EVOSCHEMA：应对模式演化的文本到 SQL 鲁棒性

张天舒 俄亥俄州立大学 俄亥俄州哥伦布
zhang.11535@osu.edu

Kun Qian 奥多比公司 华盛顿州西雅图
kunq@adobe.com

Siddhartha Sahai 奥多比公司 华盛顿州西雅图
siddharthas@adobe.com

Yuan Tian 普渡大学 印第安纳州西拉斐特
ti
an211@purdue.edu

Shaddy Garg 奥多比公司 班加罗尔
shadgarg@adobe.com

Huan Sun 俄亥俄州立大学 俄亥俄州哥伦布 sun.397@osu.edu

Yunyao Li 奥多比公司 加利福尼亚州圣何塞
yunyaol@adobe.com

arXiv:2603.10697v1 [cs.DB] 11 Mar 2026

摘要

神经文本到 SQL 模型能够根据数据库模式将自然语言问题（NLQ）转换为 SQL 查询，且已取得令人瞩目的性能。然而，数据库模式经常演化以满足新的需求。这种模式演化往往会导致在静态模式上训练的模型性能下降。现有研究要么主要致力于对自然语言问题、数据库和 SQL 之间的一些句法或语义映射进行简单改写，要么缺乏全面且可控的方式来研究模式演化下的模型鲁棒性问题，因此难以应对现实中日益复杂多样的数据库模式变更，尤其是在大语言模型时代。

为应对模式演化带来的挑战，我们推出了 EvoSchema，这是一个全面的基准测试，旨在评估并提升文本到 SQL 系统在真实世界模式变更下的鲁棒性。EvoSchema 提出了新颖的模式演化分类体系，涵盖列级和表级修改中的十种扰动类型，系统地模拟数据库模式的动态特性。借助 EvoSchema，我们对不同的开源和闭源大语言模型进行了深入评估，结果表明，与列级变化相比，表级扰动对模型性能的影响显著更大。此外，EvoSchema 还推动了更具韧性的文本到 SQL 系统的发展，涵盖模型训练和数据库设计两个方面。在 EvoSchema 多样化模式设计上训练的模型，能够促使模型针对相同问题辨别模式差异，从而避免学习伪相关模式；总体而言，与在未扰动数据上训练的模型相比，这些模型展现出显著更强的鲁棒性。该基准测试为理解模型行为提供了宝贵见解，也为设计能够适应动态真实环境的系统指明了方向。

PVLDB 引用格式：
张天舒、KunQian、Siddhartha Sahai、Yuan Tian、Shaddy Garg、Huan Sun 和 Yunyao Li。EvoSchema：迈向文本到 SQL 应对模式演化的鲁棒性。PVLDB，18(10)：3655 - 3668，2025年。
doi:10.14778/3748191.3748222

PVLDB 工件可用性：
源代码、数据和/或其他工件已在 https://github.com/zhangtianshu/EvoSchema 上提供。

1 引言

文本到 SQL 解析旨在根据数据库模式将自然语言问题（NLQ）转换为 SQL 查询，从而开发自然语言接口，使用户无需编程技能即可查询数据并调用服务 [18, 27, 29, 32, 33, 36]。现有神经文本到 SQL 模型在现有基准测试中取得了显著成果 [18, 32], ，这些模型在赋能商业和营销平台等不同平台方面发挥着重要作用 [26, 34] ，并被集成到虚拟助手中，以支持实时数据查询和分析 [4]。

然而，数据库模式并非一成不变；它们经常演化，以适应新的用例并提高效率 [3, 11]。例如，根据具体场景，大型患者表可能会被合并或拆分为两张表：患者信息表和患者诊断表（图 1-c），以减少冗余、增强数据完整性并优化性能 [14]。这种模式演化十分常见，往往会导致分布偏移[13, 24] ，例如命名偏移、数据粒度偏移、表与列关系偏移以及模式复杂度偏移。当模型基于旧数据库模式训练后适应新的模式设计时，这些分布偏移可能导致性能显著下降。

本作品依据知识共享署名—非商业性使用—禁止演绎 4.0 国际许可协议授权。请访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看此许可协议副本。对于超出本许可协议涵盖范围的任何使用，请发送电子邮件至 info@vldb.org 获取许可。版权归所有者/作者所有。出版权授权给 VLDB 基金会。VLDB 基金会会议录，第 18 卷，第 10 期，ISSN 2150-8097。doi:10.14778/3748191.3748222

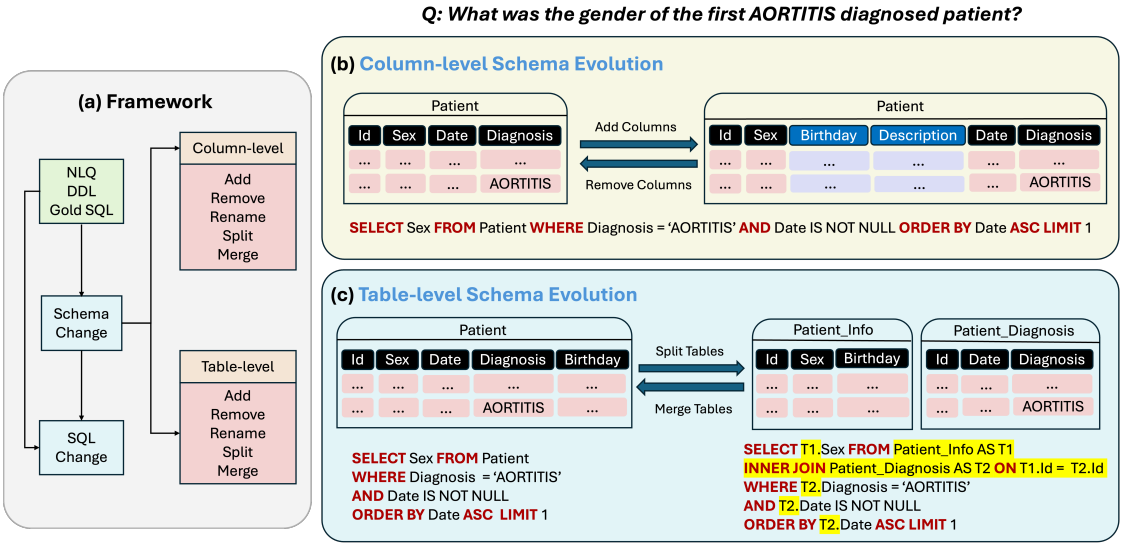


图 1：左侧 (a) 展示了收集 EvoSchema 数据集的框架概览。右上方 (b) 是列级模式演化示例；右下方 (c) 是表级模式演化示例。

这一挑战凸显了模型鲁棒性中的一个关键问题：文本到SQL模型适应数据库模式变化的能力有多强？近期研究引入了评估基准，通过扰动自然语言问题、数据库或 SQL 查询来揭示鲁棒性问题 [2, 6, 20, 23]。然而，这些研究至少存在以下一项局限：1）主要关注自然语言问题、数据库和 SQL 之间的句法改写或简单语义映射[2, 6]；（2）缺少对全面的模式演化类型进行分类的分类法 [23]；（3）只关注不会导致 SQL 变更的模式演化 [20]。面对现实中日益复杂且多样的数据库模式变更，这些工作还远远不够。与此同时，虽然模式演化后收集新数据以重新训练模型是很自然的做法，但频繁重复整个模型训练生命周期会耗费大量时间和资源。

在此背景下，我们试图回答以下两个问题：（1）现有文本到 SQL 模型对各种类型的数据库模式变更有多敏感？（2）我们如何训练出更鲁棒的文本到SQL模型，使其不仅能在现有数据库模式上表现良好，还能有效适应模式变更？为此，我们推出了 EvoSchema，这是一个新数据集，通过对 BIRD [18] 进行扰动，涵盖了广泛且真实的模式设计变更。如图 1 和图 2 所示，EvoSchema 基于我们新定义的分类法构建，涵盖了模式上的十种扰动类型，包括列级和表级变更。列级扰动包括添加、删除、重命名、拆分和合并列，而表级扰动则涉及添加、删除、重命名、拆分和合并表。我们保持自然语言问题不变，并考察模型在不同粒度的模式演化下的鲁棒性，结果表明，与列级扰动相比，现有模型更容易受到表级扰动的影响。

此外，EvoSchema 中的训练集可用于增强模型的鲁棒性。使用相同的问题并搭配不同的模式设计进行训练，模型便能生成相应的 SQL 查询。这一训练过程促使模型辨别模式差异，帮助模型提升识别正确表列关系并将其映射到问题的能力。实验结果表明，EvoSchema 中的扰动训练数据有助于训练出更好的文本到 SQL 模型，使其对不同模式演化类型的平均鲁棒性更强，尤其是在表级扰动方面。

总之，我们的主要贡献如下：

- 我们提出了一个关键的模式演化自适应文本到 SQL 问题，并推出了一个用于研究该问题的新数据集 EvoSchema。我们介绍了模式演化类型的全面分类法，并基于该分类法构建数据集，通过对 BIRD 进行列级和表级扰动，生成符合真实情况的模式设计。
- 我们在评测基准上，对各种模式扰动下不同开源和闭源大语言模型的模型鲁棒性进行了全面深入的评估，发现与列级变化相比，表级扰动对模型性能的影响显著更大。此外，我们引入了两个评测指标：表匹配 F1 和列匹配 F1，用于严格评估文本到 SQL 模型在模式演变场景下的性能，并深入分析模型鲁棒性。

