[实验与分析] 评估语言模型中的 SQL 理解能力

Ananya Rahaman
University of Western Ontario
London, Ontario
arahaman@uwo.ca

Anny Zheng
University of Western Ontario
London, Ontario
azheng45@uwo.ca

Mostafa Milani
University of Western Ontario
London, Ontario
mostafa.milani@uwo.ca

Fei Chiang
McMaster Unviersity
Hamilton, Ontario
fchiang@mcmaster.ca

Rachel Pottinger
不列颠哥伦比亚大学
温哥华, 不列颠哥伦比亚省
rap@cs.ubc.ca

摘要

大语言模型（LLMs）的兴起对自然语言处理（NLP）和图像生成等多个领域产生了显著影响，使复杂的计算任务变得更加可及。尽管大模型展现出令人印象深刻的生成能力，但其“理解”程度，特别是在结构化领域如SQL方面，仍存在持续争议。在本文中，我们通过一系列关键的SQL任务来评估大模型对SQL的“理解”程度。这些任务包括语法错误检测、缺失token识别、查询性能预测、查询等价性检查以及查询解释，用以评估模型在识别、上下文感知、语义理解和连贯性方面的熟练程度——这些技能对于实现SQL理解至关重要。我们从知名的工作负载中生成标注数据集，并评估最新的大模型，重点关注查询复杂度和语法特征对性能的影响。我们的结果表明，虽然GPT4在需要识别和上下文感知的任务中表现优异，但所有模型在深层次的语义理解和连贯性方面仍存在困难，尤其是在查询等价性和性能估计任务中，揭示了当前大模型在实现完整SQL理解方面的局限性。

1 引言

大模型的兴起正在各个领域产生重大影响，使计算和数据科学任务变得更加可及和高效。例如，在自然语言处理和图像生成等领域，大模型能够生成类人文本和逼真图像。尽管大模型显然不具备与人类相同的“理解”水平，但其解决未直接训练过的问题的能力，暗示了一定程度的“理解”[15, 26, 31]。因此，对于大模型而言，“理解”指的是模型在不同情境下执行基本任务的能力，至少与人类一样熟练，甚至可能更优。

这种熟练程度可以通过一组特征性的技能来衡量，以评估理解能力。识别涉及识别目标对象/实体，例如在图像生成中识别并区分不同物体和场景。语义涉及识别意义的构建与解释方式，例如红色八边形的意义是停止，理解词语和短语的含义。上下文定义了语义被解释的范围和情景，例如在自然语言处理中，根据上下文解决歧义（当存在多种含义时），理解意图（理解说话者话语背后的意图，如识别讽刺或礼貌），以及处理分布外元素（识别某物体或场景是否不符合熟悉模式）。最后，连贯性识别对象之间的逻辑关联，例如物体连贯性确保在图像生成中物体相对于彼此处于合理位置，并识别共享相同意义的词语、句子和段落之间的逻辑联系。实现“理解”需要模型在这些技能上展现出（逐步提升的）熟练程度，并准确且有意义地完成特定任务操作。深入理解大语言模型的“理解”能力对于在真实应用中实现可靠性能至关重要，因为准确性在此类场景中尤为关键。

为了实现这一目标，我们研究了大模型在数据管理应用中的使用方法，特别是它们执行与SQL相关任务的能力。我们的研究不仅限于内容生成；还评估了那些展现出上述技能的具体SQL任务。我们提出这样一个问题：*大模型对SQL的“理解”程度如何？*

SQL Tasks. 我们提出了一系列核心SQL任务，旨在探测大模型对SQL“理解”程度。这些任务在许多与SQL相关的应用中非常重要，我们设计它们时从较简单的任务（如语法错误检测）逐步过渡到更复杂的任务（如查询等价性检查和解释）。从初学者到高级用户，人们需要完成的任务范围涵盖从语法错误识别到查询性能估算，再到查询等价性和解释。我们按难度递增的顺序评估大模型执行此类任务的能力，以反映技能熟练度的逐步提升。

语法错误识别。检测违反结构和语义要求的高级语法错误，与检测基本错误（例如缺少括号）相比，反映了不同程度的SQL“理解”能力。例如，检测属性、聚合函数在SELECT、GROUP BY、HAVING子句之间的

Skill	syntax error	missing token	Q.perf. estimate	Q.equiv.	Q.explain.
Recognition	✓	✓			
Semantics				✓	✓
Context		✓	✓		✓
Coherence	✓		✓	✓	

表 1: 技能到 SQL 的任务映射

错位，外部查询与内部查询之间属性类型不兼容，以及无效的连接操作，都需要对查询有复杂的“理解”。

缺失 token 的识别。 识别缺失的 token 是查询推荐等应用中的关键预处理步骤，其中缺失 token 的补全和查询自动补全功能至关重要 [14, 39]。我们评估模型不仅能够识别缺失的 token，还能精确定位其位置并判断其类型（例如，缺失的关键字（如 SELECT 或 WHERE）、表名、连接或条件中使用的别名，或字面量值）。

查询性能估算。 仅根据 SQL 查询文本准确估算其运行时性能具有挑战性，因为数据库模式、特定数据样本以及查询工作负载等多种因子均会产生影响 [39]。利用公开可用的查询工作负载，近期研究显示，包含多个连接和多个谓词条件的更复杂、更长的查询会带来更高的执行开销 [10, 16, 35]。我们评估大语言模型对查询复杂性的“理解”能力，用于性能估算，超越表面语法层面的分析。

查询等价性。 两个语法上不同的查询是等价的，如果它们在所有数据库实例上返回相同的结果。这对于查询最优化 [4, 13] 和查询推荐 [39] 至关重要，其中更简单的查询表示有助于更快的执行速度。我们使用标注的等价（正例）和非等价（负例）查询对来评估查询等价性。虽然生成等价对较为微妙，但负例情况需要仔细考虑。如果我们随机配对非等价查询并将其标记为非等价，则任务会变得过于简单，因为表面差异通常就能识别出非等价性，而无法测试模型理解深层查询语义的能力。

查询可解释性。 我们评估大模型通过描述查询输出来解释 SQL 查询的能力。该任务类似于代码和图像理解中的评估，分别用于生成代码文档 [22] 和图像标题，以衡量理解程度。我们对一系列复杂的查询进行了评估，包括多表查询、嵌套子查询以及复杂的逻辑条件。

Skill to task proficiency. 每个 SQL 任务都关联一组用于评估（SQL）理解能力的技能，如表 1所示。在语法错误识别中，我们评估模型对语法违规的识别与一致性，确保各子句之间的逻辑一致性，例如聚合函数与 GROUP BY 子句之间的不匹配问题。缺失 token 识别需要模型具备识别能力和上下文理解能力，以检测缺失的 token，并确定其正确的类型和位置（例如关键字、表名或值）。查询性能估计评估模型的上下文理解与一致性，要求模型考虑查询复杂度、数据库模式以及工作负载来估算性能。在查询等价性判断中，我们评估模型的语义理解和一致性能力，使其能够解释不同语法格式的查询但表达相同的功能输出。最后，在查询解释任务中，我们评估模型的语义理解与上下文理解能力，以描述查询的目的及其结果，结合数据库模式和数据上下文进行说明。

Paper Contributions. 我们提出了一项实验研究，评估主流大模型在核心 SQL 任务上的表现。

SQL 任务驱动的数据基准。 这些任务中的许多都需要标注数据，我们通过修改来自流行 SQL 工作负载的原始查询来生成这些数据，例如斯隆数字巡天（SDSS）[35]、SQLShare [10] 和连接顺序 [16]。对于语法错误和缺失 token 识别任务，我们通过从工作负载中随机选择查询并注入错误或删除 token，创建半合成数据集。针对每个任务，我们选择合适的类型，例如要注入的语法错误类型或缺失 token 的类型（如关键字、表名、列名）。对于查询性能任务，我们依赖于 SDSS 工作负载，该工作负载包含过去查询评估的日志信息。我们根据查询的运行时间对其进行分类，高运行时间代表计算开销大的查询。对于查询等价性任务，我们手动修改选定的查询以生成等价和非等价的配对，确保修改反映现实中的查询变换，例如使用连接重写嵌套查询。我们的 SQL 任务驱动数据基准已公开。[1]

Prompt-to-SQL 任务性能。 提示词调优是确保大模型结果一致性的关键。我们尝试了多种提示词，在使用标注数据子集进行小规模试验的基础上，测试它们以确定每个任务的最佳提示词。然而，与大模型的交互远不止提示词设计。处理其响应的过程十分复杂，在我们的工作中，我们通过结合自动化脚本和人工检查来提取标签，从而解决这一问题。

SQL 任务评估框架。 我们系统地评估影响大模型在 SQL 任务中表现的因子。我们的评估框架考虑了三个关键维度。首先，我们在不同大模型之间比较 SQL 任务的表现。其次，我们分析查询工作负载的特性，特别是 SQL 查询的语法复杂度，例如表的数量、条件数量、嵌套子查询数量以及查询的整体长度。我们研究这些语法特性如何影响大模型处理和理解查询的能力。最后，我们评估特定 SQL 任务在不同参数下的表现，例如不同类型缺失 token 或查询变换对大模型性能的影响，以及某些形式的查询等价性或错误检测是否更难以识别。通过综合考虑这三个维度：大模型性能比较、工作负载特性以及任务类型，我们旨在全面评估影响大模型在 SQL 任务中表现的因素。

