually crafted with a focus on representativeness and diversity, shown in Appendix.

However, the optimal demonstration for different questions varies. For example, for question *"What is the most common language in Norway?"*, the LLM query of the question *"What is the major language spoken in Canada?"* is more informative than *"Where the queen of Denmark lives?"*. Thus we propose a dynamic demonstration retriever to retrieve k most similar questions of q from the training dataset D_{train}=$\{(q_{train}, a_{train})\}$ as the demonstrations. Specifically, given a question q, we use a text encoder E to encode q and training question q_{train} into vectors. Then we calculate the similarity of the question vectors and select the m most similar training questions (denoted as $\mathcal{S}$) where $m > k$. Then we iteratively generate the LLM query $\mathcal{Q}_{llm}^{q_i}$=$LLM(q_i, p_{q_i})$ from the most to the least similar $q_i \in \mathcal{S}$. If the result from $Exe(T_Q(\mathcal{Q}_{llm}^{q_i}), \mathcal{D}_{cg})$ exactly matches to the labeled answer a_{train}, we regard $(q_i, \mathcal{Q}_{llm}^{q_i})$ as a question-query pair demonstration. We repeat this step until k demonstrations are collected. If k demonstrations are not collected, we supplement the remaining ones by the initial demonstrations.

4 Experiments

We adopt 5 datasets covering 3 data types: WikiSQL (2017) and WTQ (2015) for table, WebQuestionsSP(WebQSP) (2016) and MetaQA (2018) for KG, and CronQuestions (2021) for temporal KG. Their statistics and demonstrations are shown in the Appendix.We use GPT-3.5 (gpt-3.5-turbo-0613) as the LLM with self-consistency strategy of 5 times, and SentenceBERT (2019) as the dense text encoder. If the answer is "None" due to mismatched entity-relation pairs and key-value inconsistencies etc., we implement the retry mechanism with 3 times trials. We set the number of retrieves $m = 15$ and the number of demonstrations $k = 8$.

4.1 Table QA Experiment

Methods	**WikiSQL**	**WTQ**
Data Type Specific Models		
MAPO (2018)	72.6	43.8
TAPAS (2020)	83.6	48.8
UnifiedSKG (2022)	86.0	49.3
TAPEX (2022)	**89.5**	57.5
DATER (2023)	-	**65.9**
Unified Models (with GPT3.5 as the LLM)		
StructGPT (2023a)	65.6	52.2
Readi (2024)	66.2	**66.7**
TrustUQA(ours)	**85.9**	44.2

Table 2: Denotation accuracy of Table QA.

Experiment Setting We adopt denotation accuracy (2023a) to assess whether the predicted answer matches the labeled answer based on set-level equivalence. We write 8 initial demonstrations for both WTQ and WikiSQL. We add the table's column names and one randomly selected record under each column in a linearized format behind the question.

Result Analysis As shown in Table 2, on WikiSQL, TrustUQA surpasses MAPO and TAPAS and approaches UnifiedSKG with an accuracy of 85.9%. Compared to unified models, it achieves a nearly 20% improvement.

On WTQ, TrustUQA achieves an accuracy of 44.2%, surpassing MAPO. However, there remains a gap compared to the other methods. We analyze the cases and find three reasons. Firstly, some questions have more than one correct answer, like *"name a player that had more than 5 league goals but no other goals"*, but only one is included in the labeled answer. Secondly, some tables are presented in a none standard format, such as the table for *"how many countries won a gold medal"* whose last row is the total statistics of the previous rows. Thirdly, there are limitations of TrustUQA during element mapping. For the question *"What are the number of times a race was held in August?"* the term "August" could refer to either "August 1st" or "August 4th", but the translator only retrieves one instance.