- 我们构建的训练集启发了一种新的训练范式：使用不同的模式设计增强现有训练数据，这不仅增加了数据的多样性，还迫使模型区分模式

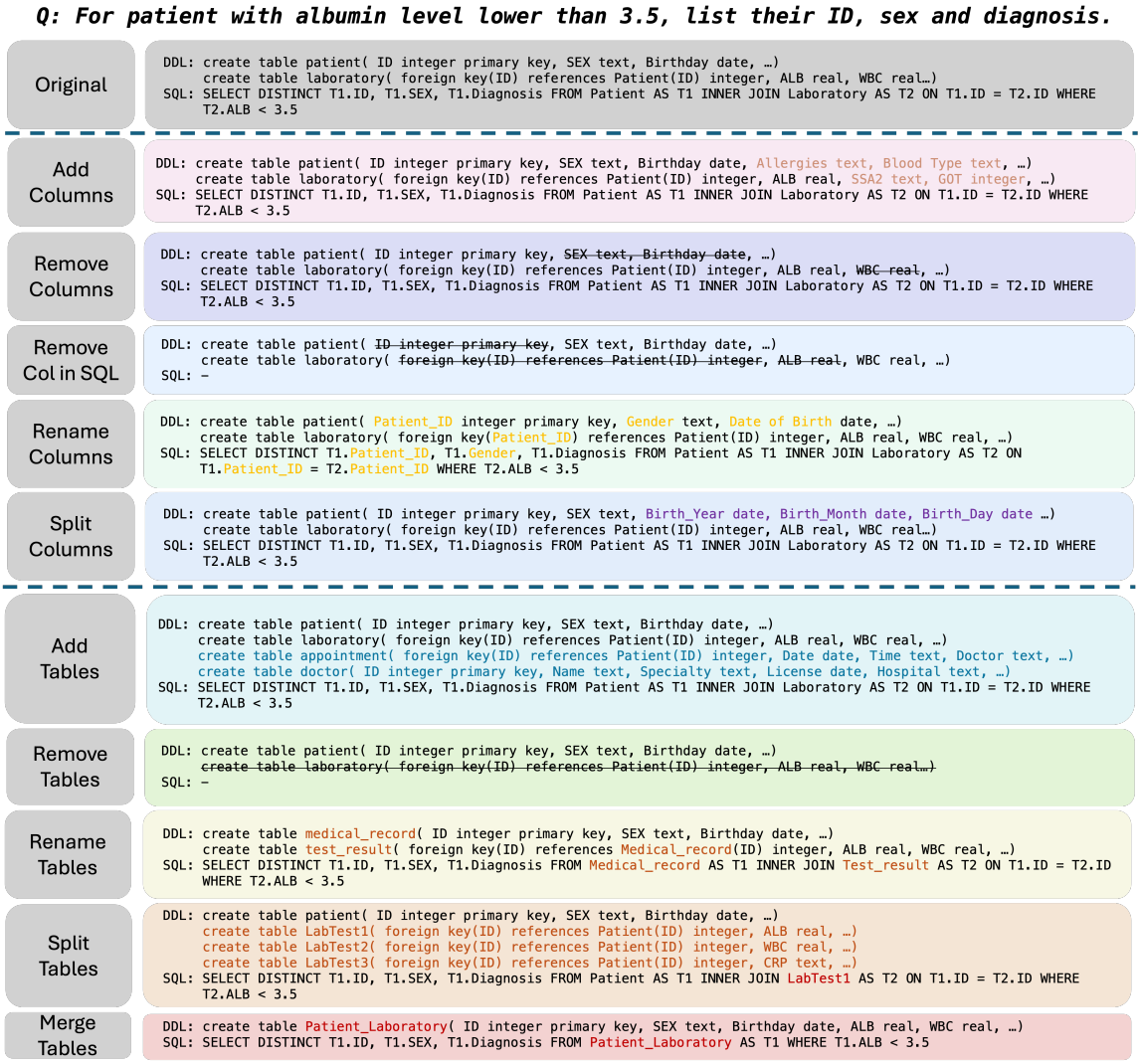


图 2：EvoSchema 不同扰动类型的概览。顶部是 BIRD [18] 中未经过扰动的示例；中间是列级扰动；底部是表级扰动。“在 SQL 中删除列”：删除标准 SQL 中出现的列；“删除表”：删除标准 SQL 中涉及的表。因此，这两种情况下都不存在标准 SQL。请注意，我们没有在图中展示“合并列”，因为此示例不适合应用合并列变更。

训练期间的差异。与使用未扰动的原始训练数据训练的模型相比，我们的方法能带来更优秀的文本到 SQL 模型，在不同类型的模式扰动评估数据上，执行准确率最高可提升 33 个百分点。

2 相关工作

文本到 SQL 的鲁棒性：现有关于文本到 SQL 鲁棒性的研究主要分为两类：鲁棒性评估和鲁棒性训练。近期研究提出了评测基准，通过扰动自然语言问题、数据库或 SQL 查询来揭示鲁棒性问题。然而，这些研究往往侧重于句法

NLQ、数据库和 SQL 中数字表示方式不同或名称缩写等释义转换或简单语义映射 [2, 6]。尽管有些工作分析了模式变更，但主要关注不会影响 SQL 的无关列修改 [20]，或仅涵盖有限的扰动类型 [23]。面对现代数据集中日益复杂且丰富的数据库模式，这些工作仍显不足。尽管 FootballDB [8]也着眼于类似的模式设计问题，以便编写更好的 SQL，但其重点是减少表之间的多个外键映射，并缩短 SQL 中的 JOIN 路径。与其不同，我们解决的是模式演化问题，这不仅涉及现有数据的模式设计，还需要考虑新数据和信息将如何改变模式设计。此外，我们

从不同角度着手，我们的模式设计包含 10 种列级和表级变更。我们提供的模式演化框架使我们能够在多个数据库上尝试不同的模式设计，从而获得更具普适性的发现；而 FootballDB [8] 只能支持在单个数据库上进行探索。此外，大语言模型的出现缓解了许多语言方面的挑战，进一步凸显了针对数据库模式结构变更进行鲁棒适配的必要性。为了实现鲁棒训练，现有方法采用了分解任务等策略，让模型先分别生成各个子句，再将其合并[9], , 或使用执行引导解码来消除错误的子句 [30]。这些方法侧重于增强文本到 SQL 鲁棒性的各个方面，而我们的工作则专门应对模式演化这一挑战。

用于文本到 SQL 的大语言模型：最近，基于大语言模型的文本到 SQL 方法主要分为两类：上下文学习 [10, 15, 16, 27, 35] 和 [15 17, 38] 微调。前者通过提示 GPT 系列 [1] 和 Claude 等专有大语言模型 [2] 生成 SQL，无需额外训练模型；后者则将开源大语言模型适配到文本到 SQL 数据集，通过监督学习使这些模型直接针对该任务进行调整。这些模型旨在理解问题、理解模式并生成 SQL，在现有开放基准测试上取得了显著成果 [18, 32]。Liu 等人 [19] 全面回顾了 NL2SQL 生命周期，涵盖模型、基准测试、数据合成、评估和错误分析。虽然该综述指出模式变化是一项挑战，却未对此进行深入探讨。我们的工作专门关注模式演化的鲁棒性，通过评估近期性能强大的大语言模型（如 Code Llama、Mistral、SQLCoder、LLaMA 3、GPT-3.5 和 GPT-4），且不进行预处理或后处理。我们提出了EvoSchema，这是一个包含受控模式扰动的基准测试，可用于指导评估和结构化训练数据合成。除了标准的执行准确率和人工评估外，我们还提出了两项细粒度指标：表匹配 F1 和列匹配 F1，它们直接反映了我们的表级和列级扰动分类法。Li 等人 [15]在未受扰动的 Spider 和 BIRD 数据集上评估大语言模型，并开展了自然语言变化实验，但保持模式和 SQL 不变；相比之下，我们的工作系统地改变模式，同时保持自然语言不变。

3 EvoSchema 数据集

3.1 背景

在不断变化的数据库环境中，数据库模式经常演进以满足新的需求，这给文本到 SQL 模型带来了重大挑战 [3, 5]。这些模式变更形式多样，从细微修改到全面重构都有，可能显著影响在静态模式上训练的模型的性能。在现实场景中，数据库通常包含大量表，而只有几张相关表与某个自然语言问题（NLQ）有关。在我们的实验中，我们使用数据定义语言（DDL）表示相关的数据库模式 [3]

[1]https://platform.openai.com/docs/models [2] https://www.anthropic.com/news/claude-3-family[3]DDL 定义了数据库的结构和属性，并提供创建数据库所需的详细信息，包括列类型以及主键/外键。

并将其与 NLQ 结合用作输入。随后使用该输入提示模型生成相应的 SQL 查询。

3.2 模式演化类型的依据

数据库模式演化可能会导致数据分布偏移，进而影响模型性能。我们将潜在的分布偏移分为四种类型：命名偏移、数据粒度偏移、表与列关系偏移，以及模式复杂度偏移。(1) 当表和列被重命名时，会发生命名偏移，这可能改变模式中既定术语的惯例。例如，原名为“Products”“Customers”和“Orders”的表，可能分别被重命名为“Items”“Clients”和“Purchases”。此类变更通常反映了业务术语的更新，或是为了符合新的标准。理想的模型应能应对这类命名偏移，以适应新的术语。(2) 添加或删除列或表会造成数据粒度偏移，从而改变数据库所记录信息的详细程度。例如，原本只有一个“ContactNumber”字段的“Employee”表，之后可能会增加“WorkContact”和“PersonalContact”两个独立字段。这样做会提高数据粒度，以满足新的需求，因此模型需要适应更复杂、更细致的语义。(3) 表与列关系偏移和模式复杂度偏移主要源于通过拆分或合并来重构表。这一过程可能显著影响各个表通过哪些列与其他表关联。表重构时，主键和外键也可能发生变化。此外，多个表合并为一个表或一个表拆分为多个表时，模式复杂度也可能改变。理想的模型应能稳健应对这类变化。通过对模式演化引发的分布偏移进行分类，我们可以更有效地理解和评估模型适应底层数据库模式变化的能力。