广泛的对比评估。 我们的实验表明，GPT4 在大多数任务中表现最佳，而其他模型在第二名的位置上并无一致表现。尽管大多数模型在二分类任务（如识别语法错误或缺失 token）中表现出较强性能，但所有大模型在多分类任务（如识别缺失 token 或语法错误的类型）中均面临挑战，准确率下降。大模型通常在处

[1] https://github.com/AnanyaRahaman/LLMs_SQL_Understading

理更长、更复杂的查询时表现不佳，尤其是在涉及逻辑推理或数值计算的任务中，这与先前的研究结果一致 [5, 8, 33]。

实验结果表明，大模型在需要识别和上下文理解的任务中表现良好，例如语法错误检测和缺失 token 识别。然而，在需要连贯性和语义理解的任务中，如查询等价性判断和性能估算，模型表现出局限性。这说明尽管大模型在表面层次的“理解”上表现出色，但在深入理解 SQL 查询中的语义关系和逻辑连贯性方面仍存在困难，凸显了进一步改进的必要性。

论文结构. 在第 2 节中，我们描述了研究中使用的负载，包括我们数据集中查询语法属性的综合统计量。第 3 节概述了实验设置，涵盖了 SQL 任务、从负载生成数据、大模型（LLMs）以及我们与大模型的交互方式，例如提示调优。第 4 节展示了实验结果与分析，第 5 节回顾了相关工作，第 6 节总结了全文并讨论了未来工作的方向。

2 查询工作负载

查询工作负载，或简称工作负载，是一组针对数据库执行的 SQL 查询，用于模拟真实世界的使用模式，以进行性能评估和最优化。我们对实验研究中使用的四个工作负载进行了概述和详细分析。

The Sloan Digital Sky Survey (SDSS) dataset [35]. SDSS 由一个关系型数据库组成，该数据库包含来自一项重大天文调查的数据，提供天空的详细图像和光谱，并包含用于与 SDSS 数据库交互的 SQL 查询工作负载。SDSS 工作负载的特点是其复杂性以及对精确天文学数据检索的需求。该工作负载在过去二十年中持续收集，包含了数百万条查询。在我们的研究中，我们使用了 2023 年记录的查询。

SQLShare [10]. SQLShare 是一个开源数据平台，旨在使数据共享和查询更加便捷。SQLShare 工作负载包含一系列用户生成的 SQL 查询，涵盖从简单数据检索到复杂数据操作的各种任务。与我们其他工作负载不同，SQLShare 包含针对多个具有不同模式的数据库的查询声明。

Join-Order [16]. Join-Order 基准是一种合成工作负载，旨在评估数据库系统在优化连接查询方面的性能。该基准包含复杂的 SQL 查询，用以测试优化器寻找高效连接顺序的能力。

Spider [36]. Spider 是一个大规模、复杂且跨领域的文本转 SQL 基准，用于评估模型的自然语言理解能力以及 SQL 生成能力。它包含多种数据库，以评估模型在不同数据库模式之间的泛化能力。Spider 在自然语言处理领域被广泛用于基准测试模型性能，即把自然语言查询转换为 SQL。我们仅使用 Spider 数据集进行查询解释任务，而其余三个工作负载则用于其他任务。

Workload	Number of Queries		Query Type		Aggregate		NestLvl	
	Original	Sampled	SELECT	CREATE	Yes	No	0	1
SDSS	5,081,188	285	Fig 1a		21	264	Fig 1e	
SQLShare	9,623	250	Fig 2a		59	192	Fig 2e	
Join-Order	157	157	113	44	119	38	-	-
Spider	4, 486	200	200	0	96	104	185	15

表 2: 工作负载统计概览

我们所有的任务负载（除 Join-Order 外）都包含大量查询；因此，在研究中使用全部查询是不现实的。为此，我们通过采样有限数量的查询创建了更小的数据集。表 2 显示了任务负载中的“原始”查询数量，“采样”则表示我们在实验中使用的采样查询数量。数据集生成过程详见第 3.2 节。接下来，我们分析采样查询的语法特性，以提供上下文来解释我们的实验结果。从今以后，我们将使用 SDSS、SQLShare、Join-Order 和 Spider 来指代从原始任务负载的采样查询中生成的数据集。

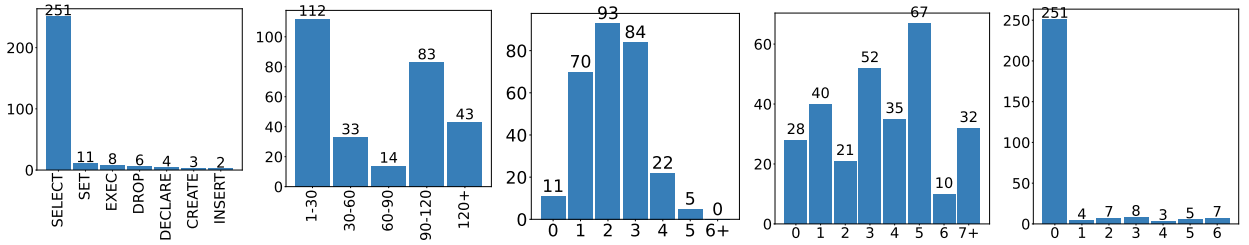
2.1 SQL 查询的语法特性

对于每个 SQL 查询，我们评估以下属性：

- char_count 和 word_count 分别指查询中的字符数和词数。
- query_type 指查询的类型，例如 SELECT、UPDATE 和 CREATE。
- table_count 和 join_count 分别表示查询中引用的不同表的数量以及总的连接操作数。连接包括显式连接（使用如 INNER JOIN 等连接关键字）和隐式连接（在 FROM 子句中的表与连接条件）。
- column_count 指的是查询的 SELECT 子句中使用或引用的不同列的数量。
- function_count 指查询中函数的总数，包括内置函数（如 min、avg）和用户定义函数。predicate_count 是 WHERE 子句中指定条件的数量。
- nestedness 表示查询中子查询的嵌套深度。
- aggregate 指的是查询是否使用了聚合函数。

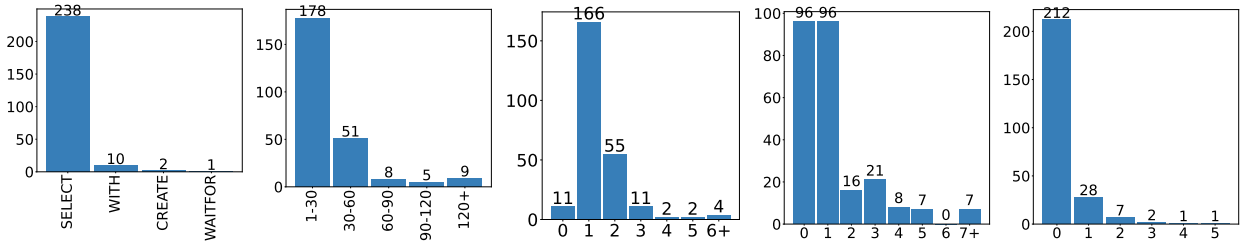
表 2 提供了所有四个工作负载的统计概览，包括 SELECT 和 CREATE 查询的数量，以及聚合查询与简单查询的细分。图 1-3 展示了其他属性。每个图都是直方图，显示在 y -轴上的查询数量和在 x -轴上的查询属性，其中 x -值表示属性的值域。例如，图 1b 显示了不同查询长度（word_count）范围内的查询数量（y -轴）。这些图表表明，SDSS 和 SQLShare 包含更多复杂的查询，涉及多个表和更广泛的谓词类型。相比之下，Join-Order 的查询更为简单且嵌套程度较低。就查询长度（word_count）而言，SDSS 和 Join-Order 的查询比 SQLShare 更长。

由于成对属性之间可能存在强相关性，导致冗余和低效，我们使用皮尔逊相关系数 [24] 检查成对查询属性之间的相关性，并采用 0.7 的阈值来表示强相关性。图 4展示了以下观察结果：



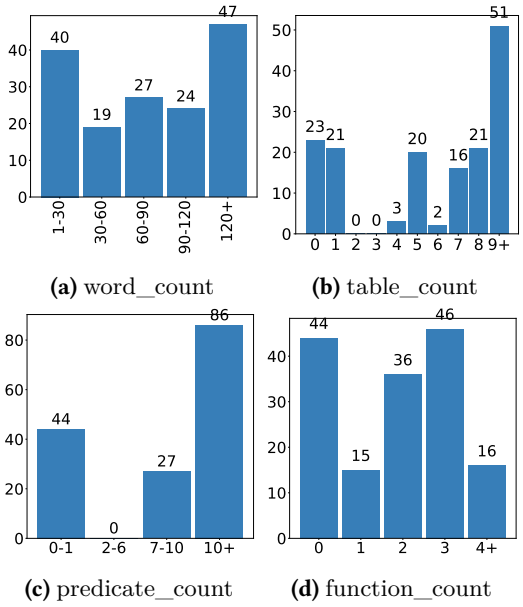
(a) query_type **(b)** word_count **(c)** table_count **(d)** predicate_count **(e)** nestedness

图 1: SDSS 统计量



(a) query_type **(b)** word_count **(c)** table_count **(d)** predicate_count **(e)** nestedness

图 2: SQLShare 统计量



(a) word_count **(b)** table_count **(c)** predicate_count **(d)** function_count

图 3: 连接顺序统计量

- char_count 和 word_count 之间具有高度相关性，因为较长的查询通常包含更多单词。
- table_count 和 join_count 也具有高度相关性，因为涉及更多表的查询通常包含更多的连接操作，这是多表 SQL 查询中的常见模式。

