StructGPT：一种基于 LLM 的处理结构化数据的通用框架

1. 引言

1.1 结构化数据

1. 1.1 结构化数据的重要性及其应用

结构化数据通常是以严格格式组织的信息，这使得机器和算法能以高效和自动化的方式访问及处理这些数据。在商业智能、决策支持系统和科学研究等领域，结构化数据因其高度组织性和易于查询处理的特性，被广泛应用。例如，知识图谱以实体及其相互关系的三元组形式存储信息，而传统数据库则通过表和字段来组织数据。这些结构化形式提供了数据处理的标准化方法，从而支持复杂的查询和数据分析。

1.1.2 LLMs 在处理结构化数据时面临的挑战

尽管大型语言模型如 GPT 系列在自然语言处理领域取得显著成就，它们在处理结构化数据方面还存在一些挑战：

1.特殊数据格式的处理困难：结构化数据，如知识图谱和数据库，具有其特殊的组织格式，这对于未专门训练处理此类数据的 LLMs 来说较为复杂。模型在训练期间主要接触非结构化文本，缺乏对于结构化数据中的实体、关系及其连接方式的深入理解。

2.庞大数据量的处理限制：结构化数据的另一个挑战是其庞大的数据量。例如，Wiki Data 包含超过一亿条数据项，Freebase 拥有近 19 亿条三元组。相比之下，GPT-4 的令牌限制为 8192，这意味着无法在一次处理中涵盖所有相关数据。这限制了 LLMs 在不借助外部信息的情况下进行复杂推理的能力。

面对这些挑战，未来的研究需要开发新的模型架构和训练方法，以让 LLMs 能更有效地理解和处理结构化数据。这可能包括引入新的内部结构以更好地捕捉数据间的关系，或开发新的训练策略以提高从结构化数据中检索和推理的能力。通过这些创新，LLMs 可以更好地应对复杂数据环境，为数据驱动的决策和智能系统的未来发展提供支持。

2. 现有研究

在处理结构化数据方面，传统的大型语言模型（LLMs）如 GPT 通常面临理解和操作结构化数据格式的困难。现有研究主要集中在如何使这些模型能够更好地理解和生成结构化数据的查询和回答。

1. **特定模型架构：**早期的工作集中于为特定类型的结构化数据设计定制的模型架构，如图神经网络（GNNs）用于处理知识图谱，表格转换器（Table Transformers）用于处理数据表。这些方法虽然在特定任务上表现良好，但通常缺乏跨不同类型结构化数据的通用性，且难以迁移到其他任务。
2. **预训练语言模型的利用：**随着预训练语言模型（如 T5 和 BART）的成功，一些研究开始利用这些模型作为不同结构化数据任务的通用编码器或解决方案。例如，UnifiedSKG 项目通过将结构化数据任务统一到文本到文本的格式，利用 T5 进行微调来学习生成正确的答案。

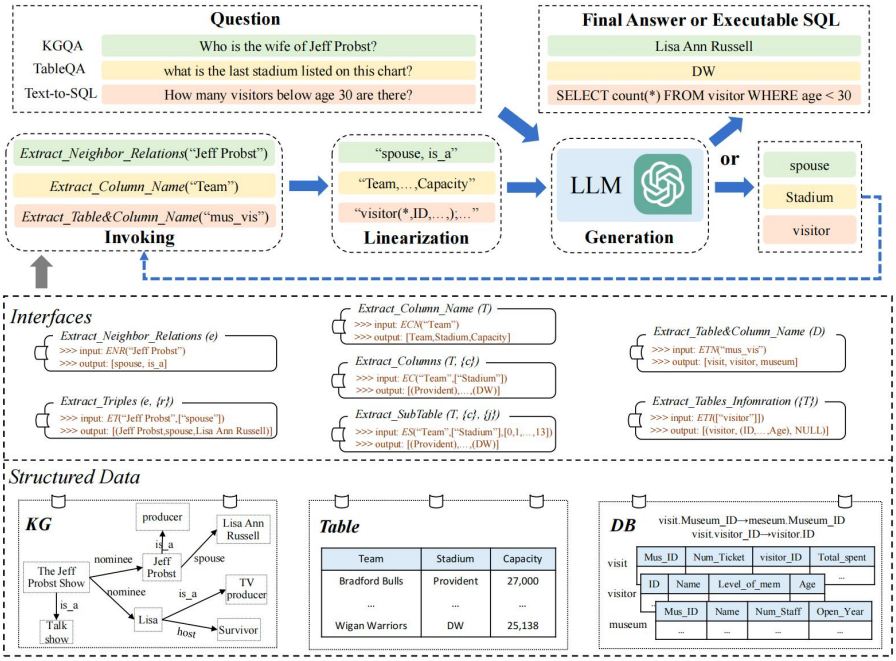
虽然这些方法在提升结构化数据处理方面取得了一些进展，但它们通常需要对模型参数进行调整，并且在处理大规模结构化数据时受到输入长度的限制。此外，这些方法往往专注于特定类型的结构化数据，缺乏处理多种数据类型的通用性。