Methods	**MetaQA**			**WebQSP**
	1 hop	*2 hop*	*3 hop*	
Data Type Specific Models				
KV-Mem (2016)	96.2	82.7	48.9	46.7
GraftNet (2018)	97.0	94.8	77.7	66.4
EmbedKGQA (2020)	**97.5**	98.8	94.8	66.6
NSM (2021)	97.1	**99.9**	98.9	68.7
UniKGQA (2023c)	**97.5**	99.0	99.1	**75.1**
Davinvi-003 (2022)	52.1	25.3	42.5	48.3
KB-BINDER (2023b)	93.5	99.6	96.4	-
KB-BINDER-R	92.9	**99.9**	**99.5**	-
Unified Models (with GPT3.5 as the LLM)				
StructGPT (2023a)	97.1	97.3	87.0	69.6
Readi (2024)	**98.4**	**99.9**	**99.4**	74.3
TrustUQA(ours)	97.1	97.9	98.4	**83.5**
	99.94*	99.99*	99.99*	

Table 3: Hit@1 results of KG QA. Digits with * are set comparison accuracy.

4.2 Knowledge Graph QA

Experiment Setting For MetaQA, we construct 13, 15, and 11 demonstrations for 1-hop, 2-hop, and 3-hop, respectively. Include description of CG data, prompt also lists relations.

For WebQSP, we used a processed version of the official WebQSP Dataset.For each question, a relevant subset of the KG (a set of triples) is retrieved from Freebase. Some relations and entities in WebQSP represented by IDs without entity names lack clear semantic meaning. Thus we divide the questions in WebQSP into two types, questions regarding

Methods	All	Question Type		Answer Type	
		Com	*Sim*	*Ent*	*Tim*
Data Type Specific Models					
BERT (2019)	24.3	23.9	24.9	27.7	17.9
RoBERTa (2019)	22.5	21.7	23.7	25.1	17.7
EmbedKGQA(2020)	28.8	28.6	29.0	41.1	05.7
EaE (2020)	28.8	25.7	32.9	31.8	23.1
CronKGQA(2021)	64.7	39.2	98.7	69.9	54.9
TempoQR-S(2022)	79.9	65.5	99.0	87.6	65.3
TempoQR-H(2022)	91.8	86.4	99.0	92.6	90.3
TSQA (2022)	83.1	71.3	98.7	82.9	83.6
TMA (2023a)	78.4	63.2	98.7	79.2	74.3
CTRN (2023)	92.0	86.9	99.0	92.1	91.7
LGQA (2023b)	96.9	94.5	99.2	96.2	96.6
Unified Models (with GPT3.5 as the LLM)					
TrustUQA(ours)	**97.2**	**95.4**	**99.5**	**96.1**	**99.1**

Table 4: Hits@1 of temporal KG QA on CronQuestion.

semantic-clear and semantic-unclear relations which can be found in Appendix. The initial demonstrations include topic entity, first step relations (relations have topic entity) and second step relations for WebQSP.

Result Analysis We report the Hit@1 results in Table 3.

For MetaQA, TrustUQA performs competitively in comparison with existing unified methods. During our experiments, we discover name ambiguity issues in MetaQA. For instance, two movies might share the same name, making it unclear which one is referenced in a question. Consequently, the gold answer might only represent a subset of the true answers. Based on this analysis, higher Hit@1 results may not indicate better QA performance on MetaQA, especially when the Hit@1 results exceed 97%, due to the incompleteness of labeled answers. Therefore, we report the accuracy of set comparison between predicted and labeled answers (last row in Table 3), where predicted answers are considered correct if they include all labeled answers. TrustUQA achieves over 99.9% accuracy across all three types of questions. Figure 2(a) shows the frequency of numbers of answers predicted by TrustUQA and answers labeled. It shows the two frequency distributions are nearly identical, and thus we can conclude the TrustUQA performs perfectly on MetaQA and this is not due to giving a large number of answers.

For WebQSP, TrustUQA achieves a Hit@1 score of 83.5%, surpassing all the baselines and achieves the state-of-the-art.

4.3 Temporal Knowledge Graph QA

Experiment Setting We use Hits@1 for evaluation.We design 9 demonstrations, each including the question, relation, and annotation. Entities that are not originally presented in the annotation but exist in the entity list are appended at the end. We use the question and answer type of each question as metadata during retrieving.