3.3 模式演化合成框架

我们的研究旨在覆盖全面的潜在模式演化类型，从而促进对现有文本到 SQL 模型的鲁棒性评估，并启发鲁棒的模型训练。我们通过混合策略合成所有模式演化类型，这些策略将同时利用启发式规则来保证数据质量，并利用大语言模型来确保多样性。

对不同模式演化类型的广泛覆盖：我们旨在囊括广泛的模式演化类型，并认识到它们在现实场景中的普遍性与影响。具体而言，我们的模式演化分类体系包括列级和表级扰动，这些扰动被划分为十种不同的类型。列级扰动包含五种类型：添加、删除、重命名、拆分和合并列，其中修改仅限于现有表内的列。表级扰动涵盖五种类型：添加、删除、重命名、拆分和合并表。这些扰动在实践中频繁发生，凸显了需要能够鲁棒地处理此类变化的文本到 SQL 模型。

混合数据合成策略： 为确保生成的模式扰动兼具多样性和质量，我们结合使用启发式方法与 GPT 模型来合成各种

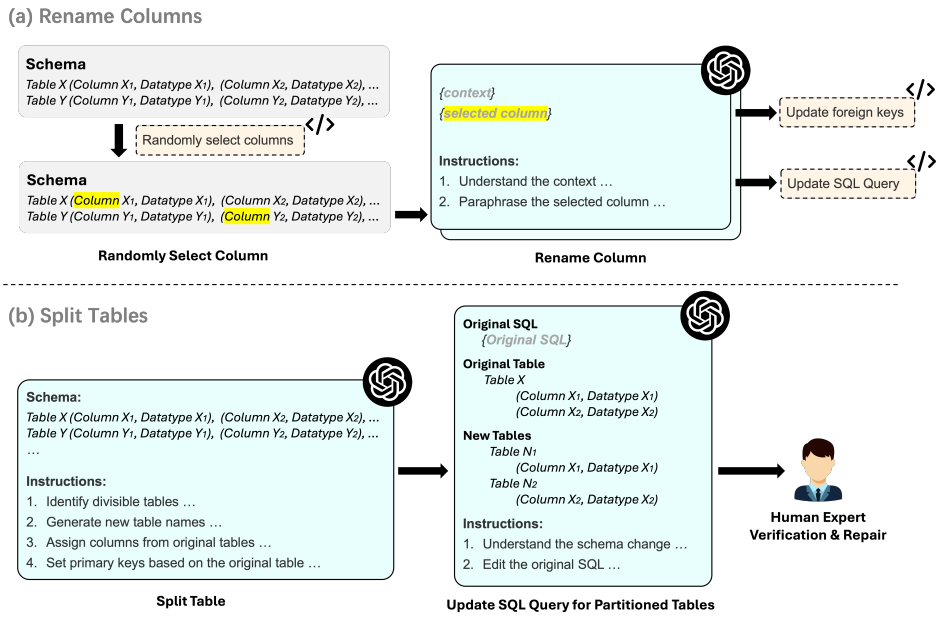


图 3：该图展示了 EvoSchema 数据收集流程中的两个示例。上方（a）为“重命名列”数据收集流程；下方（b）为“拆分表”数据收集流程。蓝色方框表示提示 GPT 模型进行生成。“</>”表示以编程方式处理数据。

扰动类型。对于 BIRD [18], 中的每个给定种子实例，该实例由一个 <自然语言问题、相关模式、*SQL*> 三元组组成，我们在所有扰动类型中保持自然语言问题（NLQ）不变，仅修改相关模式。相应的 SQL 查询会根据需要进行调整，以确保其与数据库模式中的变更保持一致。

3.4 种子数据集选择

为了构建 EvoSchema 基准测试，我们采用 BIRD [18] 数据集作为种子数据，该数据集专为文本到 SQL 任务设计。与常用于研究文本到 SQL 鲁棒性的 Spider [32]，相比，BIRD 包含更复杂、贴近现实且规模更大的数据库，以及更复杂的 SQL 查询，其中包含 Spider 中常缺失的关键词。BIRD 由自然语言问题、对应的数据库模式和标准 SQL 查询组成，涵盖广泛的真实世界数据库场景，为评估模型将自然语言问题转换为 SQL 的性能提供了可靠基础。

模式扰动：为了评估文本到 SQL 模型的鲁棒性，EvoSchema 不仅包含原始形式的 BIRD数据集，还通过多种列级和表级模式扰动对其进行了增强。我们确保自然语言问题保持不变，同时根据扰动引入的变化，对模式和 SQL 查询进行必要的调整。我们遵循 BIRD 提供的标准训练集/开发集划分，并将所有扰动应用于训练数据和评估数据。EvoSchema 的数据统计见表2，不同扰动类型的示例见图2。

3.5 数据生成

我们设计了一个框架，以可配置的方式模拟不同类型的模式扰动。对于添加或重命名列，修改后的列数量和列在表中的位置均随机设置，并将表中的原始列数量设为待更改列数的最大值。对于删除列，我们可以从现有的相关表中随机删除重要列或不重要列。重要列是指出现在标准 SQL 中的列，这些列的变化必然会影响预测结果。对于添加、删除或重命名表，我们会随机添加、删除或重命名一个或多个表。

模式变更： 为确保合成模式的多样性和合理性，我们利用 GPT-3.5 和 GPT-4 的能力，合成真实且符合上下文的列或表，从而有效生成满足我们要求的高质量合成数据。对于添加或重命名列和表，我们将现有的相关表输入 GPT-3.5，让模型生成符合上下文的潜在表或列。对于拆分表或合并表，由于其比其他扰动更复杂，我们使用 GPT-4 选择可拆分或合并的表，然后用修改后的表替换原始表。对于添加或重命名列和表，我们采用启发式方法，过滤合成表或列中的重复项。此外，为确保修改模式后不同表之间的关系正确，我们采用启发式方法，确保所有外键都随其引用的表名和列名一同变更。删除列或表时，其他表中所有引用被删除列或表的外键也会一并删除。

SQL 变更： 为确保 <自然语言问题、相关模式、*SQL*>的一致性，在变更相关表模式后，我们会相应修订标准 SQL。由于添加或删除列和表时自然语言问题保持不变，且此处的模式演化不会影响问题的解答，因此对于这些扰动类型，我们保留标准 SQL 不变。对于重命名列或表，如果它们出现在标准 SQL 中，我们会修订标准 SQL。对于拆分表或合并表，由于所需 SQL 变更的复杂性和多样性，我们使用 GPT-4 修订标准 SQL。修订时会依据原始表和列到新表和列的映射关系，并对 JOIN 路径进行必要调整。我们会人工检查评测基准中编辑后的标准 SQL，确保其正确无误。

3.6 各扰动类型的数据收集

我们首先为不同的扰动类型定义启发式方法，然后结合 GPT 模型的生成能力与编程来收集数据。最后，我们加入人工核验环节，以控制数据质量。为保持一致性并避免在操作数据时发生冲突，我们应考虑以下通用启发式方法：1）保留含义：重命名时，新的列名或表名应体现与原名称相同的含义，以避免语义混淆。2）避免冲突：确保新列名或表名不会与数据库中同一表或其他表的现有列名或表名冲突。3）更新引用：更新其他表的外键中对新列或表的所有引用。4）修改 SQL：更新所有引用新列或表的 SQL 查询，确保重命名后能够正确运行。这些启发式方法旨在确保系统地执行这些扰动，同时维护数据库的完整性以及与 SQL 查询的兼容性。每种扰动类型的具体细节如下：

添加列：我们将表名及其所有列名和数据类型输入 GPT-3.5，并提示它根据当前场景，结合合理性和常识，生成多个合适且相符的列名及其对应的数据类型；同时提示 GPT-3.5 不要生成与现有输入列名含义相近的列名。然后，我们添加一种启发式方法，在生成的列名重复时将冗余列过滤掉。接着将这些合成列随机插入相关表中。值得注意的是，在此过程中，自然语言问题和标准 SQL 均保持不变。

删除列：我们从给定模式中随机删除列，并确保被删除的列不会出现在标准 SQL 查询中。同样，在此操作期间，自然语言问题和标准 SQL 保持不变。

删除标准 SQL 中的列：在此情形下，我们从模式中随机删除列，特别是删除标准 SQL 查询中引用的列。因此，标准 SQL 将变为无效。我们改用回复“给定的列信息不足以生成 SQL 查询来回答该问题”作为真实值。

重命名列：如图 3 (a) 所示，我们将表名及其所有列名和数据类型一并输入 GPT-3.5。我们随机选择多个列名及其数据类型，并提示 GPT-3.5 生成相似且符合上下文的名称。这些合成名称将替换原始列名。此外，为了保持表之间关系的正确性，如果某个表中的列已被重命名，那么对于其他表中引用该列的外键，我们也会进行重命名。我们还会相应修订标准 SQL，以确保修订后的模式与标准 SQL 始终与未改变的自然语言问题保持一致。

拆分列：由于在真实世界数据库中，姓名、地址和日期等列通常以更细粒度的格式存储（例如，将完整姓名拆分为名和姓；将日期拆分为年、月和日；将地址拆分为州、城市和街道等），我们找出 BIRD 开发集中涉及这些属性的示例，并手动将相应的列拆分为更细粒度的列，以进行评估。由于这些更改会影响标准 SQL 查询的结构，我们手动修改标准 SQL，以反映更新后的模式。对于训练集，我们同样选取 BIRD 训练集中涉及姓名、地址或日期的示例，并使用 Claude 3.5 合成相应的细粒度列，同时相应地更新标准 SQL。