3. 实验

3.1 实验原理

我们注意到，结构化数据组织良好，并支持通过形式化语言或查询（称为通用接口）进行轻松访问（比如说使用 SQL 与 SPARQL）。我们的基本思想是把 LLM 进行 Reading 和 Reasoning 的两个过程分开来，分别进行处理，先利用结构化数据的接口实现精确、高效的数据访问和过滤（获取相关证据），进而利用 LLMs 的推理能力找出问题的最终方案或结果（完成任务）。通过这种方式，LLMs 可以集中精力在推理过程中回答问题，而不需要考虑专门阅读结构化数据的方法。我们暂且可以称这个处理方式的框架叫做 IRR（Iterative Reading-then-Reasoning）框架。

在论文中设计了专门用于读取结构化数据的接口，并迭代执行 invoking-linearization-generation 过程，以利用大型语言模型（LLMs）在接口上进行推理，直到得出最终答案或可执行的 SQL 语句。



实验原理图（源自论文）

针对不同的结构化数据设计接口

在实验中，针对三种不同的结构化数据，我们设计了三种不同类型的接口用来提取这类数据类型的信息，这些接口的设计使得 LLMs 能够有效地处理各种结构化数据，而无需对每种数据类型进行特定的训练。

接下来我们将讲解 StructGPT 是如何利用这些接口来理解和阅读这些结构化的数据，然后对于用户的问题进行推理的。在这个过程中，StructGPT 一共需要三步，分别是：invoking（调用），linearization（信息的线性化）以及 generation（生成答案）。

3.1.1 Invoking

在这一步中，我们的目标是调用一个接口来从结构化数据中提取相关信息。根据前面设计的接口，我们可以根据当前可用的数据（例如，实体和表格）构建输入，然后调用接口以获得更详细的相关信息（例如，相邻关系和列名），这些信息将被输入到 LLM 中以收集有用的信息或生成答案。

3.1.2 Linearization

根据前面调用接口得到的从结构化数据提取的信息，我们可以将其转换为 LLM 能够理解的文本句子。

对于来自知识图谱（KG）的信息（即关系和三元组），我们将它们连接成一个由特定分隔和边界符号标记的长句子。

对于表格和数据库，我们利用相同的方法来线性化提取的表名或列名。而对于列和行中的内容，我们可以将它们转换为三元组。

3.1.3 Generation

在线性化之后，我们为 LLM 设计了两种类型的输入提示，以满足不同的目的：

第一种类型的提示主要采用以下模式："这里是[Y]。哪些[X]对于回答问题[Q]最相关。"它旨在引出 LLM 从线性化提取的信息（即[Y]）中选择有用证据（即[X]）的能力，根据问题（即[Q]）。

第二种类型的提示遵循以下模式："基于[Y]，请为问题[Q]生成[Z]。"它旨在预测给定问题（即

[Q]）的目标结果（即[Z]），基于线性化提取的信息（即[Y]）。请注意，目标结果可以是答案字符串或可执行的正式语言（例如，SQL），它可以引导到最终答案。

通过在设计的接口上迭代上述 invoking-linearization-generation 过程，LLM 可以逐步捕获更多有用的证据，以得出最终答案。

3.2 实验设计

本实验在结构化数据上进行了三个复杂推理任务的实验，即知识图问题回答（KGQA），表格问题回答（TableQA）和基于数据库的文本到 SQL。

3.2.1 数据集

知识图谱问答（KGQA）评估：通过使用一个基准数据集 WebQuestionsSP（WebQSP）。WebQSP 要求在 Freebase 知识图谱上进行最多 2 跳推理。在这个任务中，系统需要根据给定的问题，利用知识图谱中的实体和关系进行推理，以生成正确的答案。这种评估方法可以帮助衡量系统在理解和利用知识图谱上的推理能力方面的表现。