Result Analysis As Table 4 shows, TrustUQA is the only unified model for this task and achieves the state-of-the-art. Since TrustUQA and LGQA could achieve high performance on CronQuestion, more challenging QA datasets about temporal KG are expected to be created.

4.4 Model Analysis

Ablation Study As shown in Table 5, *-dynamic* means static initial demonstration and *-two-layer* means making LLM directly generate the execution query, the performance of TrustUQA decrease in majority of the datasets.

	Wiki.	WTQ	MetaQA	Web.	CronQ.
TrustUQA	**85.9**	**44.2**	**99.97***	**83.5**	**97.2**
-dynamic	84.5	43.7	99.87*	75.1	96.1
-two-layer	84.4	**44.2**	99.87*	83.4	87.6

Table 5: Ablation Study. The result of MetaQA is the average of 1,2,3 hops.

Using Different LLMs and Text Encoders We have tested GPT-4 and other three text encoders on two Table QA datasets. The results can be seen in Table 6. The results show that TrustUQA performs slightly differently with various text encoders, and consistently achieves better performance with the more advanced LLM GPT-4 compared to GPT-3.5.

	SentB.	DPR	ANCE	M3E
WikiSQL	85.9/87.4	86.0/87.4	85.5/87.4	86.1/87.7
WTQ	44.2/52.1	44.5/51.4	45.3/51.2	44.5/50.8

Table 6: TrustUQA with different LLMs (GPT-3.5/GPT-4) and text encoders.

Efficiency Analysis We evaluate the time cost of each step of our method on the MetaQA dataset with 100 random samples[2]. Steps include (1) *dynamic demo retrieval*, (2) *generating* $\mathcal{Q}_{llm}$, (3) translating $\mathcal{Q}_{llm}$ to $\mathcal{Q}_{exe}$ and (4) execute the query. Results are in Figure 2(e). We can see that most of the time are cost for $\mathcal{Q}_{llm}$ generation. The dynamic demonstration retrieval, query translation and execution are quite efficient which takes less than 20% of the whole time.

Hyperparameter Analysis In Figure 2 (b)-(d), we show that more demonstrations and more re-try leads to slightly better results. And generating answers in the self-consistency strategy by 5 times achieves the best results. Considering the tradeoff between time/computation cost and the performance, we set a moderate number for demonstrations and retry, which are 8 and 3 respectively.

Error Analysis We recognized three types of errors.(1) The most frequent type is *query generation error*, such as logical error where the generated query is logically wrong, unclear

[2]Our system is equipped with 2*NVIDIA A100 PCIe 40GB GPUs, 40 physical cores across 2 sockets, each socket containing 20 cores. The Intel Xeon Gold 6148 processors operate at a base speed of 2.40 GHz, with a maximum of 3.70 GHz.

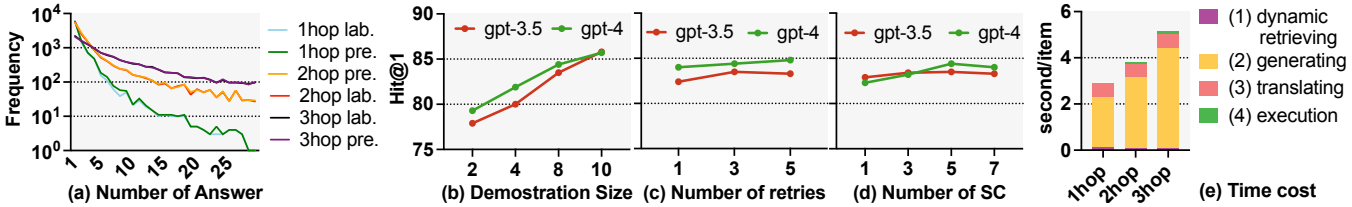

Figure 2: (a) The frequency of numbers of answers labeled and predicted by ours on MetaQA. (b) - (d) The hyperparameter of demostration size, retry and self-consistency on WebQSP. (e) The time cost of MetaQA.