合并列：作为列拆分的反向操作，我们模拟真实世界数据库中常见的更抽象的列表示形式（例如，将名和姓合并为完整姓名；将年、月和日合并为日期；将州、城市和街道合并为地址）。我们找出 BIRD 开发集中相关的示例，并手动合并细粒度的列，同时相应地更新标准 SQL。对于训练，我们对 BIRD 训练集采用相同的策略，并使用 Claude 3.5 合成合并后的模式，同时更新标准 SQL。

添加表：我们为每个问题随机添加无关表，这些表与 BIRD 中的相关表位于同一数据库中。原始 BIRD 数据集保证，其数据库中的其他表无法得出其他正确的 SQL 答案。添加的表不会影响自然语言问题和标准 SQL。

删除表：在此情境下，我们从相关模式中随机删除被标准 SQL 查询引用的表。因此，标准 SQL 将变得无效。我们改用回答“给定的表信息不足以生成 SQL 查询来回答该问题”作为真实值。

重命名表：我们将表名及其所有列名和数据类型输入 GPT-3.5。我们随机选择一个或多个表名，并提示 GPT-3.5 生成相似且符合上下文的名称。这些合成名称将替换原表名。此外，为了保持表之间关系的正确性，如果其他表中的外键引用了被重命名的表，我们也会重命名这些外键。最后，标准 SQL 中的表名也会相应重命名。

拆分表：如图 3 (b) 所示，我们将表名及其所有列名和数据类型输入 GPT-4。我们提示 GPT-4 识别可以在逻辑上划分为两个或多个更小表的表。借助 GPT-4，我们生成新表名，并以符合上下文的方式将原表的列分配到各新表中。拆分后，原表中的主键将被复制到所有新表中。标准 SQL 由 GPT-4 修订，以引用新创建的表，从而确保各组成部分之间的一致性。我们还会人工检查新黄金SQL，以确保其正确。

表1：EvoSchema 与现有基准测试的统计数据对比。“Tab”：表；“DB”：数据库；“Col”：列；“PK”：主键；“FK”：外键。

扰动数据	列级	表级	模式演化影响 SQL	多个数据库	种子数据	种子数据特征（平均值）				
						表/数据库	列/数据库	列/表	主键/数据库	外键/数据库
FootballDB [8]	-	减少主键/外键引用，减少 JOIN 路径	✓	✗	国际足联世界杯 [1]	15	107	7.1	-	16
Dr.Spider [2]	重命名	✗	✓	✓	Spider [2]	5.1	22.1	5.4	3.7	3.2
ADVETA [23]	添加；重命名	✗	✓	✓	Spider [2]	5.1	22.1	5.4	3.7	3.2
MT-TEQL [20]	添加；移除；打乱；重命名	拆分；合并；打乱	✗	✓	Spider [2]	5.1	22.1	5.4	3.7	3.2
EvoSchema（我们的）	添加；移除；重命名；拆分；合并	拆分；合并；重命名；添加；移除	✓	✓	BIRD [18]	7.3	72.5	10.6	6.5	9.3

合并表：我们选择两个或多个相关表，并将它们合并为一张表。使用 GPT-4 为合并后的表生成合适的名称，并将原表中的列整合到这张新表下。更具体地说，提示 GPT-4：1）在合并后将原表的所有主键列复制到新表中，但仅保留其中一列作为新表的主键，其余则作为普通列；2）如果这两个原表中的主键列相同，则合并后的新表中只保留一个；3）合并表时，如果原表中存在两个并非主键列但列名相同的列，则在将它们合并到新表时相应修改其列名，使它们彼此不同。最后，标准 SQL 由 GPT-4 相应更新。我们还会人工检查新黄金 SQL，以确保其正确。

质量控制：为确保 EvoSchema 中的数据质量，我们结合使用先进的语言模型和严格的人工验证。具体而言，在需要添加或重命名时，我们使用 GPT-3.5 生成合成的列名、表名和数据类型（仅针对列）。我们随机抽取 200 个生成的示例进行人工审查，结果表明，GPT-3.5 对输入上下文理解充分，能够有效生成符合要求的名称。对于拆分或合并表等更复杂的操作，我们利用能力更强的 GPT-4，以高准确率处理模式变更及相应的 SQL 修改。

为补充这些自动化流程，我们邀请了五位具备丰富 SQL 专业知识的标注人员，仔细审查涉及复杂模式转换的案例。标注人员验证了生成的标准 SQL 查询，并在必要时进行人工修正，以确保其正确无误且与修改后的模式一致。为进一步提高可靠性，我们采用交叉验证，让多位标注人员共同处理复杂案例，并通过讨论或达成共识来解决分歧。先进 AI 工具与细致人工审查相结合，确保 EvoSchema 成为稳健、准确的基准测试，忠实反映现实世界中的模式演变场景。

成本分析： 我们有 1.5K 个拆分表样例和 1.1K 个合并表样例需要进行人工核验。在拆分表样例中，1.1K 个相对简单，每个样例约需 3 分钟核验；其余 0.4K 个则更复杂，每个约需 7 分钟，总计约 100 小时。对于合并表样例，0.8K 个较简单（每个 3 分钟），0.3K 个较复杂（每个 7 分钟），总计约 75 小时。请注意，这些人工投入用于整理评估数据，而非训练数据。我们的训练数据完全自动生成，不涉及任何人工标注或人工核验。我们的分析还表明，大语言模型生成的拆分与合并表中约有 30% 的低质量数据，这凸显了对这两种类型进行仔细人工核验的必要性。

3.7 与现有基准测试的比较

EvoSchema（如表1所示）提出了一套全面且独特的分类体系，用于评估模式演化对 SQL 查询产生影响时模型的行为，这使其有别于 Dr.Spider [2], ADVETA [23], MT-TEQL [20] 和 FootballDB [8]等其他基准测试。Dr.Spider 和 ADVETA 仅关注列重命名和添加等有限扰动，而EvoSchema 则涵盖了范围更广的转换，包括在列级和表级进行添加、删除、重命名、拆分和合并。这种多样性使我们能够在现实且动态的模式演变场景下测试系统。此外，尽管 MT-TEQL 涵盖了多种扰动，但它只修改 SQL 中未提及的列，因此没有直接考虑模式演化对 SQL 的影响。EvoSchema 独特地将模式演化及其对 SQL 查询的影响纳入考量，从而能够在高度仿真实际数据库演化挑战的环境中评估模型。 与 FootballDB [8] 主要通过重构模式来减少表间外键映射和 SQL 的 JOIN 路径不同，我们定义了一套配置更灵活、系统性更强且结构化程度更高的模式演化分类体系。此外，我们提供的模式演变与合成框架使我们能够轻松探索多个数据库中的模式变更，而 FoodballDB 仅限于一个数据库。 最后，在种子数据选择方面，与常用于研究文本到 SQL 鲁棒性的 Spider 相比，BIRD 包含更复杂、真实且规模更大的数据库，以及更复杂的 SQL 查询，其中包含 Spider 中经常缺失的关键词。这些区别使EvoSchema 尤其适合研究系统如何适应不断演变的模式，推动研究超越以往基准测试中较为简单或不够全面的设置。

3.8 数据统计

表2概述了 EvoSchema 中的数据统计，展示了对原始 BIRD 数据集应用的各种扰动类型。“列操作”是指对原始 BIRD 数据中的列执行列级操作；“表操作”是指对原始 BIRD 数据中的表执行表级操作。所有扰动数据均通过对原始 BIRD 数据集执行列操作或表操作获得。“操作项目”显示被修改的列或表的数量。“每个查询中被操作的项目”是指模式中每个查询所涉及的被修改列数或表数

表2：EvoSchema 的数据统计。“原始”指原始 BIRD 数据集；“列操作”指对原始 BIRD 数据中的列执行列级操作；“表操作”指对原始 BIRD 数据中的表执行表级操作。“*”：用于计算执行准确率的评估数据。我们合成数据值，以重建模式演化后的数据库，并筛除无法通过标准 SQL 执行的项目，因此用于计算执行准确率的评估数据规模较小。

数据统计												
扰动类型	训练	评估	评估*	每个表中被操作的项目				每个查询中被操作的项目			
				Min	Mean	中位数	Max	Min	Mean	中位数	Max
原始	9426	1534	1068	-	-	-	-	-	-	-	-
				列操作							
添加列	9219	1506	846	1	5.7	3	83	1	5.9	4	43
删除列	9426	1534	1076	1	6.2	2	87	1	6.9	3	70
在 SQL 中删除列	9424	1534	-	1	2.5	2	8	1	2.5	2.5	6
重命名列	9385	1533	947	1	4.3	3	46	1	4.4	3	46
拆分列	140	37	37	1	2	2	4	1	2	2	4
合并列	148	44	44	2	3	3	4	2	3	3	4
				表操作							
添加表	9387	1530	1014	-	-	-	-	1	2	2	3
删除表	7212	1171	-	-	-	-	-	1	1	1	1
重命名表	9392	1534	1063	-	-	-	-	1	1.5	1	4
拆分表	9254	1515	824	-	-	-	-	1	2.6	3	5
合并表	6930	1139	569	-	-	-	-	2	2	2	2

针对每个 SQL 查询，具体操作的是与生成该查询相关的表。对于“拆分表”，“每个查询中被操作的项目”表示每个原表被拆分成的表数。对于“合并表”，“每个查询中被操作的项目”表示合并为单个表的表数。

4 训练范式

在我们的工作中，我们提出了一种新的训练范式，以增强模型应对不同模式演化的鲁棒性。对于每个<自然语言问题、相关模式、*SQL*>三元组，我们固定训练数据中的自然语言问题，并为每个三元组扩增不同的模式设计，这些设计可能会导致 SQL 发生变化，也可能不会。因此，每个原始三元组都可以衍生出多个三元组。我们通过学习多个模式设计和 SQL 到原始问题的映射关系来训练模型，这可以提升模型识别不同表和列与问题之间正确关系的能力，并更好地区分不同模式设计之间的差异。通过这一过程，模型能够更好地避免学习伪相关模式，从而增强其应对不同模式演化类型的鲁棒性。