基于表格的问答（TableQA）评估：采用了三个广泛使用的数据集，包括弱监督的 WikiSQL、WikiTableQuestions（WTQ）和 TabFact。这些数据集在表格问答领域具有代表性。其中，WikiSQL 和 WTQ 是基于表格的问答数据集，而 TabFact 则专注于表格事实验证的多项选择数据集。在处理这些数据集时，面临不同的挑战和要求。例如，WikiSQL 需要对表格内容进行过滤和汇总以提取相关信息，而 WTQ 则要求更高级的推理能力，例如排序。而 TabFact 则需要实验判断提供的声明是否与表格中储存的事实一致。

基于数据库的语义解析（Text-to-SQL）评估：本实验采用了三个公共数据集，分别是 Spider、Spider-SYN 和 Spider-Realistic。Spider 是一个典型的 Text-to-SQL 数据集，包含了 20 个数据库和 1034 个评估样本。为了进一步提高挑战性，实验从 Spider 衍生出了 Spider-SYN 和 Spider-Realistic 这两个数据集。在 Spider-SYN 中，实验手动替换了自然语言问题中的同义词，增加了对系统的理解和泛化能力的要求。而 Spider-Realistic 则移除了评估集中明确提到所需列名的问题，增加了对系统的灵活性和适应能力的考验。

3.2.2 评估标准

知识图谱问答（KGQA）评估方法：采用了 Hits@1 指标来检验预测的第一答案是否正确。重点在于生成最有信心的答案，然后检查预测是否命中目标。

基于表格的问答（TableQA）评估方法：采用了两种评估指标：指称准确性和准确性。在 WTQ 和 WikiSQL 中，实验使用指称准确性来评估预测答案与金标准答案是否基于集合级别等价。而在 TabFact 中，实验采用准确性来评估预测的正确性。

基于数据库的语义解析（Text-to-SQL）评估方法：实验采用执行准确性（EX）来评估预测的 SQL 和金标准 SQL 的执行结果是否相同。

3.2.3 基准

实验采用了一个通用的迭代阅读然后推理（IRR）框架作为方法，适用于不同的大型语言模型（LLMs）。在本实验中，使用了 ChatGPT。
在知识图谱问答（KGQA）方面，实验选择了一些基线模型，包括 KV-Mem、GragtNet、EmbedKGQA、NSM 和 UniKGQA 等。

在基于表格的问答（TableQA）任务中，实验选择了 MAPO、TAPAS、UnifiedSKG（T5-3B）、TAPEX 和 DATER 等基线模型。

而在基于数据库的语义解析（Text-to-SQL）任务中，实验选择了 RATSQL+BERTLarge、TKKLarge、T5-3B+PICARD、RASAT+PICARD 和 RESDSQL-3B+NatSQL 等基线模型。
此外，实验还整合了直接使用 ChatGPT 来实现上述任务的基线，并在零样本设置中进行了评估。为了确保公平比较，实验使用相同的指令提示来评估这些模型，确保与实验方法的唯一区别是结构化数据的使用。

3.3 代码分析

我们对论文对应的仓库中的代码进行了研究与分析，并运行实践进行了实验。在实验中涉及以下三个代码文件，用来实现论文中提及的 StructGPT 框架：
每个脚本中都有一个 ChatGPT 类，它负责与 OpenAI 的 GPT 模型交互，发送提示并获取响应。
Retriever 类用于处理结构化数据，如数据库表格和知识图谱实体，为 LLMs 提供必要的信息以进行推理。