	Wiki.	**WTQ**	**MetaQA**	**Web.**	**CronQ.**
Calling Rate	4.0%	30.7%	<1%	<1%	2.0%
Results	38.9%	11.4%	0.0%	2.2%	20.4%

Table 7: The calling rate and results of the LLM function.

	Wiki.	WTQ	MetaQA	Web.	CronQ.
D_{mix} + S_{unknown}	56.0	8.0	93.0	37.0	72.0
dynamic Acc	96%	42%	100%	51%	85%
D_{unmix} + S_{unknown}	79.0	25.0	97.0	46.0	84.0
D_{unmix} + S_{known}	85.0	42.0	99.0	84.0	95.0

Table 8: Results of QA over mixed structured data.

logic in query with steps either missing or unnecessary, and grammar error in query. (2) The second type is *mapping error* including errors in relation and entity mapping during the parameter-value mapping step. (3) The third type is *LLM function error*. When calling LLM function, though enough data are provided, LLM still generates wrong answers.

Calling Rate and Result of LLM function. Results are shown in Table 7. We observe the following: *(1) Except for WTQ, the usage rates under each dataset are generally low, with most being less than 1%. (2) The accuracy is low.* Compared to the experimental results on each dataset, the results of the LLM function are lower. Thus calling the execution functions we design in TrustUQA is more trustful.

4.5 Potentials of TrustUQA

QA over Mixed Structured Data In this task, each question relies on one of the mixed multiple data sources but which data source it relies on is not specified, which we call *mixed data (D_{mixed}) + answer source unknown ($S_{unknown}$)* setting. In contrast, the setting of the evaluation tasks in Section 4.1 - 4.3 is *unmixed data (D_{unmix}) + answer source known (S_{known})*. To simulate this scenario, we randomly extract 100 data from each of WikiSQL, WTQ, MetaQA-1hop, WebQSP, and CronQuestion, and experiment on these 500 test questions. During experiments, for D_{mix} setting, we translate data from 5 sources into CG form and store them together, and we store them independently for D_{unmix} setting. For S_{unknown} setting, we use a unified prompt with the demonstration's format only including the question, and for S_{known} setting, we use the same prompt for each data source as before.

Results are shown in Table 8, in which we report the results using the same evaluation metric on questions from each dataset. The results in the first-row show that TrustUQA could answer questions for different data sources and thus has the potential for QA over mixed structured data. But compared to the original results, i.e. the $D_{\text{unmix}} + D_{\text{known}}$ setting, performance drop occurs because the number of candidates for node-value mapping is significantly larger. Comparing the last two rows, we observe a significant performance drop.

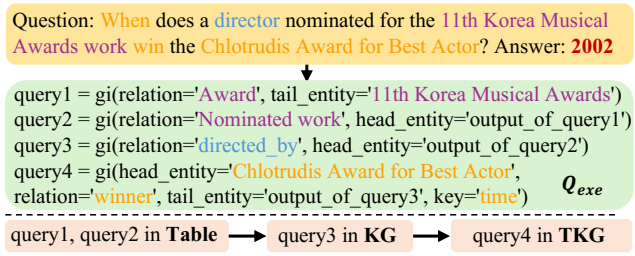

Figure 3: Case study of TrustUQA for across structured data.

This is because the results are positively correlated with the dynamic demonstration retrieval accuracy, whether the dynamic demonstrations are retrieved from the correct data source, as shown in the second row in Table 8. We refer the reader to Appendixfor more details of this experiment.

QA across Structured Data In this task, questions can be answered relying on more than one data source. We construct a case from 3 types of structure datasets: WikiSQL, MetaQA and CronQuestion. This case is shown in Figure 3. For question *When does a director nominated for the 11st Korea Musical Awards work win the Chlotrudis Award for Best Actor?* Q_{exe} first gets *Hedwig and the Angry Inch* from WikiSQL, then gets *John Cameron Mitchell* from MetaQA, and finally gets the correct answer *2002* from the CronQuestion. This case shows the potential of TrustUQA for QA across different types of structured data. We refer the reader to Appendixfor more details of this experiment.