我们考虑特定工作负载独有的相关系数：

- 在 SDSS 中，column_count 和 char_count 之间存在强相关性，因为较长的查询通常涉及选择更多列或添加更多条件。此外，nestedness 和 join_count 也存在相关性，因为嵌套较深的查询往往包含多个连接操作。
- SQLShare 和 Join-Order 在 function_count 与 predicate_count 之间表现出较高的相关系数，这是由于在条件中频繁使用函数所致。
- 在 Join-Order、char_count 和 word_count 中，与 table_count 和 join_count 存在相关性，表明较长的查询涉及更多的连接和表。

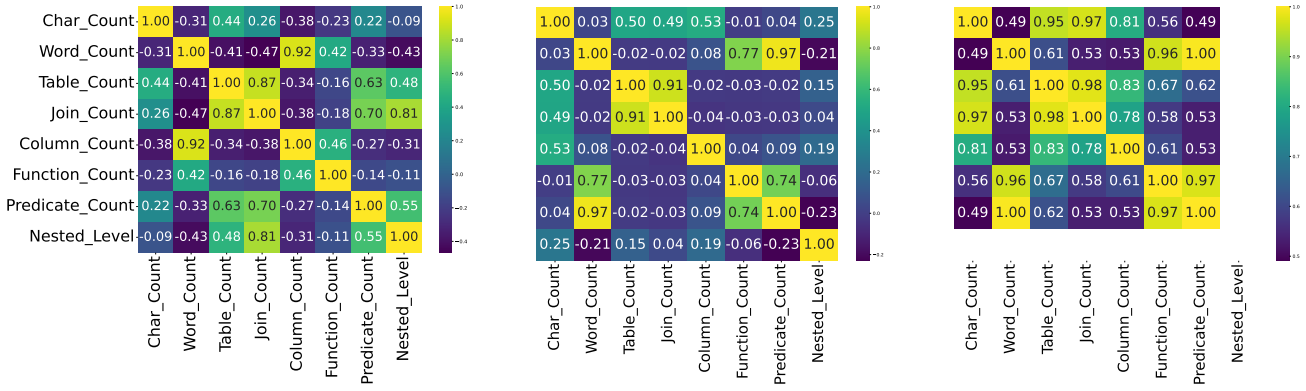
3 实验设置

我们在第 3.1 节介绍了我们的 SQL 任务，第 3.2 节描述了注入错误、缺失 token 以及推导等价与非等价查询的数据准备步骤。随后，我们在第 3.3 节概述了评估的大模型（LLMs），并在第 3.4 节说明了如何提示和响应这些大模型。

3.1 SQL 任务

3.1.1 **二元任务**. 我们首先从二分类任务开始，这些任务用于识别语法错误、缺失的 token 以及查询等价性。

syntax_error. 我们评估大语言模型识别语法错误存在与否的能力。我们研究了六种错误类型，具体描述如下。图 1 展示了每种类型的样本错误。



(a) SDSS **(b) SQLShare** **(c) Join-Order**

图 4: 每个工作负载下查询属性之间的成对相关性

- aggr-attr. 聚合函数在未对非聚合列进行正确分组的情况下被使用。
- aggr-having. 错误地使用 HAVING 子句来过滤非聚合列，而不是使用 WHERE。
- nested-mismatch. 嵌套查询中的内层查询返回多行数据，而外层查询未能正确处理这种情况。
- condition-mismatch. 数据类型不兼容的操作，例如将数值列与字符串进行比较。
- alias-undefined. 在查询中使用了别名，但该别名未被定义。
- alias-ambiguous. 同一列出现在多个表中，但在查询中的使用未指定表引用。

```
-- Q1: Aggregation without GROUP BY (aggr-attr)
SELECT plate,mjd,COUNT(*), AVG(z)
FROM SpecObj WHERE z > 0.5;
-- Q2: Incorrect Use of HAVING (aggr-having)
SELECT plate,COUNT(*) AS NumSpectra
FROM SpecObj GROUP BY plate HAVING z > 0.5;
-- Q3: Type mismatch in subquery (nested-mismatch)
SELECT p.ra,p.dec,s.z
FROM PhotoObj AS p JOIN SpecObj AS s
ON s.bestobjid = (SELECT bestobjid FROM SpecObj);
-- Q4: Type mismatch in condition (condition-mismatch
    )
SELECT plate,mjd,fiberid FROM SpecObj WHERE z = 'high
    ';
-- Q5: Undefined alias (alias-undefined)
SELECT s.plate,s.mjd,z
FROM SpecObj AS s JOIN PhotoObj AS p
ON s.bestobjid = photoobj.bestobjid;
-- Q6: Ambiguous alias (alias-ambiguous)
SELECT plate,fid FROM SpecObj AS s JOIN PhotoObj AS p
ON s.bestobjid = p.bestobjid WHERE bestobjid > 1000;
```

Listing 1: SQL syntax error examples

miss_token. 我们对大语言模型进行探测，以确定一个 SQL 查询是否缺少任何 token。尽管这可以被视为一种 syntax_error，但由于其在应用中的重要性，我们将其单独考虑。我们考虑六种类型的 token：keyword、table、column、value、alias 和 predicate（比较）。

query_equiv. 判断两个 SQL 查询是否等价，即它们是否具有相同的模式并产生相同的结果。我们研究了十种等价类型和八种非等价类型。清单 2 展示了使用 SDSS 工作负载的若干示例。关于完整的等价与非等价类型列表，以及详细的解释和示例，读者可参考我们的 GitHub 仓库。

- swap-subqueries。在嵌套查询中交换内层子查询和外层子查询。
- join-nested。将连接转换为子查询，或反之。
- cte. 使用公用表表达式（CTEs）重写查询，CTE 是使用 WITH 定义的临时结果集，可简化复杂查询，并在主查询中被引用。
- reorder-conditions。重新排列 WHERE 子句中条件的顺序。

我们研究四种非等价变换：

- agg-function。修改聚合函数，例如从 AVG 更改为 SUM。
- change-join-condition。修改连接类型，例如从 INNER JOIN 切换到 LEFT JOIN。
- 逻辑条件。修改逻辑运算符，例如将 AND 改为 OR。
- value-change. 更新筛选条件，例如更改比较值。

```
-- Q7: swap-subqueries (Equivalent)
SELECT s.plate,s.mjd FROM SpecObj AS s WHERE s.plate
    IN
    (SELECT p.plate FROM PhotoObj AS p WHERE p.ra >
        180);
-- Equivalent Query:
SELECT p.plate,p.mjd FROM PhotoObj AS p
WHERE p.ra > 180 AND p.plate IN
    (SELECT s.plate FROM SpecObj AS s);
-- Q8: join-nested (Equivalent)
SELECT s.fiberid FROM SpecObj AS s JOIN PhotoObj AS p
ON s.bestobjid = p.objid WHERE p.ra > 180;
```

```
-- Equivalent Query:
SELECT fiberid FROM SpecObj WHERE bestobjid IN
    (SELECT objid FROM PhotoObj WHERE ra > 180);
-- Q9: cte (Equivalent)
SELECT plate,mjd FROM SpecObj WHERE z > 0.5;
-- Equivalent Query:
WITH HighRedshift AS
    (SELECT plate,mjd FROM SpecObj WHERE z > 0.5)
SELECT plate,mjd FROM HighRedshift;
-- Q10: reorder-conditions (Equivalent)
SELECT * FROM SpecObj WHERE plate = 1000 AND mjd >
    55000;
-- Equivalent Query:
SELECT * FROM SpecObj WHERE mjd > 55000 AND plate =
    1000;
-- Q11: agg-function (Non-Equivalent)
SELECT plate,AVG(z) FROM SpecObj GROUP BY plate;
-- Non-Equivalent Query:
SELECT plate,SUM(z) FROM SpecObj GROUP BY plate;
-- Q12: change-join-condition (Non-Equivalent)
SELECT s.plate,s.mjd FROM SpecObj AS s
JOIN PhotoObj AS p ON s.bestobjid = p.objid;
-- Non-Equivalent Query:
SELECT s.plate,s.mjd FROM SpecObj AS s
LEFT JOIN PhotoObj AS p ON s.bestobjid = p.objid;
-- Q13: logical-conditions (Non-Equivalent)
SELECT plate,mjd,fiberid
FROM SpecObj WHERE z > 0.5 AND ra > 180;
-- Non-Equivalent Query:
SELECT plate,mjd,fiberid
FROM SpecObj WHERE z > 0.5 OR ra > 180;
-- Q14: value-change (Non-Equivalent)
SELECT plate, mjd, fiberid FROM SpecObj WHERE z >
    0.5;
-- Non-Equivalent Query:
SELECT plate,mjd,fiberid FROM SpecObj WHERE z > 5;
```

Listing 2: Examples of SQL equivalence and non-equivalence

performance_pred. 我们评估模型预测查询运行时间性能的能力。只有 SDSS 工作负载包含查询执行时间的真实值。图 5 清楚地展示了短运行（低代价）查询与长运行查询（高代价）之间的分离，我们将此视为一个二分类任务，并将高代价查询视为正类。

