arXiv:2603.04176v1 [cs.DB] 4 Mar 2026

大型上下文图的可扩展连接推断

Shivani Tripathi、Ravi Shetye、Shi Qiao、Alekh Jindal research@tursio.ai

Tursio，贝尔维尤，美国

摘要

上下文图对于现代人工智能应用至关重要，包括问答、模式发现和数据分析。要从结构化数据库构建准确的上下文图，需要推断实体之间的连接关系。无效连接会引入歧义和重复记录，损害图质量。我们提出了一种可扩展的连接推断方法，将统计剪枝与大语言模型（LLM）推理相结合。不同于纯粹基于统计的方法，我们的混合方法模仿人类语义理解，同时通过数据驱动推断减少大语言模型的幻觉。我们首先识别主键候选项，并使用大语言模型进行判定；随后采用相同的两阶段流程检测包含依赖。这种统计方法与大语言模型的结合能够扩展至大型数据库模式，同时保持准确率并尽可能减少假阳性。我们还利用数据库查询历史，随着查询工作负载的变化，持续优化连接推断。在 TPC-DS、TPC-H、BIRD-Dev 和生产工作负载上的评估表明，该方法在结构良好的模式上具有较高的精确率（78–100%），同时也凸显了在规范化程度较低的环境中进行连接发现本身的难度。

1 引言

上下文图对于在企业数据上进行推理正变得日益重要，例如提出问题、发现模式、关联观察结果、追踪血缘等。它们能够捕捉记录系统所遗漏的连接并消除歧义，有望成为新时代人工智能代理 [10]持久的上下文层。对于结构化数据库，要构建高保真的上下文图，就需要识别数据中各个实体之间的连接关系，这些关系称为连接推断。错误的连接推断会造成歧义，甚至生成重复数据，使上下文图变得不可靠。传统上，连接关系是在数据库模式中手动定义的。然而，随着组织中的数据库迅速增长，这种做法变得难以管理 [15]。因此，新兴方法会自动推断连接关系 [12]。遗憾的是，这些方法需要考虑呈指数级增长的列组合空间，因此难以扩展到生产环境中的数据库模式。

为说明这一挑战，图1展示了过去一年中接入 Tursio 的84个真实世界多表模式中，连接推断候选项的估计数量。我们将连接推断候选项的数量估算为表对数量与每个表中维度列平均数量的乘积，即对于总共有 d 个维度的 t 个表，$\binom{t}{2} \cdot (d/t)$。如图1所示，许多模式的连接推断候选项多达数十万个。这些候选项需要通过实际数据进行验证，以识别有效连接关系，这使得不加筛选地逐一考虑所有候选项

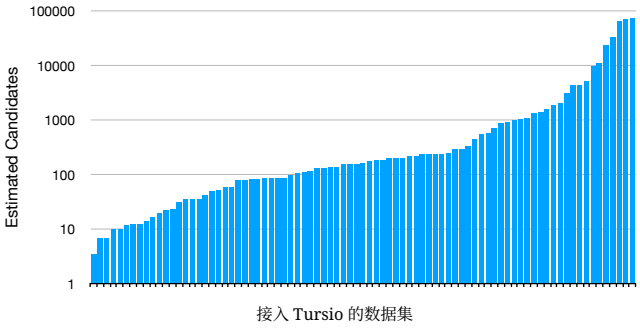


图1：展示接入 Tursio 的84个真实世界模式中连接推断候选项的估计数量。

实际数据来识别有效连接关系，因此无法天真地考虑所有这些关系。

本文提出一种可扩展的方法，用于推断大型上下文图中的连接关系。与完全依赖统计信息的以往方法不同，我们利用大语言模型（LLM）作出语义上显而易见的判断，也就是模拟人类的判断方式。然而，大语言模型容易产生幻觉，因此我们仍利用统计信息来剪枝搜索空间，并为大语言模型提供高质量的推理输入。具体而言，我们首先确定一组经过剪枝的主键候选项，然后请大语言模型进行裁定。随后，我们确定一组经过剪枝的包含依赖候选项，即键-外键关系，并再次使用大语言模型对其进行裁定。这种结合基于统计的剪枝与基于大语言模型的判定的双管齐下方法，使我们能够扩展到包含许多表和列的大型数据库模式。该方法还确保只向大语言模型提供相关上下文，从而以较低的误报率实现高质量的连接推断。我们还利用同一数据库上所有其他工作负载的查询历史，随着时间推移进一步完善判定结果。