5 Discussion and Conclusion

We introduce a trustful framework for unified structured data QA, called TrustUQA, which based on Condition Graph and two-layer query. We experimentally prove its effectiveness and the potential of dealing with more challenging scenarios.

Acknowledgments

This work is founded by National Natural Science Foundation of China (NSFC62306276/NSFCU23B2055/NSFCU19B2027), Zhejiang Provincial Natural Science Foundation of China (No. LQ23F020017), Yongjiang Talent Introduction Programme (2022A-238-G), and Fundamental Research Funds for the Central Universities (226-2023-00138). This work was supported by AntGroup.

References

Cheng, S.; Zhuang, Z.; Xu, Y.; Yang, F.; Zhang, C.; Qin, X.; Huang, X.; Chen, L.; Lin, Q.; Zhang, D.; et al. 2024. Call Me When Necessary: LLMs can Efficiently and Faithfully Reason over Structured Environments. *arXiv preprint arXiv:2403.08593*.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*.

Dubey, M.; Banerjee, D.; Chaudhuri, D.; and Lehmann, J. 2018. EARL: Joint Entity and Relation Linking for Question Answering over Knowledge Graphs. In *ISWC (1)*.

Févry, T.; Soares, L. B.; FitzGerald, N.; Choi, E.; and Kwiatkowski, T. 2020. Entities as Experts: Sparse Memory Access with Entity Supervision. In *EMNLP (1)*.

Gu, Z.; Fan, J.; Tang, N.; Zhang, S.; Zhang, Y.; Chen, Z.; Cao, L.; Li, G.; Madden, S.; and Du, X. 2023. Interleaving Pre-Trained Language Models and Large Language Models for Zero-Shot NL2SQL Generation. *CoRR*, abs/2306.08891.

Guo, J.; Zhan, Z.; Gao, Y.; Xiao, Y.; Lou, J.; Liu, T.; and Zhang, D. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *ACL (1)*.

He, G.; Lan, Y.; Jiang, J.; Zhao, W. X.; and Wen, J. 2021. Improving Multi-hop Knowledge Base Question Answering by Learning Intermediate Supervision Signals. In *WSDM*, 553–561. ACM.

Herzig, J.; Nowak, P. K.; Müller, T.; Piccinno, F.; and Eisenschlos, J. M. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *ACL*.

Hu, N.; Wu, Y.; Qi, G.; Min, D.; Chen, J.; Pan, J. Z.; and Ali, Z. 2023. An empirical study of pre-trained language models in simple knowledge graph question answering. *World Wide Web*, 26(5): 2855–2886.

Hu, S.; Zou, L.; Yu, J. X.; Wang, H.; and Zhao, D. 2018. Answering Natural Language Questions by Subgraph Matching over Knowledge Graphs. *IEEE Trans. Knowl. Data Eng.*, 30(5): 824–837.

Jiang, J.; Zhou, K.; Dong, Z.; Ye, K.; Zhao, X.; and Wen, J. 2023a. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *EMNLP*.

Jiang, J.; Zhou, K.; Zhao, W. X.; Li, Y.; and Wen, J. 2023b. ReasoningLM: Enabling Structural Subgraph Reasoning in Pre-trained Language Models for Question Answering over Knowledge Graph. In *EMNLP*.

Jiang, J.; Zhou, K.; Zhao, X.; and Wen, J. 2023c. UniKGQA: Unified Retrieval and Reasoning for Solving Multi-hop Question Answering Over Knowledge Graph. In *ICLR*. OpenReview.net.

Jiao, S.; Zhu, Z.; Wu, W.; Zuo, Z.; Qi, J.; Wang, W.; Zhang, G.; and Liu, P. 2023. An improving reasoning network for complex question answering over temporal knowledge graphs. *Appl. Intell.*, 53(7): 8195–8208.

Jung, H.; and Kim, W. 2020. Automated conversion from natural language query to SPARQL query. *J. Intell. Inf. Syst.*, 55(3): 501–520.