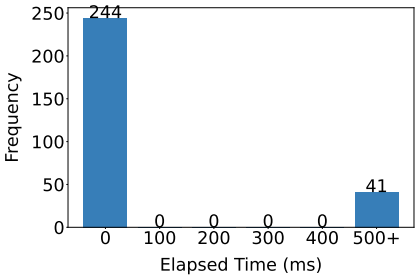


图 5: 采样 SDSS 查询的耗时

3.1.2 **多类别任务**. 我们通过探测大模型来指示语法错误的类型(syntax_error)、缺失 token 的类型(miss_token_type)以及查询等价性的类型（query_equiv_type），将二分类任务扩展为多分类任务。我们还评估了识别缺失 token 位置的任务（miss_token_loc）。

3.1.3 **查询说明**. 此任务（query_exp）解释了 SQL 查询的作用。它是文本到 SQL 任务的逆过程，现有的文本到 SQL 基准（如 WikiSQL [38]）提供了查询的自然语言描述。然而，这些基准中的许多查询相对简单，与 SDSS 和 SQLShare 中更复杂的负载相比。因此，我们选择了包含更复杂查询的 Spider 数据集 [36]，并进一步采样了更长、更复杂的查询。

我们的分析是定性的，而非定量的。我们手动审查大语言模型生成的解释，并将其与工作负载中提供的真实值描述进行对比。我们的目标是分析和讨论模型在何时以及为何无法提供准确、有意义的解释(第 4.5 节)。尽管此任务并不严格依赖现有的解释，但我们使用 Spider 的解释来辅助验证，并简化评估流程。

3.2 数据准备与标签生成

我们描述了如何为每个数据集生成特定任务的标签。

syntax_error 和 miss_token。我们通过从每个工作负载中随机选择查询并注入错误来生成半合成数据集。对于 syntax_error，我们随机选择要注入的错误类型，包括无错误（无错误）的情况。我们创建二元标签以指示是否存在错误（或不存在），以及用于多分类 (syntax_error_type) 任务的错误类型。对于 miss_token，我们采用类似的方法，即随机删除特定的 token 类型。这些任务使用 SDSS、SQLShare 和 Join-Order 工作负载。

query_equiv. 我们使用 SDSS、SQLShare 和 Join-Order 工作负载来生成等价和非等价对。我们随机选择等价类型和要修改的查询。生成非等价对具有挑战性，因为必须在查询之间保持足够的相似度（以使任务具有挑战性），同时包含功能上的差异。因此，每对查询都有一个标签（等价/非等价）和一种等价类型。

performance_pred. 对于此任务，我们仅使用了包含运行时数据的 SDSS。我们随机选取了 285 个查询，并分析其运行时间，将每个查询分类为高运行时间或低运行时间，以此作为计算开销的代理。运行时间超过 200 毫秒的查询被视为高成本，否则为低成本。我们选择此阈值是基于对图 5 的观察结果。

3.3 大型语言模型

我们使用了几种先进的大模型，简要描述如下。

GPT3.5. 2022 年底由 OpenAI 发布，GPT3.5 包含 1750 亿个参数，并在包含 Common Crawl、维基百科以及

各种书籍和学术文献的大规模语料库上进行训练。它旨在处理多种自然语言处理任务 [3]。

GPT4. OpenAI 的 GPT4 于 2023 年发布，拥有超过 2000 亿个参数，相较于其前代产品在上下文理解与推理能力方面有显著提升。它使用更大且更丰富的数据集进行训练，从而增强了其在各类语言任务中的表现 [23]。

Gemini. 由谷歌开发并于 2024 年推出，Gemini 在人工智能交互中优先考虑准确率与安全性，并高度重视伦理 AI。其参数量估计为 500 亿，基于庞大的多模态数据集进行训练，能够处理文本和视觉数据。其对多模态能力的重视以及与人类价值观的一致性，使其有别于主要针对文本任务设计的模型 [1]。

Llama3. Meta 的 Llama3 于 2023 年发布，其版本参数量最高可达 700 亿。该模型在来自通用数据集的数万亿 token 上进行训练，旨在实现高效性和可扩展性。LLaMA 专注于广泛的通用语言任务，使其非常适合部署在资源受限的环境中，在这些环境中保持性能 至关重要 [30]。

MistralAI. 2024 年推出，MistralAI 在参数量仅为 160 亿的情况下，针对高准确率进行了优化。其训练数据集范围广泛，特别注重领域特定内容和多语言能力。MistralAI 特别适用于需要深入理解 SQL 及其他结构化数据等领域的专业任务，在计算效率与领域目标性能之间取得了平衡 [21]。

大模型之所以能实现高性能，是因为它们在包含数百亿到数万亿 token 的数据集上进行了大规模训练。大模型发展的趋势是利用更大规模的数据集和更复杂的架构，以持续提升在多样化任务上的泛化能力 [3, 21, 23, 30]。

3.4 优化大语言模型的交互

与大模型交互需要仔细关注输入提示以及对其响应的处理。通过优化提示，我们可以引导模型生成更准确、更相关的内容。然而，响应结果也需要进行后处理，因为它们通常不会以我们任务所需的直接格式（例如简单的标签）提供。后处理包括从可能冗长或复杂的响应中提取必要信息，并确保其符合评估所需的特定格式。

Prompt Tuning. 为了引导大模型生成准确的回答，精心设计和优化输入提示在复杂任务中尤为重要，恰当的提示能够显著提升模型性能 [20, 27, 34]。在本研究中，提示调优对于有效处理 SQL 语法和语义的复杂性至关重要。我们的调优过程包含两个关键步骤：

(1) *提示生成与优化*。我们使用大模型生成了多种提示候选，然后手动进行优化，以确保其清晰性并符合我们的任务目标 [2, 32]。

(2) *模拟实验*。我们使用部分数据进行了模拟实验，以评估每个提示的有效性。从这些测试中选出表现最佳的提示，用于大规模实验。

按照这种方法，我们开发了一套针对特定任务的提示，以从模型中提取有意义的响应。这些提示根据每个实验任务进行了定制，复杂度各不相同，解决了语法错误检测、查询等价性和运行时估计等挑战：

- *语法错误和语法错误类型*。以下查询是否存在任何语法错误？如果存在，请解释错误原因。[查询]
- miss_token，miss_token_type，和 miss_token_loc。以下查询是否存在语法错误？(是/否) 如果存在，是否有缺失的词？（是/否）如果存在，缺失词的类型是什么？如果存在，缺失的词是什么？如果存在，缺失词的位置是？（请提供缺失词所在的位置的词数位置。）[query]
- query_equiv 和 query_equiv_type。以下两个查询是否等价（在相同的数据库模式下是否产生相同的结果）？如果等价，原因是什么？[query 1, query 2]
- performance_pred. 以下查询的运行时间是否比平常长？[query]
- query_exp. 提供一条描述此查询的声明：[query].

上述提示反映了我们提示调优方法的成果，该方法专门针对本研究中的 SQL 任务而设计。

Handling LLM Output. 尽管提供了明确的指令，仍有必要对大模型输出结果进行后处理。大模型常常产生冗长且🄵嗦的响应，需要仔细提取相关信息。例如，在上述预测任务中，虽然大多数大模型会以二元的“是”或“否”作答，但它们通常还会提供关于查询执行时间长短原因的解释。类似地，在 miss_token 任务中，响应格式并不总是符合我们的评估标准。为解决此问题，我们结合了人工处理和自动化脚本。人工处理涉及逐条阅读响应，以提取所需的具体信息，例如从解释中分离出“是”或“否”。为了加快这一过程，我们使用脚本检测常见的响应模式，并在响应具有可预测结构时自动提取相关内容。然而，对于更复杂或结构不清晰的输出，仍需人工干预以确保准确率。这使我们能够一致地格式化大模型的输出，并有效评估其性能。

Zero-Shot, Few-Shot, and Fine-Tuning. 零试学习指的是模型在未见过任何特定示例的情况下执行任务的能力，仅依赖其预训练知识。这种方法对于评估模型对某一领域的内在理解能力具有重要意义。在我们的实验中，我们仅专注于零试学习，以评估模型检测语法错误、评估查询等价性以及预测查询运行时成本的能力。我们的目标是研究模型的原始状态，不引入额外的任务特定信息，这反映了在实际场景中此类数据可能并不总是可用的情况。

少样本学习为模型提供少量示例以提升性能，而微调则在特定任务数据集上进一步训练模型以提高准

Case	Model	SDSS			SQLShare			Join-Order		
		Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Syntax Error	GPT4	**0.98**	**0.95**	**0.97**	0.94	**0.93**	**0.93**	**0.95**	0.91	**0.93**
	GPT3.5	0.94	0.85	0.89	0.91	0.86	0.89	0.93	0.81	0.86
	Llama3	0.95	0.76	0.84	0.92	0.81	0.86	**0.95**	0.65	0.77
	MistralAI	0.93	0.91	0.92	0.92	0.91	0.92	0.85	**0.94**	0.89
	Gemini	0.94	0.70	0.80	**0.97**	0.53	0.68	0.84	0.61	0.70
Syn. Error Type	GPT4	**0.96**	**0.95**	**0.95**	**0.89**	**0.88**	**0.88**	**0.90**	**0.89**	**0.89**
	GPT3.5	0.87	0.85	0.85	0.85	0.82	0.83	0.83	0.78	0.78
	Llama3	0.83	0.79	0.79	0.79	0.76	0.76	0.78	0.67	0.64
	MistralAI	0.90	0.88	0.89	0.81	0.80	0.79	0.86	0.81	0.82
	Gemini	0.81	0.74	0.73	0.73	0.60	0.58	0.68	0.53	0.52