5 实验设置

5.1 训练与评估设置

训练设置：我们选择四个开源模型：Code Llama-7B [25], Mistral-7B [12], Llama 3-8B [7] 和 SQLCoder-7B [4]，以及两个闭源模型：GPT-3.5 5 和 GPT-4 [22] 用于我们的实验。对于这四个开源模型，我们探索两种设置：1）不含扰动类型：模型在原始训练数据上训练，不引入任何扰动类型在训练过程中。2）含扰动类型：通过合并原始训练数据和扰动训练数据来训练模型。对于闭源模型，我们仅将其用于评估。

评估设置：对于所有闭源模型以及经过微调的开源模型，我们在两种设置下对其进行评估：1）无扰动类型：该设置使用标准的、未经改动的原始评估数据来评估模型性能。2）有扰动类型：模型在引入不同扰动的数据上进行评估。通过比较这两种设置下的模型性能，我们可以评估微调模型与 GPT 模型对 NL2SQL 中模式演化的适应能力。该设置对标准环境和扰动环境下的模型性能进行全面评估，从而能够针对不同模型和模式演化类型的鲁棒性与适应性进行详细分析。

5.2 评估指标

1）表匹配 F1：该分数是一项指标，用于衡量模型正确识别生成有效 SQL 查询所需相关表的能力。F1 分数是精确率与召回率的调和平均数，其中精确率是模型预测的所有表中被正确预测的表所占的百分比，召回率是所有本应被选中的实际表中被正确预测的表所占的百分比。表匹配 F1 分数综合这两项指标，提供均衡的评估，从而能够评估文本到 SQL 模型从数据库模式中正确识别所需表以形成准确查询的能力。更高的表匹配 F1 表明在为 SQL 查询选择正确表方面表现更好。

2) 列匹配 F1：该分数用于评估模型从自然语言输入生成有效 SQL 查询时，识别相关列的准确程度。与表匹配 F1 类似，它衡量精确率与召回率之间的平衡，但具体针对数据库中的列进行评估。列匹配 F1 分数越高，表明模型为 SQL 查询选择正确列的表现越好。

3) 执行准确率：该指标用于衡量预测的 SQL 查询在数据库上执行时，能否返回与标准 SQL 相同的正确结果。

5.3 训练与评估详情

我们选择 Code Llama-7B [25], Mistral-7B [12], Llama 3-8B [7]和 SQLCoder-7B [4] 作为开源基座模型。我们使用 Huggingface Transformers库 [31]对这些模型进行微调。对于扰动训练，我们合并所有扰动数据，并随机打乱，作为最终的训练数据。训练 Code Llama、Llama 3 和 SQLCoder 时，我们使用 2e-5 的学习率；训练 Mistral 时使用 5e-6 的学习率。批大小为 4。我们使用 4 个 A100 80GB GPU 训练所有模型，并采用余弦调度器，预热期为 0.03，共训练 6 个训练轮次。我们使用全分片数据并行 [37] 高效地训练模型。训练时的最大输入长度设为 1024，推理时的最大输出长度设为 500。推理时，我们使用 vLLM [31] 进行批量评估，并将批大小设为 16。推理在一块 80G A100 GPU 上进行。对于闭源大语言模型，我们使用

[4]https://huggingface.co/defog/sqlcoder-72

5 b- https://openai.com/chatgpt/

表 3：在 EvoSchema 上的评估。“含”：通过合并原始数据与所有扰动训练类型来训练模型；“不含”：模型仅使用原始训练数据进行训练。每个模型的最佳性能以粗体显示，红色 表示提升幅度更大。“-”：部分相关表已被移除，因此此处不应使用标准 SQL 计算指标。

扰动类型	Code Llama		Mistral		Llama 3		SQLCoder		GPT-3.5	GPT-4
	w/o	w/	w/o	w/	w/o	w/	w/o	w/		
表匹配 F1										
原始	89.77	**90.42**	89.58	**90.62**	**89.96**	89.53	89.69	**90.64**	87.28	88.98
添加列	89.73	**90.27**	89.65	**90.03**	89.08	**89.70**	89.30	**90.52**	86.35	88.12
删除列	89.82	**90.24**	89.89	**90.66**	**90.09**	89.82	89.81	**90.54**	87.18	88.87
重命名列	**85.28**	85.07	**84.32**	84.27	**83.74**	82.92	**85.32**	84.93	81.73	83.20
拆分列	83.78	**89.19**	83.78	**88.29**	81.08	**85.14**	86.49	**88.29**	81.44	86.31
合并列	**88.65**	87.23	87.23	**89.72**	**88.65**	86.17	87.23	87.23	83.17	89.36
添加表	57.88	**89.50**	57.67	**89.30**	55.11	**88.51**	57.44	**89.38**	83.54	85.79
删除表	-	-	-	-	-	-	-	-	-	-
重命名表	88.84	**90.32**	89.40	**90.56**	87.18	**89.14**	89.40	**90.48**	87.02	88.45
拆分表	71.99	**81.55**	66.12	**80.87**	71.08	**80.12**	72.52	**81.92**	77.52	80.68
合并表	85.29	**87.03**	83.39	**86.91**	81.68	**86.48**	84.80	**86.35**	83.04	86.99
宏平均	83.10	**88.08**	82.10	**88.12**	81.77	**86.75**	83.20	**88.03**	83.83	86.68
列匹配 F1										
原始	80.66	**81.64**	81.10	**82.36**	**79.13**	78.72	81.52	**81.97**	78.28	80.78
添加列	78.26	**80.27**	79.16	**80.18**	75.79	**76.87**	79.09	**80.46**	75.03	78.58
删除列	82.67	**82.75**	83.09	**84.00**	**81.56**	80.69	**83.20**	83.18	80.33	82.55
重命名列	76.50	**76.94**	76.35	**76.73**	**72.24**	71.07	76.84	**77.38**	73.40	75.90
拆分列	71.22	**81.81**	70.24	**80.41**	67.29	**75.04**	74.50	**79.92**	73.59	77.92
合并列	83.19	**83.30**	82.75	**83.41**	82.72	**83.68**	82.64	**83.31**	78.13	88.56
添加表	63.81	**81.14**	65.39	**81.09**	59.36	**77.96**	62.91	**81.23**	76.45	79.32
删除表	-	-	-	-	-	-	-	-	-	-
重命名表	79.60	**80.91**	80.32	**81.29**	**77.49**	77.46	80.77	**81.79**	77.78	80.04
拆分表	75.30	**78.45**	73.87	**78.11**	73.81	**73.95**	75.83	**78.59**	74.89	77.41
合并表	65.56	**67.09**	64.12	**67.46**	63.50	**64.40**	65.57	**67.29**	63.23	68.13
宏平均	75.68	**79.43**	75.64	**79.50**	73.29	**75.98**	76.29	**79.51**	75.11	78.92

Azure OpenAI API[6]。GPT-4 使用 2023-12-01-preview 版本，GPT-3.5 使用 2023-07-01-preview 版本。

5.4 基线方法

我们加入上下文学习 [10] 以及一种更先进的方法：CHESS [28] 作为基线方法进行全面比较。 为了测试上下文学习是否有助于解决模式演化问题，我们随机选取三个示例（每个示例由以下内容组成： <自然语言问题、演化后的数据库模式、模式演化后的标准 *SQL*> 三元组），将其作为提示中的示例，帮助模型理解演化后的模式（表 4）。我们还将一种先进的 NL2SQL 方法 CHESS 纳入基线方法。我们采用了该方法提出的模式选择（SS）和候选生成（CG）组件。对于模式选择，我们使用先进的 GPT-4o 模型对数据库模式进行剪枝，移除无关表以及所选表中的无关列，确保只有最相关的表和列会传入模型用于 SQL 生成。为确保公平比较我们主要的微调方法时，我们使用一个经过微调、且未使用任何模式扰动数据进行训练的 Code Llama 模型作为 SQL 生成模型。此设置使我们能够单独评估模式选择和剪枝组件在解决模式演化问题上的效果。结果见表 4。

6 结果与分析

6.1 主要结果

如表 3 和表 5 所示，我们使用原始 BIRD 训练数据（包含和不包含不同的扰动类型）训练 Codellama、Mistral、Llama3 和 SQLCoder，并在原始 BIRD 评估数据及不同的扰动类型上评估模型。我们观察到：

在不同扰动类型上训练的模型平均而言对模式变化更为鲁棒，并在表级模式演化上表现出较高的鲁棒性。尽管在训练期间加入扰动数据会导致原始未演化评估数据以及添加、删除和重命名列类型上的执行准确率（EX）略有下降，但它在列拆分和表级扰动类型上取得了显著更好的结果。

[6]https://learn.microsoft.com/en-us/azure/ai-services/openai/reference

通过比较这四个模型在有无扰动数据时的性能，我们观察到：对于列拆分，使用扰动数据训练的模型在表匹配 F1 上最多可提升 5.4 分，在列匹配 F1 上提升 10.6 分，在执行准确率上提升 24 分；对于添加表，使用扰动数据训练的模型在表匹配 F1 上最多可提升 33 分，在列匹配 F1 上提升 18 分，在执行准确率上提升 19 分；对于拆分表，使用扰动数据训练的模型在表匹配 F1 上最多可提升 14 分，在列匹配 F1 上提升 4.2 分，在执行准确率上提升 12 分；对于合并表，在扰动数据上训练的模型在表匹配 F1 上最多可提升 4.8 分，在列匹配 F1 上提升 3 分。我们假设这是因为扰动增强数据对于处理大幅模式变更特别有益，但可能会引入轻微