Kapanipathi, P.; Abdelaziz, I.; Ravishankar, S.; Roukos, S.; Gray, A. G.; Astudillo, R. F.; Chang, M.; and et al. 2021. Leveraging Abstract Meaning Representation for Knowledge Base Question Answering. In *ACL/IJCNLP (Findings)*.

Katsogiannis-Meimarakis, G.; and Koutrika, G. 2023. A survey on deep learning approaches for text-to-SQL. *VLDB J.*, 32(4): 905–936.

Kim, H.; So, B.; Han, W.; and Lee, H. 2020. Natural language to SQL: Where are we today? *Proc. VLDB Endow.*

Li, H.; Zhang, J.; Li, C.; and Chen, H. 2023a. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *AAAI*.

Li, T.; Ma, X.; Zhuang, A.; Gu, Y.; Su, Y.; and Chen, W. 2023b. Few-shot In-context Learning for Knowledge Base Question Answering. *CoRR*, abs/2305.01750.

Liang, C.; Norouzi, M.; Berant, J.; Le, Q. V.; and Lao, N. 2018. Memory Augmented Policy Optimization for Program Synthesis and Semantic Parsing. In *NeurIPS*, 10015–10027.

Liu, Q.; Chen, B.; Guo, J.; Ziyadi, M.; Lin, Z.; Chen, W.; and Lou, J. 2022. TAPEX: Table Pre-training via Learning a Neural SQL Executor. In *ICLR*. OpenReview.net.

Liu, Y.; Liang, D.; Fang, F.; Wang, S.; Wu, W.; and Jiang, R. 2023a. Time-Aware Multiway Adaptive Fusion Network for Temporal Knowledge Graph Question Answering. In *ICASSP*, 1–5. IEEE.

Liu, Y.; Liang, D.; Li, M.; Giunchiglia, F.; Li, X.; Wang, S.; Wu, W.; Huang, L.; Feng, X.; and Guan, R. 2023b. Local and Global: Temporal Question Answering via Information Fusion. In *IJCAI*, 5141–5149. ijcai.org.

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

Luo, H.; E, H.; Tang, Z.; Peng, S.; Guo, Y.; Zhang, W.; Ma, C.; Dong, G.; Song, M.; and Lin, W. 2023. ChatKBQA: A Generate-then-Retrieve Framework for Knowledge Base Question Answering with Fine-tuned Large Language Models. *CoRR*, abs/2310.08975.

Luo, Y.; Wang, W.; Lin, X.; Zhou, X.; Wang, J.; and Li, K. 2011. SPARK2: Top-k Keyword Query in Relational Databases. *IEEE Trans. Knowl. Data Eng.*, 23(12): 1763–1780.

Mavromatis, C.; Subramanyam, P. L.; Ioannidis, V. N.; Adeshina, A.; Howard, P. R.; Grinberg, T.; Hakim, N.; and

Karypis, G. 2022. TempoQR: Temporal Question Reasoning over Knowledge Graphs. In *AAAI*.

Miller, A. H.; Fisch, A.; Dodge, J.; Karimi, A.; Bordes, A.; and Weston, J. 2016. Key-Value Memory Networks for Directly Reading Documents. In *EMNLP*.

Omar, R.; Dhall, I.; Kalnis, P.; and Mansour, E. 2023. A Universal Question-Answering Platform for Knowledge Graphs. *Proc. ACM Manag. Data*, 1(1): 57:1–57:25.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C. L.; Mishkin, P.; Zhang, C.; and et al. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Pasupat, P.; and Liang, P. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *ACL (1)*.

Pourreza, M.; and Rafiei, D. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *CoRR*.

Qi, J.; Tang, J.; He, Z.; Wan, X.; Cheng, Y.; Zhou, C.; Wang, X.; Zhang, Q.; and Lin, Z. 2022. RASAT: Integrating Relational Structures into Pretrained Seq2Seq Model for Text-to-SQL. In *EMNLP*.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP/IJCNLP (1)*, 3980–3990. Association for Computational Linguistics.