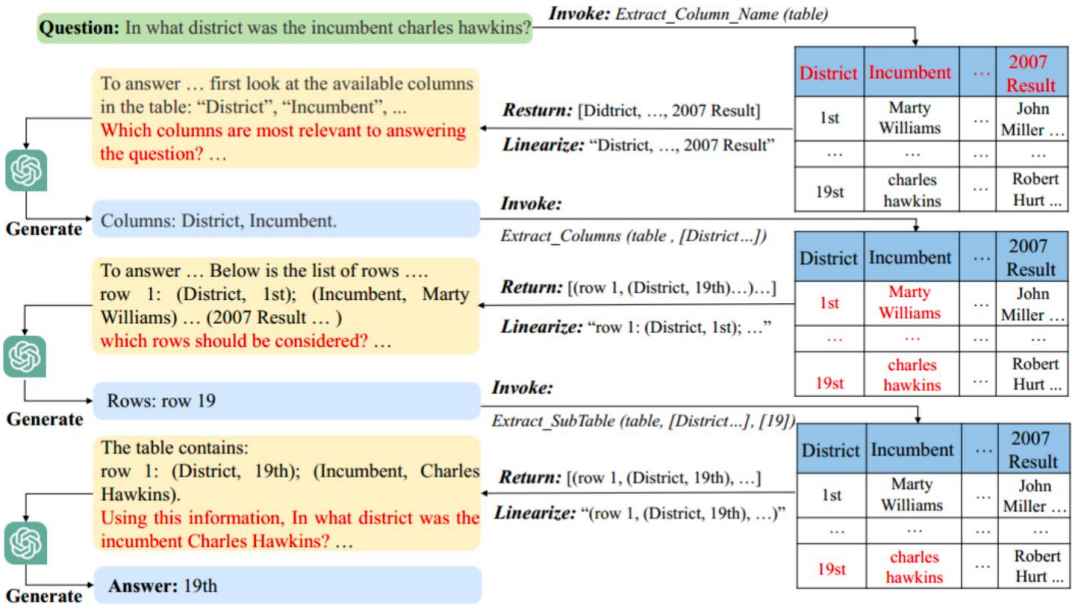
Solver 类结合了 ChatGPT 和 Retriever 的功能，管理整个问答过程，包括迭代调用 LLM、解析模型响应、应用接口和处理错误案例。

项目代码文件

接下来以 structgpt_for_tableqa.py 文件为例进行具体的分析。structgpt_for_tableqa.py 文件实现了一个针对表格问答（TableQA）任务的系统，包含以下类：

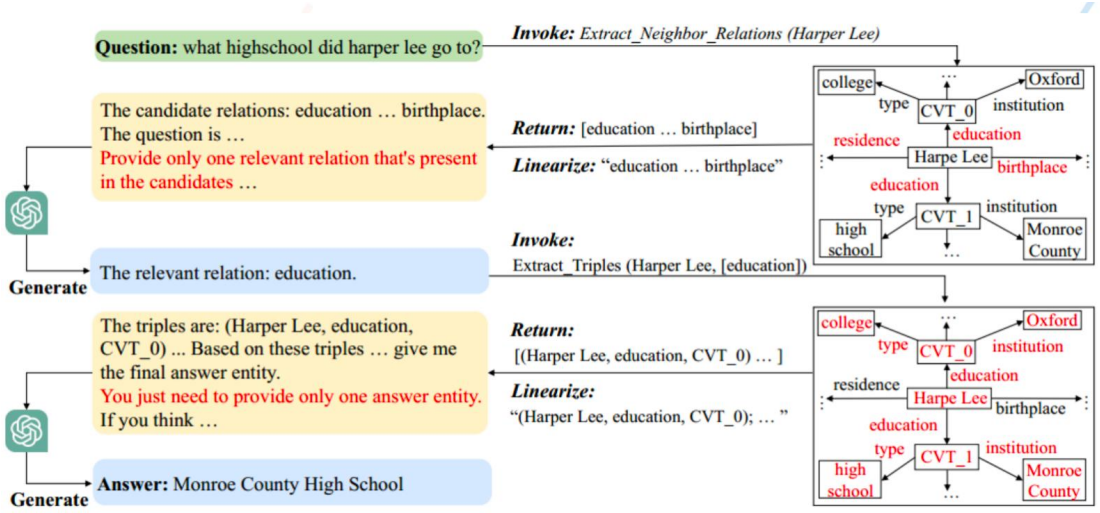
对代码中类的分析

下面这张图展示了一个基于 structgpt_for_tableqa.py 脚本的表格问答系统的工作流程。在这个流程中，系统首先通过调用 Extract_Column_Name 接口来识别表格中可用的列，例如“District”和“Incumbent”，这些列被认为与回答“查尔斯·霍金斯是哪个选区的现任者”这一问题最为相关。接着，系统将问题和相关列名转换为 LLM 可以理解的格式，并利用 LLM 生成可能包含答案的列内容。通过进一步分析表格内容，系统识别出特定的行（例如“19th”），并使用 Extract_SubTable 接口提取这些行的详细信息。最终，系统根据提取的信息，使用 LLM 生成并输出答案“19th”，即查尔斯·霍金斯所在的选区。整个流程是一个迭代的过程，涉及信息的提取、转换和生成，以确保从结构化表格数据中准确回答问题。