表 4：对 EvoSchema 的人工评估。“ZS”指零样本，即在不提供任何示例的情况下提示模型。“ICL”指上下文学习，即向模型提供三个演示示例。“w/o”表示对不含扰动训练数据的模型进行微调；“w/”表示对含扰动训练数据的模型进行微调。粗体颜色表示每行中的最佳表现。

对 EvoSchema 的人工评估					
扰动类型	GPT-4		Code Llama		$CHESS_{SS+CG}$
	零样本	上下文学习	w/o	w/	
原始	62	58	**65**	64	63
添加列	59	55	62	61	**66**
删除列	65	61	**66**	63	64
重命名列	57	56	57	57	**62**
拆分列	46	59	41	**62**	49
合并列	68	66	70	**70**	66
添加表	56	55	46	**62**	57
删除表	-	-	-	-	-
重命名表	58	60	**64**	61	61
拆分表	57	53	48	**60**	53
合并表	55	57	54	**58**	53
宏平均	58	58	57	**62**	59

表5：执行准确率（EvoSchema）。“含”：模型使用所有扰动类型进行训练；“不含”：模型仅在原始训练数据上训练。

EvoSchema上的执行准确率										
扰动类型	Code Llama		Mistral		Llama 3		SQLCoder		GPT-3.5	GPT-4
	w/o	w/	w/o	w/	w/o	w/	w/o	w/		
原始	58	57	59	58	55	51	58	58	44	47
添加列	57	55	56	56	52	49	55	57	43	46
删除列	59	57	60	58	56	53	60	58	45	47
重命名列	54	52	55	54	49	47	56	55	43	45
拆分列	41	**62**	35	**54**	38	**49**	43	**67**	41	46
合并列	70	70	70	70	73	73	66	**82**	61	68
添加表	40	**58**	39	**58**	37	**52**	40	**57**	44	48
删除表	-	-	-	-	-	-	-	-	-	-
重命名表	56	55	55	56	52	50	56	55	43	47
拆分表	38	**46**	36	**48**	40	41	43	**49**	40	47
合并表	43	45	45	46	42	44	47	46	37	45
宏平均	52	**56**	51	**56**	49	**51**	52	**58**	44	49

在较简单的模式变更中，未使用扰动数据训练的模型已经最大程度地学会了相关模式，因此这些扰动带来的噪声影响有限。为了更好地理解较简单列级扰动下的轻微性能差距，我们进行了错误分析和案例研究，比较使用和未使用扰动数据训练的模型。我们发现有两类错误导致了这一现象：（1）WHERE子句中条件多余或缺失。例如，对于问题“在非致癌分子中，原子 ID 为 TR004_7 的元素是什么？”，使用扰动数据训练的模型（“w/”）漏掉了 WHERE子句中的条件 T2.label = ’-’，而“w/o”模型则正确地包含了该条件。然而，对于另一个问题“2012/8/26 上午有多少笔交易以 CZK 支付？”，“w/”模型引入了一个多余的 WHERE 条件：T1.TransactionID BETWEEN 1 AND 1000，而该条件并不属于标准 SQL。（2）SELECT 或 WHERE 子句中的列选择错误。例如，对于问题“在门诊随访的患者中，有多少人的碱性磷酸酶水平正常？”，“w/”模型在 WHERE子句中预测了 T1.Description，而不是 T1.Admission；“w/o”模型则选择了正确的列。同样，对于问题“超级英雄 A-Bomb 属于哪个群体？”，“w/”模型选择了 T2.team affiliation，而不是正确的 T2.race。这些例_子表明，使用扰动数据训练虽然能够提升整体鲁棒性，尤其有助于应对较大的模式变更，但也可能引入少量噪声，导致模型在较简单的扰动下错误地选择条件或列。

闭源模型总体上对不同类型的模式演化具有鲁棒性。如表 3 和表 5 所示，我们比较了 GPT 模型以及在有无扰动类型条件下训练的四个开源模型的模型性能。我们观察到：与原始未演化测试集相比，GPT 模型在不同扰动类型上的性能相对稳定。相比之下，未使用扰动训练数据进行微调的开源模型出现了显著的性能下降——尤其是在拆分列、添加表、拆分表和合并表上——这些会引入更大的模式变更。我们假设，闭源模型的稳定性与鲁棒性源于更广泛的预训练接触以及更强的内部模式推理能力，而未使用扰动类型训练的开源模型由于在多样化模式变化上的训练有限而更为敏感。这促使我们需要使用扰动训练数据对开源模型进行微调，以提升其在模式演化下的泛化能力。我们注意到，在比较开源大语言模型与闭源大语言模型的模型性能时，使用扰动数据训练的模型在列级和表级扰动评估数据上的表现均优于 *GPT* 模型。这表明，我们使用扰动数据训练的模型比 GPT 模型更具鲁棒性。

表级扰动对模型性能的影响大于列级扰动。如表 3 和表 5 所示，与原始评估数据上的性能相比，添加表和拆分表会导致表匹配 F1 显著下降；添加表、拆分表和合并表会导致列匹配 F1 显著下降。这一现象表明，添加表或拆分表容易使模型在选择正确的表以生成 SQL 查询时感到困惑。对于合并表，尽管模型能够正确选择表，

但当来自不同表的列合并到同一张表中时，模型要选出正确的列会有些困难。至于列级性能，除了拆分列之外，与原始数据上的性能差异有限。**降低表模式复杂度有利于提升模型性能**。比较列级扰动评估数据和原始评估数据上的模型性能可知，添加列会导致列匹配 F1 下降，而删除列则会使列匹配 F1 上升。这表明，更简单的表模式有利于模型选择列，因为删除列会简化表模式，而添加列会使表模式更加复杂。

6.2 不同基线方法的比较

由于 EvoSchema 的测试集规模较大，而且我们需要分别调用 GPT-4 和 GPT-4o API 进行上下文学习和 CHESS 评估，为节省成本，我们从原始 BIRD 测试集及每种扰动类型中随机选取 200 个示例，以比较不同的基线方法。我们在下采样测试集上比较了 GPT-4 零样本提示、GPT-4 3-shot 上下文学习、使用和未使用扰动训练数据训练的 CodeLLama，以及 CHESS（含模式选择（SS）和候选生成（CG））。由于我们发现，当不同 SQL 查询有时会产生相同结果但与自然语言问题不一致，或者标准 SQL 和错误的预测 SQL 有时都返回空结果时，执行准确率仍可能出错并误导评估，因此我们在此采用人工评估，以获得更准确的评估结果。如表 4 所示，与 GPT-4 零样本（ZS）相比，上下文学习（ICL）仅在拆分列扰动上表现出显著优势，在其他类型上则略有提升或下降。这表明 ICL 在处理模式演化时并非始终有效。我们推测，这是因为 ICL 中的演示示例无法涵盖模式和 SQL 变更的全部范围；因此，对于与演示示例差异较大的示例，ICL 的帮助有限。不过，对于拆分列而言，变更通常涉及姓名、地址或日期拆分等模式，因此演示示例具有更好的泛化能力，使 ICL 在这种情况下更有效。对于 CHESS，我们使用强大的闭源模型 GPT-4o 进行模式选择和剪枝，并使用未经过扰动训练的 Code Llama（CodeLlama 不含）作为 SQL 生成模型。CHESS 在添加列和重命名列上表现最佳，并且在拆分列、添加表以及平均表现上显著优于 CodeLlama 不含。这凸显了准确的模式选择和剪枝对于提升 SQL 生成效果的重要性。然而，我们也发现，剪枝阶段的错误可能会传播，导致性能下降。具体而言，在合并列和合并表的情况下，CHESS 往往会过度剪枝，遗漏相关模式信息，因而表现不如 CodeLlama 不含。最后，我们发现仍有必要使用扰动训练数据对 CodeLlama 进行微调，因为该方法在所有评估数据类型上的平均表现优于所有基线方法，并且在“拆分列”“添加表”“拆分表”和“合并表”类型上显著优于其他方法。我们采用麦克内马尔检验 [21] 来衡量我们的方法与之间性能差异的统计显著性。

表 6：EvoSchema 上的扰动类型消融。基础模型为 Code Llama。“两者”：模型使用列级扰动和表级扰动类型进行训练；“不含表级扰动”：模型训练时未使用表级扰动类型；“不含列级扰动”：模型训练时未使用列级扰动类型。

扰动类型消融						
扰动类型	表匹配 F1			列匹配 F1		
	both	不含表扰动	不含列-p	both	不含表-p	不含列-p
原始	90.73	90.80 (+0.07)	90.04 (-0.69)	81.09	82.15 (+1.06)	80.49 (-0.60)
添加列	90.86	90.80 (-0.06)	89.75 (-1.11)	79.63	80.81 (+1.18)	77.29 (-2.34)
删除列	90.72	90.83 (+0.11)	90.48 (-0.24)	83.28	83.85 (+0.57)	82.61 (-0.67)
重命名列	85.35	85.38 (+0.03)	84.57 (-0.78)	76.49	77.53 (+1.04)	75.17 (-1.32)
添加表	88.95	58.94 (-30.01)	88.57 (-0.38)	79.87	64.11 (-15.76)	79.33 (-0.54)
删除表	-	-	-	-	-	-
重命名表	90.54	90.77 (+0.23)	89.29 (-1.25)	81.13	81.51 (+0.38)	79.33 (-1.80)
拆分表	80.71	73.28 (-7.43)	79.05 (-1.66)	77.41	75.95 (-1.46)	76.30 (-1.11)
合并表	88.72	87.87 (-0.85)	86.83 (-1.89)	68.40	68.26 (-0.14)	67.08 (-1.32)