表 3: syntax_error **和** syntax_error_type **的准确率**

确率。尽管这两种方法在模型初始性能不足的情况下均有所帮助，但我们在研究中并未采用其中任何一种方法。我们的目标是评估大模型在极少额外训练情况下的表现，以反映其在标注数据有限的环境中的实际表现。

4 实验结果

我们呈现研究结果与分析，每个小节聚焦一个主要的 SQL 任务及其相关的次要任务。

在所有实验中，GPT4 始终优于其他模型，在大多数情况下没有明显的第二名。这种优势可能源于更大的模型大小，如我们在第 3.3 节所概述的，也可能是因为该模型在更大规模的 SQL 查询语料库上进行了训练。为避免重复，这一普遍观察结果将不再在各个结果讨论中重复提及。

4.1 语法错误任务

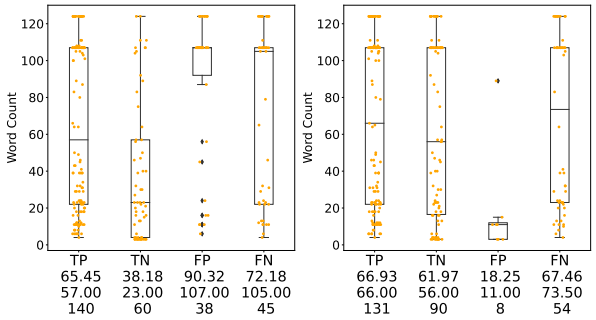
在本节中，我们展示了关于两个相关任务 syntax_error 和 syntax_error_type 的结果。

syntax_error. 表 3（上）展示了在 syntax_error 任务上的对比准确率。表现最佳的模型用粗体标出，第二佳的模型用下划线标出。GPT4、GPT3.5 和 MistralAI 表现良好，而 Llama3 和 Gemini 则表现不佳。这可能是因为 Llama3 是在通用文本上训练的，而 Gemini 更侧重于人工智能伦理和多模态任务，这意味着两者相对于其他模型而言，在 SQL 方面的知识较少。

在所有模型中，召回率往往低于准确率，这表明这些模型在检测错误时更为保守，会遗漏一些已存在的语法错误（召回率较低），但错误地声称存在错误的情况较少（准确率较高）。一种可能的解释是，这些模型可能在正确 SQL 查询上进行了更广泛的训练，而对语法错误示例的接触较少。这种准确率-召回率的不平衡在 Llama3 和 Gemini 中尤为明显，它们的召回率显著偏低，导致 F1 得分也相应降低。

一个关键问题是，大模型在什么情况下以及为何会遇到 syntax_error。为了探究这一问题，我们检验了两个假设：其一，错误与查询的语法特性相关，例如 word_count 或 table_count；其二，错误与特定类型的语法错误有关，例如 aggr-attr 或 nested-mismatch，如第 3.1 节所讨论。我们在测试这些假设时，对其他任务也应用了相同的分析方法。

对于第一个假设，我们分析了句法属性在四类中的分布：真正例（TP）、真负例（TN）、假正例（FP）和假负例（FN）。图 6 展示了 syntax_error 在 SDSS 中的情况，按 word_count 显示查询的分布。每个类别下方的数字分别表示平均值（上方）、中位数（中间）和查询总数（下方）。图 6a 和 6b 分别展示了 Llama3 与 Gemini 的类似数据。



(a) Llama3 **(b) Gemini**

图 6: word_count **与 SDSS 中** syntax_error **模型失败的关系。三个数字（例如 65.45、57.00、140）分别表示该类别（TP）的平均查询长度、中位数查询长度以及查询数量。橙色散点图表示按查询长度在纵轴上绘制的查询，展示了各分类的长度分布。**

为了探究查询长度 (word_count) 与模型失败之间的相关系数，我们比较了存在错误的查询中的真正例 (TP) 与假反例 (FN)，以及无错误查询中的真反例 (TN) 与假正例 (FP)，并判断各类中是否存在显著的查询。例如，在图 6a 中，TP 和 FN 类别均有足够数量的查询（分别为 140 和 45），可观察到一种模式：假正例 (FP) 查询通常显著更长（平均值 65.45 对比 72.18，中位数 57 对比 105）。在比较 TN 与 FP 时也观察到类似趋势，即 FP 查询更长（平均值 38.18 对比 90.32，中位数 23 对比 107）。这一趋势在图 6b 中 Gemini 的 TP 与 FN 比较中同样可见，但 FP 类别查询数量不足，无法对 TN 与 FP 得出结论。总体而言，这些发现表明查询长度 (word_count) 与 syntax_error 中的失败之间存在相关系数，较长的查询更容易被误分类。我们在其他语法属性或不同模型、数据集上未观察到类似模式，说明 word_count 是影响该任务失败似然性的最重要因子。

Case	Model	SDSS			SQLShare			Join-Order		
		Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Missing Token	GPT4	**0.99**	**0.97**	**0.98**	**0.98**	**0.96**	**0.97**	**1.00**	**0.97**	**0.99**
	GPT3.5	0.92	0.92	0.92	0.97	0.88	0.93	0.98	0.94	0.96
	Llama3	0.96	0.94	0.95	0.91	0.92	0.91	0.97	0.94	0.96
	MistralAI	**0.99**	0.86	0.92	0.96	0.87	0.91	**1.00**	0.94	0.97
	Gemini	**0.99**	0.76	0.86	**0.98**	0.68	0.80	0.97	0.69	0.81
Token Type	GPT4	**0.94**	**0.94**	**0.94**	**0.91**	**0.89**	**0.90**	**0.98**	**0.97**	**0.98**
	GPT3.5	0.76	0.75	0.75	0.75	0.71	0.73	0.84	0.82	0.82
	Llama3	0.88	0.85	0.86	0.78	0.69	0.72	0.87	0.82	0.84
	MistralAI	0.89	0.85	0.86	0.82	0.75	0.78	0.93	0.88	0.90
	Gemini	0.63	0.63	0.54	0.75	0.53	0.57	0.44	0.60	0.39

表 4: miss_token **和** miss_token_type **的准确率**

对于第二个假设，图 7 展示了每种语法错误类型的误报（FN）查询比例，其中柱状图越高表示模型检测该类型错误的难度越大。SDSS 的结果（图 7a）表明，类型不匹配错误（nested-mismatch ■ 和 condition-mismatch ■）对所有模型来说都特别难以检测。这在预期之中，因为工作负载中的查询包含大量条件，使得操作数的类型较难判断。对于 SQLShare （图 7b），歧义别名（alias-ambigous ■）错误更为严重，这也在预料之中，因为这些查询中使用了大量模式和多样化的表别名。最后，在 Join-Order （图 7c）中，涉及嵌套查询使用不当的错误（nested-mimatch ■）最常被模型遗漏，这是因为与 SDSS 类似，Join-Order 中的查询在 WHERE 子句中也包含冗长的条件。

syntax_error_type. 表 3（底部）展示了 syntax_error_type 的权重准确率，该指标考虑了六种语法错误类型（见第 3.1 节）。GPT4、MistralAI 和 GPT3.5 在检测语法错误方面的优异表现也延伸到了错误类型识别任务中，而 Llama3 和 Gemini 的表现依然较差，这与预期一致。总体而言，syntax_error_type 的结果低于 syntax_error，反映了该任务的难度更高。另一个重要观察是，所有模型在 SQLShare 数据集上的表现均较低，这可能是由于其更复杂的模式结构，使得识别语法错误类型更具挑战性。

Takeaways: The analysis of syntax_error and syntax_error_type shows that GPT4, MistralAI, and GPT3.5 outperform Llama3 and Gemini, likely due to differences in training focus. Longer queries are more prone to errors, and the types of syntax errors the models struggle with largely depend on the specific dataset.

4.2 缺失的 token 任务

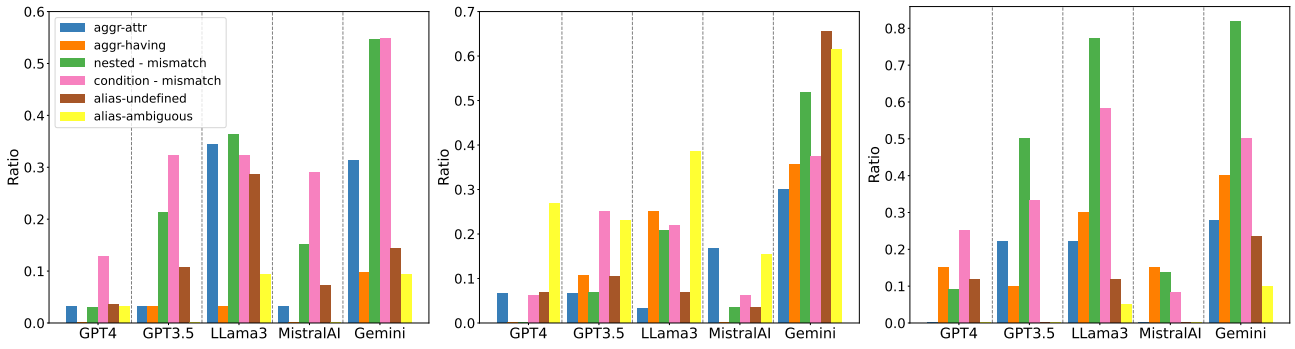
关于缺失 token，我们首先从 miss_token 开始，然后介绍与 miss_token_type 和 miss_token_loc 相关的结果。

miss_token. 表 4（上）展示了各种大模型在 miss_token 任务中的准确率。相较于 syntax_error，miss_token 的准确率更高，这是因为 miss_token 是一个更简单的任务。一个显著的变化是 Llama3 在该任务中的表现有所提升。这可以归因于检测缺失 token 更依赖于通用的模式识别能力，而这一能力对 SQL 并不特别专精。Llama3 在模式识别方面更广泛的训练可能使其在此情境下表现更好。总体而言，miss_token 中的召回率仍低于准确率，与 syntax_error 类似，这可能是由于模型在检测错误时更为保守，如前所述。