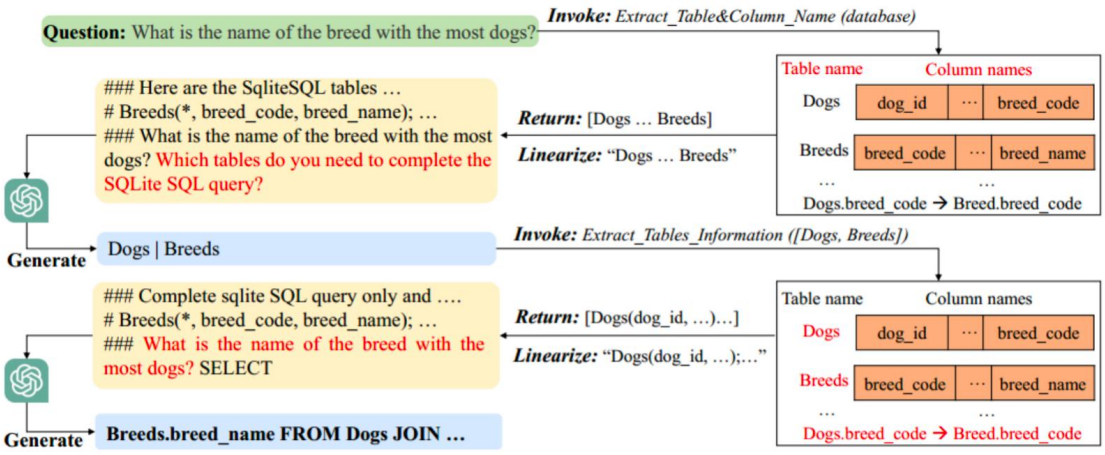


structgpt_for_tableqa.py 文件执行原理图

同时，还有一些关于其他脚本文件的运行原理图：



structgpt_for_webqsp.py 文件执行原理图



structgpt_for_text_to_sql.py 文件执行原理图

3.4 运行结果

在命令行中运行测试脚本，我们可以在不同的数据集上对 StructGPT 的准确性进行实验。我们可以得到以下结果：

DB 类型

```
                    easy                medium
count               248                 446
==================   EXECUTION ACCURACY     ==================
execution           0.887               0.809
```

Spider dataset

```
                    easy                medium
count               109                 203
==================   EXECUTION ACCURACY     ==================
execution           0.817               0.773
```

Spider_Realistic dataset

```
                    easy                medium
count               248                 440
==================   EXECUTION ACCURACY     ==================
execution           0.730               0.689
```

Spider_SYN dataset

Table 类型

```
Acc: 0.8674
Totally 1694 bad cases need further solved.
Right count: 11085, Error count: 1694(Max len count:
```

TabFact dataset

```
Denotation Acc: 0.4820
Totally 2250 bad cases need further solved.
Right count: 2094, Error count: 2250(Max len count: 93)
```

WTQ dataset

```
Totally 15878 test data
Totally 15878 prediction data
Denotation Acc: 0.5450
Totally 7224 bad cases need further solved.
Right count: 8654, Error count: 7224(Max len count: 0)
```

WikiSQL

KG 类型

```
Hits@1: 0.7328
Totally 435 bad cases need further solved.
Right:1193, Wrong:390, Max_len:45
```

webqsp

3.5 实验结论

1. KGQA 实验分析

方法	KV-Mem	GraftNet	EmbedKGQA	NSM	UniKGQA	ChatGPT	ChatGPT + IRR (ours)
WQSP	46.7	66.4	66.6	68.7	75.1	61.2	73.3

大型语言模型（LLMs）在 WebQSP 数据集上的表现与监督学习模型相当（例如，ChatGPT 的 61.2 对比 GraftNet 的 66.4，Davinci-003 的 48.3 对比 KV-Mem 的 46.7），这是在不使用知识图谱（KGs）的零样本设置中实现的。这表明 LLMs 确实掌握了一定量的知识，可以帮助它们回答复杂问题。然而，在需要多跳推理的更难数据集上（例如，MetaQA-2hop 和 MetaQA-3hop），两个 LLMs 的表现不佳。这表明 LLMs 不能仅依赖自己的知识来回答困难问题，它们需要与 KGs 结合，而加上我们的方法后，ChatGPT 的表现有显著提高，甚至能媲美最高基准。