表 7：EvoSchema 上的范围外效应。基础模型为 Code Llama。“不含”：模型在训练时未使用扰动类型；“含”：模型使用原始数据和所有扰动类型进行训练；“+分布外”：模型使用原始数据、扰动类型和两种范围外（分布外）扰动类型进行训练；“+ 分布外 FP”：使用两种分布外扰动类型训练的模型，在原始数据和范围内扰动数据上做出了错误预测；“+ 分布外 TP”：使用两种分布外扰动类型训练的模型，在两种分布外扰动数据上做出了正确预测；“表”：由于缺少表信息，模型拒绝预测 SQL；“列”：由于缺少列信息，模型拒绝预测 SQL。

范围外效应										
扰动类型	表匹配 F1			列匹配 F1			+ 分布外假阳性		+ 分布外真阳性	
	w/o	w/	+ 分布外	w/o	w/	+ 分布外	Tab	Col	Tab	Col
原始	89.77	90.42	82.98 (-7.44)	80.66	81.64	75.43 (-6.21)	7.11	0.65	-	-
添加列	89.73	90.27	86.07 (-4.20)	78.26	80.27	77.00 (-3.27)	4.25	0.40	-	-
删除列	89.82	90.24	82.24 (-8.00)	82.67	82.75	75.90 (-6.85)	7.56	0.72	-	-
在 SQL 中删除列	-	-	-	-	-	-	5.02	-	-	84.03
重命名列	85.28	85.07	80.20 (-4.87)	76.50	76.94	73.04 (-3.90)	4.44	0.20	-	-
添加表	57.88	89.50	88.78 (-0.72)	63.81	81.14	80.71 (-0.37)	0.33	0.07	-	-
删除表	-	-	-	-	-	- -	-	1.62	83.86	-
重命名表	88.84	90.32	86.36 (-3.96)	79.60	80.91	78.06 (-2.85)	3.52	0.39	-	-
拆分表	71.99	81.55	81.07 (-0.48)	75.30	78.45	78.02 (-0.43)	0.26	0.07	-	-
合并表	85.29	87.03	82.18 (-5.15)	65.56	67.09	63.59 (-3.50)	4.65	0.35	-	-

每种基线方法。我们使用 statsmodels 软件包计算 p 值，当 $p < 0.05$时，认为差异具有统计显著性，这表明性能提升不太可能是由随机因素造成的。通过此检验，我们发现，与三个关键基线方法相比，我们的方法取得了具有统计显著性的提升：GPT-4 上下文学习、未使用扰动数据进行微调，以及 CHESS（均满足 $p < 0.05$）。

6.3 扰动类型的影响

我们探究了列级扰动类型和表级扰动类型的影响。如表 6 所示，我们使用列级和表级扰动类型训练模型，并将其与未使用列级扰动类型和未使用表级扰动类型训练的模型进行比较。从我们的

实验中，我们发现，在训练时不使用表级扰动的情况下，模型在列级扰动类型上的性能略优于使用列级和表级扰动类型训练的模型，但在表级扰动类型上的性能则会显著下降。这表明，表级扰动数据对列级扰动类型的影响有限，却会对表级扰动类型产生巨大影响。观察仅使用表级扰动类型训练的模型时，我们发现，该模型在列级和表级扰动类型上的性能均有所下降。这表明，列级扰动类型仍有助于模型训练。

表 8：无关表效应。“含”：模型使用所有扰动类型进行训练；“不含”：模型仅使用原始训练数据进行训练；“不含+"：模型仅使用原始训练数据进行训练，但对于输入表模式，我们还添加了无关表。

添加无关表效应						
扰动类型	表匹配 F1			列匹配 F1		
	w/o	w/o+	w/	w/o	w/o+	w/
原始	89.77	87.65	**90.42**	80.66	79.24	**81.64**
添加列	89.73	86.35	**90.27**	78.26	75.31	**80.27**
删除列	89.82	87.30	**90.24**	82.67	80.74	**82.75**
重命名列	**85.28**	81.90	85.07	76.50	73.28	**76.94**
添加表	57.88	88.01	**89.50**	63.81	79.51	**81.14**
删除表	-	-	-	-	-	-
重命名表	88.84	86.84	**90.32**	79.60	78.47	**80.91**
拆分表	71.99	67.27	**81.55**	75.30	70.39	**78.45**
合并表	85.29	83.56	**87.03**	65.56	63.59	**67.09**

6.4 范围外类型的影响

我们评估范围内和范围外两种情形。在范围内的设置中，模式变更可能会改变标准 SQL，也可能不会。范围外情形涉及两种特殊扰动：（1）移除标准 *SQL* 中使用的列；（2）移除标准 *SQL* 中使用的表。在这两种情形下，模式都缺少关键信息，因此预期模型应当放弃生成查询。

为了评估这些扰动的影响，我们在一个组合数据集上训练模型，该数据集包含范围外和范围内的扰动类型，以及原始训练数据。我们将该模型与仅使用原始数据或范围内数据训练的其他模型进行比较。如表 7 所示，纳入范围外类型会导致模型在原始和范围内评估集上的性能均有所下降。

错误分析显示，使用范围外数据训练的模型倾向于做出更保守的预测，有时即使标准 SQL 有效也会弃答。进一步分析表明，假阳性（FP）率与使用和未使用范围外数据训练的模型之间的性能差距高度吻合，这证实了保守性增强是造成性能下降的主要原因。此外，对于范围外扰动，真阳性（TP）率仅约为 84%，这意味着即使不应存在 SQL，模型仍有 16% 的概率会做出预测。

6.5 无关表的影响

我们观察到，使用扰动类型训练的模型对表级扰动表现出显著的鲁棒性，例如添加表和拆分表。通过分析错误，我们发现未使用扰动类型训练的模型往往会预测出连接所有可用表的 SQL 查询，即使某些表与自然语言问题和 SQL 无关。我们推测，这是因为在不使用扰动进行训练时，模型只会接触到相关的表模式，从而学会了总是尝试连接所有输入表的伪相关模式。

为了探究仅添加无关表能否取得与使用扰动数据训练的模型相近的性能，我们在 BIRD 上训练了 CodeLlama 并开展实验。如表 8 所示，添加无关表在“添加表”扰动类型上取得了相近的性能，但在其他扰动类型上导致性能下降。这表明，要训练出更稳健的模型，必须结合所有扰动数据。

表 9：库内效应。该实验强调，训练和评估是在同一数据库内进行的，而非跨数据库进行。

库内效应				
扰动类型	表匹配 F1		列匹配 F1	
	w/o	w/	w/o	w/
原始	87.24	**87.43**	79.54	**80.89**
添加列	87.14	**87.43**	76.36	**78.92**
删除列	**87.29**	87.27	81.14	**81.29**
重命名列	85.71	**86.43**	77.45	**79.09**
添加表	61.13	**83.95**	66.11	**78.57**
删除表	-	-	-	-
重命名表	86.33	**86.67**	79.44	**79.96**
拆分表	71.82	**78.52**	75.09	**77.42**
合并表	85.11	**87.44**	71.43	**74.72**

6.6 数据库内与跨数据库的影响

我们假设，在相同数据库上训练的模型不仅能够学习模式演化规律，还会熟悉特定的表名和列名。为检验这一点，我们将 BIRD 训练数据拆分为训练集和测试集，确保测试集中的每个数据库也都出现在训练集中。我们使用 Code Llama 作为基础模型。表 9 的结果表明，对于大多数扰动类型，与第 6.1 节的跨数据库场景相比，模型的性能提升更为显著，这验证了我们的假设。

7 结论

总之，我们提出了自适应文本到 SQL 系统中模式演化这一关键挑战，并推出了 EvoSchema。这是一个全面、多样且独特的基准测试，专为研究和解决这一问题而设计。我们构建了一个结构化的模式演化类型分类法，通过列级和表级扰动合成逼真的模式设计。基于这一分类法，我们构建了一个评测基准，用于严格评估模式变更下的模型鲁棒性；此外，我们还提出了一种新颖的训练范式，将多样的模式设计融入现有的<自然语言问题、相关模式、*SQL*> 三元组中进行训练，从而提升应对模式演化的鲁棒性。