我们研究了大型语言模型在 miss_token 任务中的失败与查询句法属性之间的关系。图 8a 显示，对于 GPT3.5 在 SQLShare 数据集上的表现，查询长度 (word_count) 与失败相关，FN 的平均 word_count 为 57，而 TP 的平均值为 27。我们还考察了其他属性，如谓词数量 (predicate_count)、嵌套层级 (nestedness) 以及表数量 (table_count)，如图 8b、8c 和 8d 所示。在所有情况下，FN 的平均值显著高于 TP (在 8b 中为 1.80 对比 0.90，在 8c 中为 0.44 对比 0.05，在 8d 中为 1.92 对比 1.33) 。然而，由于 FP 查询数量较少，无法对该类别得出明确结论。

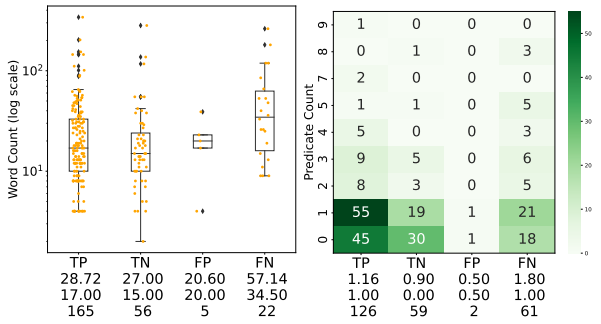
我们现在将分析重点转向缺失 token 类型对大模型在 miss_token 情况下性能的影响。我们考察了按 token 类型划分的漏报（FN）分布，如图 9 所示，这与我们在 syntax_error 情况下的分析类似。在 SDSS 中的一个关键观察是，最常见的失败类型为 keyword (■)。这很可能是由于 SDSS 包含多种多样的查询类型，其中关键词的出现频率高于 SQLShare 和 Join-Order。在 SQLShare 中，最难以处理的缺失 token 类型是别名和表 (■ 和 ■)，这可以归因于其查询中存在大量小型数据库，包含众多表和各种别名。最后，在 Join-Order 中，并没有某一种 token 类型表现出显著更高的失败率，这可能是因为查询本身较为简单，且总体失败数量相对较少。

miss_token_type. 我们在表 4（底部）中报告了加权平均准确率，权重基于每种类型的查询数量。结果表明，所有大模型中 miss_token_type 比 miss_token 更具挑战性，这从准确率的下降可以得到证据。最低准确率出现在 SQLShare，这是预期的，因为其模式比 SDSS 和 Join-Order 更复杂。相反，Join-Order 准确率最高，反映了其更简单的模式。值得注意的是，MistralAI 始终保持第二好的性能。这很有趣，因为 Llama3 在 miss_token 中排名第二，这表明尽管 Llama3 因其在检测一般模式方面的优势而表现更好，但 MistralAI 在 SQL 相关模式识别方面表现更佳，能正确判断更多查询的错误类型。

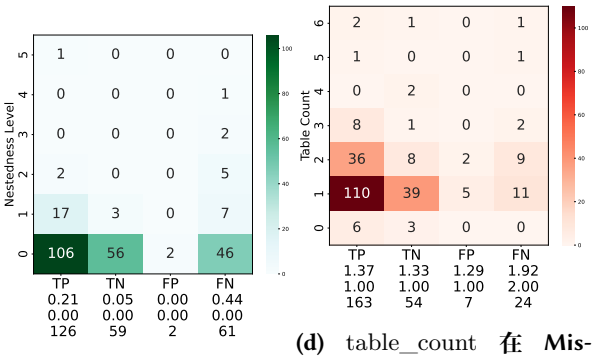


(a) SDSS **(b) SQLShare** **(c) Join-Order**

图 7: syntax_error **中语法错误类型与 FN 之间的关系**



(a) word_count **在 GPT3.5(b)** predicate_count **在中 Gemini 中**



(c) nestedness **在 Gemini 中 (d)** table_count **在 MistralAI 中**

图 8: 大模型在 miss_token **任务中对 SQLShare 的失败。**

Model	SDSS		SQLShare		Join-Order	
	MAE	HR	MAE	HR	MAE	HR
GPT4	**4.69**	**0.56**	**3.96**	**0.63**	**3.45**	**0.57**
GPT3.5	17.71	0.25	7.71	0.42	14.31	0.39
Llama3	15.60	0.33	7.57	0.40	13.11	0.39
MistralAI	18.09	0.36	8.58	0.42	9.92	0.40
Gemini	19.78	0.34	9.79	0.38	20.22	0.32

表 5: 平均绝对误差（MAE）和命中率（HR）对于 miss_token_loc

Model	Prec.	Rec.	F1
GPT4	**0.88**	**0.93**	**0.90**
GPT3.5	0.81	0.83	0.85
Llama3	0.76	0.90	0.82
MistralAI	0.47	0.90	0.62
Gemini	0.71	0.73	0.72

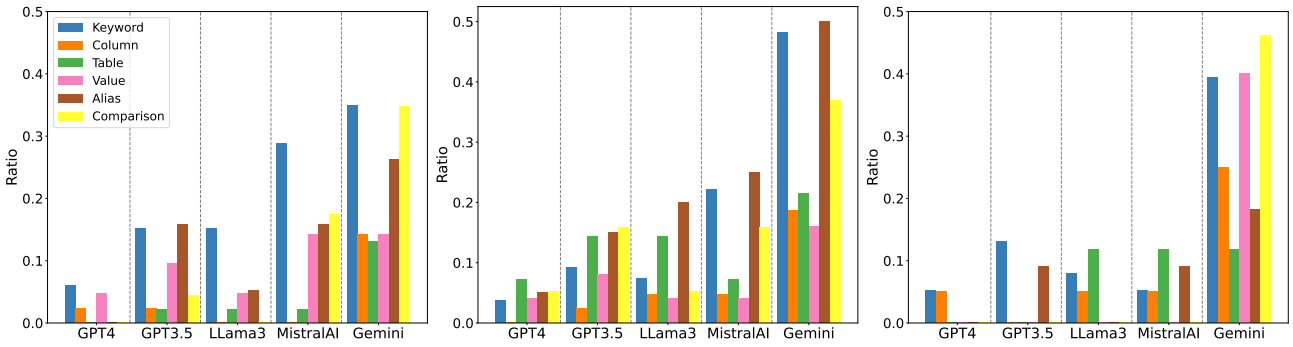
表 6: 根据 performance_pred

miss_token_loc. 表 5 比较了多种大模型在预测缺失 token 位置方面的表现，涵盖 SDSS、SQLShare 和 Join-Order。主要指标为平均绝对误差（MAE）和命中率（HR），其中 MAE 越低、HR 越高表示性能越好。

GPT4 在所有数据集上均表现出最佳性能，平均绝对误差（MAE）最低，命中率（HR）最高。Llama3 在 SQLShare 上表现良好，但在其他数据集上表现较弱。GPT3.5 和 MistralAI 提供了合理的表现，但其平均绝对误差（MAE）较高，命中率（HR）较低，反映出准确率和精度较差。

大多数模型至少有 30% 的时间能正确预测确切位置，除了在 SDSS 中的 GPT3.5，其命中率（HR）降至 25%。较长的查询，尤其是在 SDSS 中，导致平均绝对误差（MAE）值更高，使得精确位置预测更加困难。

Takeaways: All models perform better over the missing token tasks than syntax error detection, as missing token identification seems to be a simpler task related to learning frequent patterns. Llama3 shows improved performance due to its broad training in pattern detection. More complex queries tend to increase prediction errors, where complexity is related to different syntactic properties, such as word_count, predicate_count, nestedness, and table_count.



(a) SDSS **(b) SQLShare** **(c) Join-Order**

图 9: 缺失 token 类型与 miss_token 中的 FN 之间的关系

4.3 查询性能预测

表 6 展示了在 SDSS 数据集上 performance_pred 任务的性能指标。GPT4 的表现最佳，其次是 GPT3.5 和 Llama3，两者表现相似。MistralAI 和 Gemini 的整体性能较低。所有模型中，召回率普遍高于准确率，这可能是由于存在正向偏差。大模型倾向于产生过于乐观的响应，在此情况下表现为预测查询的执行时间更长。此外，这些查询选自 SDSS 中更复杂、更长的查询，因此被标注为高成本的可能性更高。

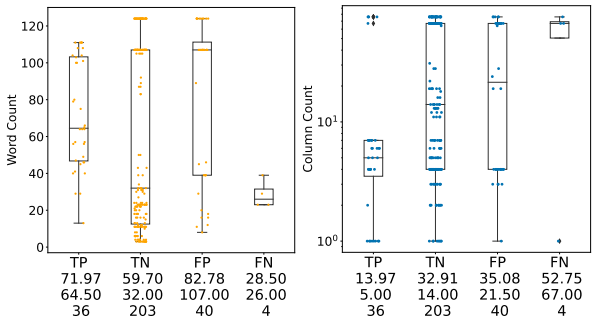
与 miss_token 类似，我们考察了句法特征与该任务失败率之间的关系。模型显示 word_count 与失败之间存在强相关性，查询越长，误报（FP）越多，如图 10a 中 MistralAI 所示。图 10b 中的 column_count 也呈现出类似趋势。这表明模型错误地将较长的查询或包含更多列的查询与更高的执行时间相关联。

Takeaways: In the query performance prediction task, GPT4 consistently shows the highest accuracy. However, all models tend to overestimate runtimes, leading to higher recall but lower precision, especially for longer and more complex queries. This suggests that improving model training with diverse query types could reduce this bias and enhance prediction accuracy.