2. TableQA 实验分析

方法	WTQ	WikiSQL	TabFact
MAPO	43.8	72.6	-
TAPAS	48.8	83.6	81.0
UnifiedSKG (T5-3B)	49.3	86.0	83.7
TAPEX	57.5	89.5	84.2
DATER	65.9	-	93.0
ChatGPT	43.3	51.6	82.9
+ IRR (ours)	48.2	54.5	86.7

在三个 TableQA 数据集上的评估显示，首先，使用完整表格作为提示，ChatGPT 在 WTQ 和 TabFact 上的表现可以与全数据监督调整方法相媲美，但在更难的 WikiSQL 数据集上表现不佳。这表明大型语言模型（LLMs）在一定程度上能够理解表格数据中的知识。其次，我们提出的方法可以在这三个数据集上持续显著提高 LLMs 的性能。实验的方法为 LLMs 提供了一种更有效的方式，使其能够迭代地访问和利用表格中的相关信息，从而减少了无关和冗余信息的影响
。

3. Text-to-SQL 实验分析

方法	Spider	Spider-	SYN
RAT-SQL + BERTLarge	72.3	-	62.1
TKK-Large	73.2	60.5	64.4
T5-3B + PICARD	79.3	69.8	71.4
RASAT + PICARD	80.5	70.7	71.9
RESDSQL-3B + NatSQL	84.1	76.9	81.9
ChatGPT	70.1	58.6	63.4
+ IRR (ours)	74.8	62.0	70.3

当使用数据库的所有信息（表名、列名和外键）作为提示时，大型语言模型（LLMs）能够直接生成适当的 SQL 查询语句，并在所有三个数据集上表现良好。然而，LLMs 的性能并不优于竞争性的全数据监督调整方法，这显示了这项任务的难度。由于实验提出的方法能够提取相关的表格和列，它也减轻了 LLMs 在生成 SQL 查询时受到无关信息的影响，LLMs 在三个数据集上的持续性能提升也表明了实验提出的方法的有效性。

4. 错误分析

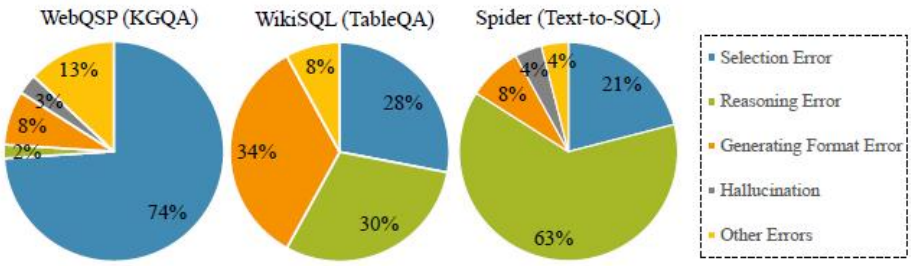


Figure 3: Proportions of different error types in three datasets over different types of structured data.

为了系统地分析实验方法的不足，实验首先选择了三个具有不同类型结构化数据的数据集（即 WebQSP、WTQ 和 Spider），并从每个数据集中随机抽取 100 个错误案例。然后，手动检查这些失败案例，并将它们分类为五个类别：

1. **选择错误**：LLM 没有选择相关信息。
2. **推理错误**：给定提取的相关信息，LLM 未能生成正确的答案或 SQL。
3. **生成格式错误**：生成的答案格式异常，无法被结果解析器识别。
4. **幻觉**：生成的结果与提取的信息不一致。
5. **其他错误**：其他无法分类的错误。

实验在上图中展示了统计数据。首先，对于这三个数据集，发生错误的分布是不同的。在 WikiSQL 中，生成格式、选择和推理错误的频率相对均匀。而在 WebQSP 中，选择错误是主要错误类型（74%），因为 KGQA 任务需要从数千个关系中选择最相关的一个，这不是一件容易的工作。在 Spider 中，推理错误更多（62%），因为 Text-to-SQL 任务要求 LLMs 生成一个可以执行以获得答案的 SQL，这对 LLMs 来说也很难。根据错误分布，改进主要错误案例以特别提高每个数据集的性能是有前景的。具体来说，可以设计更多高质量的提示，以促使 LLMs 在选择和推理 KGQA 和 Text-to-SQL 任务时仔细做出决策。此外，还考虑增加更多的接口和迭代次数，将复杂问题分解为多个简单问题，以简化复杂的推理任务，从而获得更好的性能。