总之，我们提出了一种面向大型上下文图的可扩展连接推断方法，将基于统计的剪枝与基于大语言模型的判定相结合（第3节）。我们介绍了如何利用上述双管齐下的方法推断主键和包含依赖（第4节和第5节）。我们展示了系统如何根据推断出的包含依赖生成左连接树，从而避免查询处理过程中的数据重复（第6节）。随着工作负载的演变，我们利用查询历史不断改进连接推断，并处理各种细微情况（第7节和第8节）。最后，我们在真实世界模式上进行了全面评估，结果表明，该方法在保持高准确率和低假阳性的同时，可扩展至大型模式（第10节）。下文将先讨论相关工作，再详细介绍上述各个组成部分。

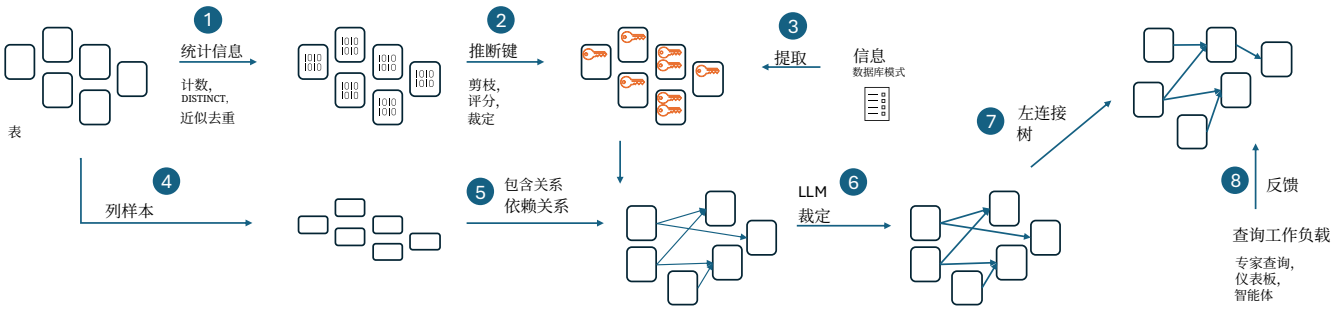


图 2：Tursio 中可扩展连接推断概览。

2 相关工作

数据集成和跨表关系发现是一个长期存在的问题 [11, 12, 15],，需要推断主键和外键——它们分别是唯一列组合（UCC）和包含依赖（IND）的特殊情况 [1]。

数据剖析。 Abedjan 等人 [1] 综述了函数依赖、UCC 和包含依赖的数据剖析技术。Metanome [24, 25]提供了可扩展的发现框架，而 HyUCC [27] 通过混合格遍历扩展了 UCC 发现的规模。

外键检测。 SPIDER [2] 通过单次扫描枚举列对包含依赖；BINDER [26] 将其扩展到 n 元包含依赖；Saw- fish [16] 通过相似性依赖处理脏数据。机器学习方法利用语义特征将包含依赖分类为外键 [17,29, 33], , 并扩展至全局模式 [23]和交互式发现 [5]。Jiang 和 Naumann [12] 通过对 UCC 和包含依赖进行评分，联合检测主键和外键——这与我们的方法最为接近；而 Koehler 和 Link [19] 则区分意外的 key 与有意义的键。

模式匹配与连接发现。模式匹配已有充分综述 [28], , Valentine [21] 对匹配技术进行了基准测试。在数据湖中，Auru m [8] 根据数据集关系构建知识图谱，WarpGate [7] 采用基于嵌入的连接发现，LakeBench [4] 则对可连接表发现进行基准测试。