Sakor, A.; Mulang', I. O.; Singh, K.; Shekarpour, S.; Vidal, M.; Lehmann, J.; and Auer, S. 2019. Old is Gold: Linguistic Driven Approach for Entity and Relation Linking of Short Text. In *NAACL-HLT (1)*.

Saxena, A.; Chakrabarti, S.; and Talukdar, P. P. 2021. Question Answering Over Temporal Knowledge Graphs. In *ACL/IJCNLP (1)*.

Saxena, A.; Tripathi, A.; and Talukdar, P. P. 2020. Improving Multi-hop Question Answering over Knowledge Graphs using Knowledge Base Embeddings. In *ACL*.

Shang, C.; Wang, G.; Qi, P.; and Huang, J. 2022. Improving Time Sensitivity for Question Answering over Temporal Knowledge Graphs. In *ACL (1)*.

Spitkovsky, V. I.; and Chang, A. X. 2012. A Cross-Lingual Dictionary for English Wikipedia Concepts. In *LREC*.

Sun, H.; Dhingra, B.; Zaheer, M.; Mazaitis, K.; Salakhutdinov, R.; and Cohen, W. W. 2018. Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text. In *EMNLP*.

Wang, B.; Shin, R.; Liu, X.; Polozov, O.; and Richardson, M. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *ACL*.

Wang, C.; Cheung, A.; and Bodík, R. 2017. Synthesizing highly expressive SQL queries from input-output examples. In *PLDI*.

Wu, L.; Petroni, F.; Josifoski, M.; Riedel, S.; and Zettlemoyer, L. 2020. Scalable Zero-shot Entity Linking with Dense Entity Retrieval. In *EMNLP (1)*.

Xie, J.; Zhang, K.; Chen, J.; Lou, R.; and Su, Y. 2023. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. In *The Twelfth International Conference on Learning Representations*.

Xie, T.; Wu, C. H.; Shi, P.; Zhong, R.; Scholak, T.; Yasunaga, M.; Wu, C.; and et al. 2022. UnifiedSKG: Unifying and Multi-Tasking Structured Knowledge Grounding with Text-to-Text Language Models. In *EMNLP*.

Yaghmazadeh, N.; Wang, Y.; Dillig, I.; and Dillig, T. 2017. SQLizer: query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA): 63:1–63:26.

Ye, Y.; Hui, B.; Yang, M.; Li, B.; Huang, F.; and Li, Y. 2023. Large Language Models are Versatile Decomposers: Decomposing Evidence and Questions for Table-based Reasoning. In *SIGIR*.

Yih, W.; Richardson, M.; Meek, C.; Chang, M.; and Suh, J. 2016. The Value of Semantic Parse Labeling for Knowledge Base Question Answering. In *ACL (2)*.

Zeng, Z.; Lee, M.; and Ling, T. W. 2016. Answering Keyword Queries involving Aggregates and GROUPBY on Relational Databases. In *EDBT*, 161–172. OpenProceedings.org.

Zha, L.; Zhou, J.; Li, L.; Wang, R.; Huang, Q.; Yang, S.; Yuan, J.; Su, C.; Li, X.; Su, A.; and et al. 2023. TableGPT: Towards Unifying Tables, Nature Language and Commands into One GPT. *CoRR*, abs/2307.08674.

Zhang, L.; Zhang, J.; Ke, X.; Li, H.; Huang, X.; Shao, Z.; Cao, S.; and Lv, X. 2023a. A survey on complex factual question answering. *AI Open*, 4: 1–12.

Zhang, Y.; Dai, H.; Kozareva, Z.; Smola, A. J.; and Song, L. 2018. Variational Reasoning for Question Answering With Knowledge Graph. In *AAAI*, 6069–6076. AAAI Press.

Zhang, Y.; Li, Y.; Cui, L.; Cai, D.; Liu, L.; Fu, T.; Huang, X.; Zhao, E.; Zhang, Y.; Chen, Y.; Wang, L.; Luu, A. T.; Bi, W.; Shi, F.; and Shi, S. 2023b. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *CoRR*, abs/2309.01219.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.00103.