4. 总结

4.1 主要贡献

StructGPT 的主要贡献可总结如下：

提出了结构化数据推理框架：StructGPT 框架能够扩展大语言模型（LLMs）的能力到结构化数据领域，如知识图谱、表格和数据库。通过迭代式阅读-推理（IRR）过程，使 LLMs 能够利用和理解结构化知识完成用户需求。

引入了 invoking-linearization-generation 过程：StructGPT 提出了一种通用的调用-线性化-生成求解过程，支持 LLMs 在结构化数据上进行推理。该过程通过调用特定的结构化数据接口，从数据源中收集相关证据，并将其线性化为文本提示，然后输入 LLMs 进行推理。这种迭代的过程使得 LLMs 能够逐渐接近目标答案。

设计了结构化数据接口：StructGPT 提供了针对知识图谱、表格和数据库的结构化数据接口设计。这些接口使得 LLMs 能够准确高效地访问和过滤结构化数据，从而获取相关证据用于推理过程。接口设计的可插拔性使用户可以根据自己的需求设计适应于不同场景的结构化数据访问接口。

StructGPT 提出了一套统一通用的推理框架，使得大语言模型能够在结构化数据上进行推理。通过设计结构化数据接口和引入调用-线性化-生成求解过程，实现了对结构化数据的有效利用。而在通过实验，我们也验证了在零样本和少样本场景下，StructGPT 显著提升了 Davinci-003 和 ChatGPT 的性能，并实现了与全量监督模型相当的性能，从而证明了 StructGPT 的有效性和通用性。

4.2 局限性

通过实验，我们也可以看到 StructGPT 仍存在一定的局限性，总结如下。

1.依赖于指令遵循能力较强的 LLM：StructGPT 模型使用的 ChatGPT 和 Davinci-003 两个 LLM 均具有较强的指令遵循能力，这可能会限制它在其他指令遵循能力较差的 LLM 上的应用。

2.仅基于结构化数据评估：StructGPT 只在结构化数据上评估了问答任务，缺乏对其方法普适性的更广泛评估。

3.生成文本的格式错误：在不同数据集的生成过程中难以控制 LLM 的答案格式，因此，正如在上文实验错误分析中提到的，在生成的文本中可能存在一些格式错误。

4.3 未来工作方向

针对目前的局限性与框架改善优化的需求，我们思考了未来工作的方向。

首先，我们需要探索更多 LLM 模型的适用性。除了 ChatGPT 和 Davinci-003，可以进一步研究和评估其他语言模型，尤其是那些在指令遵循能力方面表现较差的模型。通过在这些模型上进行实验和对比，更全面地了解 StructGPT 方法在不同 LLM 上的效果，并扩展其适用范围。

其次，我们需要拓展评估场景除了结构化数据上的进行问答任务，还可以扩展评估场景，例如数据生成文本、形式语言生成文本。通过对这些场景上的表现进行评估，可以更好地评估 StructGPT 的普适性和泛化能力，了解其在不同领域、不同数据类型和不同任务上的性能表现。

除此之外我们还需要改进生成文本的格式为了解决生成文本中存在的格式错误问题，可以通过精心设计提示、改进答案解析的方法，来改进LLM在不同数据集上的答案生成过程，提升其适应性和性能，确保生成的文本符合预期的格式和结构。

5. 参考文献

[1] Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. arXiv preprint arXiv:2305.09645. Available at https://arxiv.org/pdf/2305.09645.

[2] Wenhu Chen. 2023. Large language models are few(1)- shot table reasoners. In Findings of the Association for Computational Linguistics: EACL 2023,Dubrovnik, Croatia, May 2-6, 2023, pages 1090 – 1100. Association for Computational Linguistics.

[3] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-tosql. arXiv preprint arXiv:2205.06983.

[4] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023b. A comprehensive evaluation of chatgpt’s zero-shot text-to-sql capability. CoRR, abs/2303.13547.