4.4 查询等价

表 7 展示了 query_equiv (top) 和 query_equiv_type (bottom) 的结果。对于两项任务，GPT4 均取得了最佳性能，GPT3.5 和 Llama3 紧随其后，但表现略显不一致。MistralAI 和 Gemini 在各数据集上表现出更大的波动性，且整体得分较低。在 query_equiv 中可观察到正向偏差（召回率高于准确率），但在 query_equiv_type 中则不明显，这可能是由于后者为多分类任务而非二分类任务所致。

总体而言，query_equiv_type 的挑战性更高，在各类大模型和数据集上的表现均较低，仅 GPT4 保持



(a) word_count **(b)** column_count

图 10: MistralAI 在 performance_pred 中的失败

Case	Model	SDSS			SQLShare			Join-Order		
		Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Equivalence	GPT4	**0.98**	**1.00**	**0.99**	**0.97**	**1.00**	**0.99**	**0.91**	**1.00**	**0.95**
	GPT3.5	0.87	0.99	0.93	0.96	**1.00**	0.98	0.83	0.99	0.90
	Llama3	0.88	**1.00**	0.93	0.94	0.98	0.96	0.87	**0.99**	0.93
	MistralAI	0.95	0.95	0.95	0.95	0.93	0.94	0.86	0.89	0.88
	Gemini	0.84	0.97	0.90	0.92	0.99	0.95	0.85	0.96	0.90
Equiv. Type	GPT4	**0.99**	**0.99**	**0.99**	**0.98**	**0.98**	**0.98**	**0.95**	**0.85**	**0.83**
	GPT3.5	0.97	0.91	0.91	0.96	0.92	0.94	0.90	0.78	0.77
	Llama3	0.97	0.85	0.86	0.93	0.88	0.89	0.93	0.81	0.80
	MistralAI	0.85	0.76	0.80	0.92	0.88	0.89	0.84	0.68	0.68
	Gemini	0.86	0.72	0.71	0.91	0.85	0.87	0.87	0.77	0.75

表 7: query_equiv **和** query_equiv_type **中的准确率**

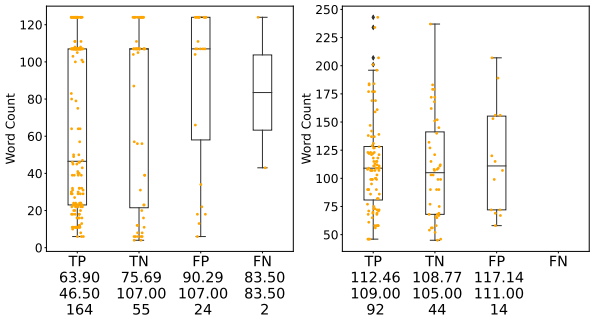
接近完美的准确率。这符合预期，因为判断等价性比识别等价类型要简单。相较于 SQLShare，Join-Order 和 SDSS 的表现较差，表明较长的查询使 query_equiv 更加困难。

在所有数据集上，大多数大模型的漏报（FN）非常少或没有，这反映在高召回率上。例如，GPT4 在 SDSS (5)、SQLShare (4) 和 Join-Order (9) 中记录了误报（FP），

但没有漏报（FN）。因此，我们重点关注误报以识别模型失败的位置。误报查询的一个共同特征是涉及修改条件，例如改变条件中的数值。例如，将 “WHERE run = 756 AND field = 103” 修改为 “WHERE run = 756 AND field = 200” 或 “WHERE run = 756 OR field = 103”。这表明大模型在逻辑推理和数值操作方面存在困难，这一局限性在文献中已有广泛讨论 [5–8, 12, 33]，我们的研究也证实了这一点。

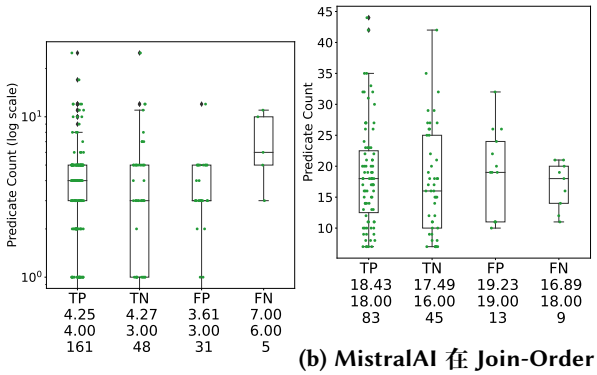
除了逻辑推理和数值问题外，这些错误在更复杂的查询中表现得更加明显，例如那些长度更长、涉及更多表和谓词的查询。在 SDSS 中，GPT4 的所有 5 个误报（FP）均来自超过 100 个单词的查询，这一模式在 GPT3.5（图 11）中也观察到。在 Join-Order 场景中，由于大多数查询都较长，所有大模型中的误报（FP）和漏报（FN）均更为频繁。我们仅报告 Llama3 在 Join-Order 的结果，因为其他大模型表现出相似趋势。将 table_count 视为复杂度参数，在 Join-Order 中，所有误报均出现在涉及超过 8 个表的查询中。图 12 显示，在 SDSS 中，误报出现在包含 5 个或更多谓词的查询中。类似地，在 Join-Order（图 12）中，所有模型的误报均由超过 19 个谓词的查询引起。我们包含 MistralAI 的图表，因为所有大模型均呈现相同模式。这些结果表明，对于复杂查询（即更长的查询，包含更多谓词和表），query_equiv 和 query_equiv_type 更具挑战性。

Takeaways: In the query equivalence task, GPT4 performs best across datasets. However, distinguishing between different types of equivalence (query_equiv_type) proves more difficult, especially for Gemini and MistralAI. The errors mainly stem from challenges in understanding complex query conditions, highlighting the need for better SQL logic comprehension in LLMs.



(a) GPT3.5 在 SDSS 中　(b) Llama3 在 连接顺序中

图 11: word_count **和** query_equiv **中的 LLM 失败。**



(a) Gemini 在 SDSS 中　(b) MistralAI 在 Join-Order 中

图 12: predicate_count **与** query_equiv **中的 LLM 失败**

4.5 案例研究：查询解释

我们研究了 query_exp 任务，并分析了大模型无法为 SQL 查询提供准确解释的若干案例。查询内容如清单 3 所示。此处我们展示来自 Spider 的正确真实值描述、模型生成的错误解释，并简要提供我们的分析：

Q15. 该查询找出每个学院参加选拔赛的学生人数，按人数降序排列。Gemini 错误地将该查询描述为：“统计 cName 列中每个唯一值的出现次数。” 这种解释将查询简化为对 cName 列中值的简单计数，忽略了该查询特别关注的是参加选拔赛的学生这一关键信息，未能完整传达其真正含义。

Q16. 该查询识别课程注册结果在不同成绩单中出现的最大次数，并显示课程注册编号。Gemini 的解释是：“找出出现次数最多的学生课程编号。” 虽然这部分捕捉到了查询的目的，但忽略了查询中在成绩单中搜索的上下文。

Q17. 该查询用于查找在 2014 年和 2015 年都举办过演唱会的场馆名称及其位置。GPT4 解释为：“该查询识别出在 2014 年和 2015 年均举办过演唱会的场馆。” 这种解释仅部分说明了查询内容，且未包含所选属性。此类问题出现的原因是大模型通常关注查询的整体语义，却忽略了具体细节（如所选属性），尤其是在更复杂的任务中。

Q18. 该查询检索加速度最低的沃尔沃汽车的气缸数量。Llama3 错误地解释为： “此 SQL 查询检索加速度最快的沃尔沃汽车的气缸数量。” 这些模型误解了 “ORDER BY ... ASC LIMIT 1” 子句，错误地认为查询是在寻找加速度最快（最高）的汽车，而非加速度最慢（最低）的汽车。只有 MistralAI 正确解释了该查询。

Takeaways: These examples highlight a common issue with LLMs when explaining SQL queries: they often miss

or misinterpret key details, particularly in tasks requiring context retention. While models may capture parts of a query, they frequently fail to provide complete and accurate explanations. This reflects known limitations of LLMs in retaining context and applying knowledge to specific scenarios [19, 25, 28].

```
-- Q15:
SELECT count(*),cName FROM tryout
GROUP BY cName ORDER BY count(*) DESC
-- Q16:
SELECT count(*),student_course_id FROM Transcript_Cnt
GROUP BY student_course_id ORDER BY count(*) DESC
    LIMIT 1
-- Q17:
SELECT S.name,S.loc
FROM concert AS C JOIN stadium AS S
ON C.stadium_id = S.stadium_id WHERE C.Year = 2014
INTERSECT
SELECT S.name,S.loc FROM concert AS C JOIN stadium AS
     S
ON C.stadium_id = S.stadium_id WHERE C.Year = 2015
-- Q18:
SELECT C.cylinders FROM CARS_DATA AS C
JOIN CAR_NAMES AS T ON C.Id = T.MakeId
WHERE T.Model = 'volvo'
ORDER BY C.accelerate ASC LIMIT 1;
```