致谢

作者谨向 OSU NLP 小组的同事致谢，感谢他们提出富有见地的讨论和建设性建议，并感谢所有匿名审稿人提供的宝贵意见。

参考文献

[1] Andre Becklas. 2018年. 国际足联世界杯：历届世界杯的所有结果。Kaggle（2018年）。https://www.kaggle.com/datasets/abecklas/fifa-world-cup [2] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexan- der Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, 和 Bing Xiang. 2023年. Dr.Spider：面向文本到 SQL 鲁棒性的诊断评测基准。在第十一届国际学习表征会议。https://openreview.net/forum?id=Wc5bmZZU9cy [3] Anthony Cleve, Maxime Gobert, Loup Meurice, Jerome Maes, 和 Jens Weber. 2015. 理解数据库模式演化：案例研究。计算机编程科学 97（2015），113–121。[4] Daiga Deksne 和 Raivis Skadiņš. 2022. 用于以自然语言查询数据库的虚拟助手。在未来技术会议论文集。Springer，555–564。[5] Julien Delplanque, Anne Etien, Nicolas Anquetil, 和 Olivier Auverlot. 2018年. 关系数据库模式演化：一项工业案例研究。在 2018年 IEEE软件维护与演化国际会议（ICSME）。635–644。https://doi.org/10.1109/ICSME.2018.00073[6] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, 和 Matthew Richardson. 2021. 面向文本到 SQL 的结构基础预训练。在 2021年计算语言学协会北美分会会议：人类语言技术论文集。计算语言学协会。https://doi.org/10. 18653/v1/2021.naacl-main.105[7] Abhimanyu Dubey 和等人. 2024 年 . Llama 3 模型家族。arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783[8] Jonathan Fürst, Catherine Kosten, Farhad Nooralahzadeh, Yi Zhang, 和 Kurt Stockinger. 2025年. 基于真实用户查询评估文本到 SQL 系统的数据模型鲁棒性。在 EDBT。158–170。https://doi.org/10.48786/edbt. 2025.13[9] Chang Gao, Bowen Li, Wenxuan Zhang, Wai Lam, Binhua Li, Fei Huang, Luo Si, 和 Yongbin Li. 2022年. 迈向通用且鲁棒的文本到 SQL 解析。在计算语言学协会 EMNLP 2022 论文集：Findings, Yoav Goldberg、Zornitsa Kozareva 和 Yue Zhang（编）。计算语言学协会，阿布扎比，阿联酋，2113–2125。https://doi.org/10.18653/v1/2022.findings-emnlp.155[10] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, 和 Jingren Zhou. 2024年. 大语言模型赋能的文本到 SQL：一项基准评测。VLDB 基金会会议录 17，5（2024年），1132–1145。[11] Andrea Hillenbrand 和 Uta Störl. 2021. 管理 NoSQL 数据库中的模式迁移：顾问启发式方法与自适应模式迁移策略。在模型驱动工程与软件开发国际会议。Springer，230–253。[12] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, De- vendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, 和 William El Sayed. 2023年. Mistral 7B。arXiv:2310.06825 [cs.CL] https: // arxiv.org/abs/2310.06825[13] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao 等. 2021. Wilds：真实环境分布偏移基准测试。在国际机器学习会议。PMLR，5637–5664。[14] Kunal Kumar 和 S. K. Azad. 2017. 数据库规范化设计模式。在 2017年 IEEE北方邦分会电气、计算机与电子国际会议（UPCON）。318–322。https://doi.org/10.1109/UPCON. 2017.8251067[15] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, 和 Nan Tang. 2024年. 自然语言转 SQL 的黎明：我们准备好了吗？arXiv 预印本 arXiv:2406.01265（2024年）。[16] Guoliang Li, Xuanhe Zhou, 和 Xinyang Zhao. 2024年. 面向数据管理的大语言模型。VLDB 基金会会议录 17，12（2024年），4213–4216。[17] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, 和 Hong Chen. 2024年. Codes：致力于构建用于文本到 SQL 的开源语言模型。ACM 数据管理会议录 2，3（2024年），1–28。[18] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo 等. 2024年. 大语言模型是否已能充当数据库接口？面向大规模、以数据库为基础的文本到 SQL 的大型基准测试。神经信息处理系统进展 36（2024年）。[19] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, 和 Yuyu Luo. 2024年. 大语言模型 NL2SQL 综述：现状如何，未来走向何方？arXiv 预印本 arXiv:2408.05109（2024年）。

[20] 马平川和王帅。2021年。MT-teql：评估并增强神经自然语言数据库在真实世界语言和模式变化下的表现。《VLDB Endowment 论文集》15卷3期，2021年，569–582，10 14778 3494124（11月），–。https://doi.org/ / 。3494139 [21] 1947 Quinn McNemar。。关于相关比例或百分比差异抽样误差的注记。《Psychometrika》12卷2期（1947年），153–157。[22] OpenAI。2024年。GPT-4 技术报告。arXiv:2303.08774 [cs.CL] https://arxiv. org/abs/ 2303 08774 。皮欣宇、王兵、高岩、郭佳琪、李周军和娄建光。2022年。提升文本到 SQL 模型抵御自然和现实对抗性表扰动的鲁棒性。载于《计算语言学协会第60届年会论文集》（第1卷：长文），Smaranda Muresan、Preslav Nakov 和 Aline Villavicencio（编）。计算语言学协会，爱尔兰都柏林，2007–2022。https://doi.org/10.18653/v1/ 2022 142 .acl-long。Joaquin Quionero-Candela、杉山将、Anton Schwaighofer 和 Neil D. Lawrence。2009年。《机器学习中的数据集偏移》。麻省理工学院出版社。[25] Baptiste Rozière、Jonas Gehring、Fabian Gloeckle、Sten Sootla、Itai Gat、Xiaoqing Ellen Tan、Yossi Adi、刘景宇、Romain Sauvestre、Tal Remez、Jérémy Rapin、Artyom Kozhevnikov、Ivan Evtimov、Joanna Bitton、Manish Bhatt、Cristian Canton Ferrer、Aaron Grattafiori、熊文翰、Alexandre Défossez、Jade Copet、Faisal Azhar、Hugo Touvron、Louis Martin、Nicolas Usunier、Thomas Scialom 和 Gabriel Synnaeve。2024年。Code Llama：面向代码的开放基础模型。arXiv:2308.12950 [cs.CL] https://arxiv.org/abs/2308.12950[26] 宋业伟、Saad Ezzini、唐训竹、Cedric Lothritz、Jacques Klein、Tegawendé Bissyandé、Andrey Boytsov、Ulrick Ble 和 Anne Goujon。2024年。增强面向金融系统设计的文本到 SQL 翻译。载于《第46届国际软件工程会议：软件工程实践论文集》。252–262。[27] 戴昌宇、陈子儒、张天舒、Xiang Deng 和 Huan Sun。2023年。探索用于文本到 SQL 的思维链风格提示。载于《2023年自然语言处理实证方法会议论文集》，Houda Bouamor、Juan Pino 和 Kalika Bali（编）。计算语言学协会，新加坡，5376–5393。https://doi.org/10.18653/v1/2023.emnlp-main.327[28] Shayan Talaei、Mohammadreza Pourreza、张育诚、Azalia Mirhoseini 和 Amin Saberi。2024年。CHESS：用于高效 SQL 合成的上下文利用框架。arXiv:2405.16755 [cs.LG] https://arxiv.org/abs/2405.16755[29] Bailin Wang、Richard Shin、刘晓东、Oleksandr Polozov 和 Matthew Richardson。2020年。RAT-SQL：面向文本到 SQL 解析器的关系感知模式编码与链接。载于《计算语言学协会第58届年会论文集》，Dan Jurafsky、Joyce Chai、Natalie Schluter 和 Joel Tetreault（编）。计算语言学协会，线上，7567 7578 10 18653 1 2020 677 – 。https://doi.org/ /v / .acl-main。[30] 王成龙、Kedar Tatwawadi、Marc Brockschmidt、黄博森、毛毅、Oleksandr Polozov 和 Rishabh Singh。2018年。通过执行引导解码实现鲁棒的文本到 SQL 生成。arXiv:1807.03100 [cs.CL] https://arxiv.org/abs/1807.03100[31] Thomas Wolf、Lysandre Debut、Victor Sanh、Julien Chaumond、Clement Delangue、Anthony Moi、Pierric Cistac、Tim Rault、Rémi Louf、Morgan Funtowicz、Joe Davison、Sam Shleifer、Patrick von Platen、Clara Ma、Yacine Jernite、Julien Plu、徐灿文、Teven Le Scao、Sylvain Gugger、Mariama Drame、Quentin Lhoest 和 Alexander M. Rush。2020年。HuggingFace 的 Transformers：先进的自然语言处理工具。arXiv:1910.03771 [cs.CL] https://arxiv.org/abs/1910.03771[32] 于涛、张瑞、杨凯、Michihiro Yasunaga、王东旭、李子凡、James Ma、李艾琳、姚清宁、Shanelle Roman、张子林和Dragomir Radev。2018年。Spider：用于复杂跨领域语义解析和文本到 SQL 任务的大规模人工标注数据集。载于《2018年自然语言处理实证方法会议论文集》，Ellen Riloff、David Chiang、Julia Hockenmaier 和 Jun'ichi Tsujii（编）。计算语言学协会，比利时布鲁塞尔，3911–3921。https://doi.org/10.18653/v1/D18-1425[33] 张斌、叶宇骁、杜国庆、胡晓茹、李志帅、杨孙、刘驰、赵睿、李子悦和毛航宇。2024年。大语言模型文本到 SQL 能力基准测试：综合评估。arXiv:2403.02951 [cs.CL] https://arxiv.org/abs/2403.02951[34] 张超、毛玉人、范一江、米宇、高云军、陈璐、娄东方和林金书。2024年。FinSQL：面向金融分析的模型无关、基于大语言模型的文本到 SQL 框架。arXiv 预印本 arXiv:2401.10506（2024年）。[35] 张汉冲、曹瑞生、陈璐、徐宏深和余凯。2023年。ACT-SQL：利用自动生成的思维链进行文本到 SQL 上下文学习。载于《2023年自然语言处理实证方法会议》。https://openreview.net/forum?id=oeZiXoCHgq [36] 张天舒、刘长昌、Wei-Han Lee、苏瑜和Huan Sun。2023年。用于语义解析的联邦学习：任务定义、评估设置与新算法。arXiv: 2305 17221 [ ] 2305 17221 . cs.CL https://arxiv.org/abs/ 。[37] 赵艳丽、Andrew Gu、Rohan Varma、罗亮、黄建钦、徐敏、Less Wright、Hamid Shojanazeri、Myle Ott、Sam Shleifer、Alban Desmaison、Can

Balioglu、Pritam Damania、Bernard Nguyen、Geeta Chauhan、Yuchen Hao、Ajit Mathews 和 Shen Li。2023年。PyTorch FSDP：扩展全分片数据并行的实践经验。arXiv:2304.11277 [cs.DC] https://arxiv.org/abs/2304.11277

[38] Alex Zhuang、Ge Zhang、Tianyu Zheng、Xinrun Du、Junjie Wang、Weiming Ren、Stephen W Huang、Jie Fu、Xiang Yue 和 Wenhu Chen。2024年。StructLM：迈向构建用于结构化知识落地的通用模型。arXiv 预印本 arXiv:2402.16671（2024年）。