Listing 3: Query statements with inaccurate explanations

4.6 对 SQL 理解的思考

我们设计的 SQL 任务旨在考察基本的理解能力（如第 1 节所述）：识别、语义、上下文和连贯性。

Demonstrated Skills. 我们的结果表明，大模型尤其是 GPT-4 在需要识别和上下文理解的任务中表现良好。例如，在语法错误检测任务中，模型在识别 SQL 语法违规方面表现出较高的准确率，表明其具备较强的识别和解析 SQL 查询结构的能力。同样，缺失 token 的识别依赖于模型的上下文感知能力，因为预测查询中缺失元素的能力取决于对周围 token 及其关系的理解。这一成功凸显了大模型能够有效解读查询中的上下文，以识别缺失或错误的组成部分。

Limitations and Shortcomings. 模型在需要更深层次连贯性和语义理解的任务中表现较差。例如，查询等价性任务对于较长且更复杂的查询尤其具有挑战性。这一困难表明，尽管大模型能够理解表层结构，但在查询内部的深层语义连贯性和逻辑关联方面仍存在困难。在查询性能估计方面，模型经常高估运行时间，说明它们缺乏对查询复杂度与数据库特定因子如何相互作用以影响性能的细致理解。

这些观察表明，尽管大模型在识别模式和上下文方面表现优异，但它们在全面“理解”SQL 查询的深层语义和逻辑连贯性方面仍处于发展阶段。未来的工作应致力于优化这些模型，以弥补这些不足，提升其整体的 SQL 能力。

5 相关工作

大模型的最新进展推动了数据管理领域的创新方法，解决了数据整理、实体匹配、表格操作以及文本转 SQL 生成等任务。

Li 等人 [18] 提出了一种基于大语言模型（LLM）的数据整理方法，该方法利用代码生成实现结构化数据变换。与传统 LLM 方法中常见的逐行处理相比，该方法显著降低了计算成本。他们的研究强调了确定性变换的重要性，以提升模型在单位转换和错误检测等数据任务中的可解释性和可靠性。

在实体匹配中，Zhang 等人 [37] 提出 AnyMatch，这是一种 zero-shot 实体匹配模型，仅使用一个小型专用大模型即可实现具有竞争力的性能。通过采用高效的数据选择技术，该模型的表现可与 GPT-4 等更大模型相媲美，同时所需的计算资源更少。与此互补的是，Steiner 等人 [29] 探讨了对大模型进行微调在实体匹配中的优势，结果显示性能有显著提升，但也指出微调可能降低跨领域通用性。

大模型已被用于表格操作，如 Li 等人 [17] 所示的 Table-GPT。该微调模型专为数据清洗和基于表格的问答系统等任务设计。研究表明，仅在自然语言文本上训练的大模型在处理二维表格数据时存在局限性，因此需要针对表格进行特定的微调以克服这些挑战。

大模型在文本转 SQL 任务上也取得了显著进展。Hong 等人 [11] 和 Gao 等人 [9] 的综述提供了大模型处理复杂且跨领域 SQL 生成的概述。这些研究指出，尽管大模型在简单查询上表现良好，但随着嵌套查询、连接和聚合等更复杂结构的引入，其准确率会下降。

这些研究突显了大模型在数据管理中日益重要的作用，涵盖实体匹配、数据整理以及文本转 SQL 等多个方面。尽管它们在自动化复杂任务方面具有巨大潜力，但仍需进一步研究以克服效率、可扩展性以及跨领域泛化等方面的挑战。

6 结论与未来工作

在本文中，我们研究了先进大模型在“理解”SQL 方面的熟练程度。我们评估了它们在关键 SQL 任务中的表现，包括语法错误识别、缺失 token 识别、查询等价性判断、查询性能估计以及查询解释。我们的评估结果表明，所有模型在需要模式识别和上下文理解的任务上表现良好。GPT4 在处理复杂 SQL 查询方面持续优于其他模型，而 GPT3.5、MistralAI 和 Llama3 在模式识别方面展现出强大的能力。相比之下，Gemini 在所有 SQL 特定任务中表现不佳，尤其是在错误检测方面。尽管存在这些优势，大模型在处理长且复杂的

查询时仍面临挑战，无法准确定位缺失 token 的具体位置，并且在需要语义连贯性和查询内部逻辑关联的任务中表现困难。作为下一步工作，我们将探索通过微调来应对查询复杂性，以及动态提示调整（以提高准确率），并研究使用大模型进行查询推荐和查询最优化所面临的障碍。

我们预期针对性的微调和动态提示调整能够显著缓解当前在处理复杂查询时的局限性，并提升特定任务的表现。这一对大模型的改进有望缩小人工智能能力与实际 SQL 需求之间的差距，从而实现与数据库系统的更好集成。

REFERENCES

[1] Anthropic. 2024. Gemini: A Safe and Ethical Large Language Model. *Anthropic Research* (2024).

[2] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L Glassman. 2024. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing. In *CHI*. 1–18.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165* (2020).

[4] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *PODS*. 34–43.

[5] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[6] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *ArXiv* abs/2110.14168 (2021). https://api.semanticscholar.org/CorpusID:239998651

[7] Antonia Creswell, Murray Shanahan, and Irina Higgins. [n.d.]. Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning. In *The Eleventh International Conference on Learning Representations*.

[8] Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. 2024. Mathematical capabilities of chatgpt. *NeurIPS* 36 (2024).

[9] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. [n.d.]. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. ([n. d.]).

[10] Alon Y Halevy et al. 2014. SQLShare: A Platform for Structured Data Sharing. *CIDR* (2014).

[11] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. *arXiv preprint arXiv:2406.08426* (2024).

[12] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*. PMLR, 9118–9147.

[13] Won Kim. 1982. On optimizing an SQL-like nested query. *ACM TODS* 7, 3 (1982), 443–469.

[14] Eugenie Yujing Lai, Zainab Zolaktaf, Mostafa Milani, Omar AlOmeir, Jianhao Cao, and Rachel Pottinger. 2023. Workload-Aware Query Recommendation Using Deep Learning. In *EDBT*. 53–65.

[15] Bruce W Lee and JaeHyuk Lim. 2024. Tasks That Language Models Don't Learn. *arXiv preprint arXiv:2402.11349* (2024).

[16] Viktor Leis et al. 2015. Join Order Benchmark. *VLDB* (2015).

[17] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks. *ACM SIGMOD* 2, 3 (2024), 1–28.

[18] Xue Li and Till Döhmen. 2024. Towards Efficient Data Wrangling with LLMs using Code Generation. In *DM4ML*. 62–66.

[19] Xing Liu et al. 2023. Attention Mechanisms in Large Language Models: A Critical Review. *JAIR* 69 (2023), 1021–1050.

[20] Ggaliwango Marvin, Hellen Nakayiza, Daudi Jjingo, and Joyce Nakatumba-Nabende. 2023. Prompt engineering in large language models. In *Springer DICI*. 387–402.

[21] MistralAI. 2023. Mistral: New Model Architecture and Training. *Company Blog* (2023). Accessed: 2024-08-14.

[22] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *ICSE*. 1–13.

[23] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[24] Karl Pearson. 1895. Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London* 58 (1895), 240–242.

[25] Fabio Petroni et al. 2020. Contextualizing Knowledge for LLMs: Challenges and Opportunities. In *EMNLP*. 123–136.

[26] Jing Qian, Hong Wang, Zekun Li, Shiyang Li, and Xifeng Yan. [n.d.]. Limitations of Language Models in Arithmetic and Symbolic Induction. ([n. d.]).

[27] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).

[28] Noah Shinn, Shuyuan Kuo, et al. 2023. Context Forgetting in Large Language Models. *arXiv preprint arXiv:2303.12345* (2023).

[29] Aaron Steiner, Ralph Peeters, and Christian Bizer. 2024. Fine-tuning Large Language Models for Entity Matching. *arXiv preprint arXiv:2409.08185* (2024).

[30] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. [n.d.]. LLaMA: Open and Efficient Foundation Language Models. ([n. d.]).

[31] Thinh Hung Truong, Timothy Baldwin, Karin Verspoor, and Trevor Cohn. 2023. Language models are not naysayers: an analysis of language models on negation benchmarks. In **SEM*. 101–114.

[32] Li Wang, Xi Chen, XiangWen Deng, Hao Wen, MingKe You, WeiZhi Liu, Qi Li, and Jian Li. 2024. Prompt engineering in consistency and reliability with the evidence-based guideline for LLMs. *npj Digital Medicine* 7, 1 (2024), 41.

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[34] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. [n.d.]. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. ([n. d.]).

[35] Donald G York et al. 2000. The Sloan Digital Sky Survey. *AJ* (2000).

[36] Tao Yu et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *EMNLP* (2018).

[37] Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. 2024. AnyMatch–Efficient Zero-Shot Entity Matching with a Small Language Model. *arXiv preprint arXiv:2409.04073* (2024).

[38] Victor Zhong et al. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv preprint arXiv:1709.00103* (2017).

[39] Zainab Zolaktaf, Mostafa Milani, and Rachel Pottinger. 2020. Facilitating SQL query composition and analysis. In *SIGMOD*. 209–224.