用于数据管理的大语言模型。 大语言模型正在改变数据管理 [9]: Magneto [31] 结合小语言模型与大语言模型进行模式匹配（类似于我们的混合方法），Table-GPT [22] 针对表格任务进行微调，但大语言模型在企业模式上表现下降 [18], ，这促使我们采用基于统计信息的设计。Kossmann 等人 [20] 探索了工作负载驱动的依赖发现，这与我们的查询反馈机制相关。

知识图谱构建。 KGFabric [32] 管理万亿关系规模的知识图谱。我们先前的工作介绍了 Tursio 的搜索平台 [14]；本文聚焦于 Tursio 中的连接推断组件。

3 系统概述

Tursio 结合统计信息与大语言模型，为大型上下文图推断连接，即利用数据中的事实性并运用推理，就像人类使用大语言模型时所做的那样。后者可避免明显错误，并使决策更加可靠。统计信息与大语言模型的这一组合，使所得连接推断既可扩展又准确。在本节的其余部分，我们将概述我们的方法。

图 2 展示了 Tursio 中可扩展连接推断的概览。首先，我们结合轻量级统计、基于启发式的剪枝与候选键评分，并在可用时从信息模式中提取约束，来推断主键。随后，我们为可能作为外键候选的列抽取样本，并识别可能的包含依赖。我们进一步使用大语言模型裁判优化这些依赖关系，最终生成有效的左连接树。用户仍可配置主键或外键，甚至定义复合键。他们也可以拒绝引擎检测到的任何连接，并用更适合其业务用例的连接予以覆盖。此外，Tursio 利用查询历史，基于现有工作负载优化连接推断，并随时间不断演进。最后，如果底层数据库引擎支持信息模式约束，Tursio 会利用这些约束来启动连接推断。

下面，我们将更详细地介绍上述各个组件。

4 推断主键

现代企业拥有大型数据库和数据仓库，允许用户在数据定义语言中指定约束，但出于性能和规模方面的原因，可能不会强制执行这些约束。因此，随着时间推移，实际情况可能会逐渐偏离已定义的约束。或者，如果键约束无论如何都不会被强制执行，用户可能不会定义这些约束。这使得以多种方式推断主键变得具有挑战性。首先，表中可能存在重复值或缺失值，使得主键变得模糊且难以识别；也就是说，仅检查某列的值是否具有 DISTINCT 性，并不足以识别主键。即使所有值都不重复，该列也可能不是主键，例如姓名、电话号码或社会安全号码，它们很可能各不相同，但并不是 key。最后，情况更糟的是，在大型数据库上运行 DISTINCT 或 COUNT(DISTINCT) 极其缓慢，并且当列数较多时难以扩展，甚至会影响系统性能。

我们的方法是计算全局计数（分别针对表和列），并将其与不同值计数进行比较。对于小型表（例如，行数少于 1M），不同值计数可以是精确值；对于大型表，则可以是近似值。具体而言，我们检查不同值计数是否至少达到全局计数和同一表中所有列的最大不同值计数的 $x\%$。后一个条件确保我们只考虑不同值最多的列。另请注意，近似不同值计数可能大于计数，即可能偏高或偏低，因此我们需要进行归一化。形式化地说，

$$\begin{aligned}\text{key_candidates} = \{c \in \text{columns} :\ & c.count > 0,\\ & c.distinct >= x * c.count,\\ & c.distinct <= (2-x) * c.count,\\ & c.distinct >= x * max_distinct,\\ & c.distinct <= (2-x) * max_distinct\}\end{aligned} \tag{1}$$

通过实验，我们发现将 x 设为 0.95（或 95%）效果最佳。确定键候选后，我们会为每个候选项评分。键列通常与表名关系密切，例如，包含 *order_key* 列的 *Order* 表。因此，我们按如下方式计算列名 *cn* 与表名 *tn* 之间的距离：

$$d(tn, cn) = \begin{cases} 1 & \text{if } cn \subseteq tn \\ seq_match(tn, cn)/len(tn) & \text{otherwise} \end{cases} \tag{2}$$

这里，seq_match(tn,cn) 计算表名与列名之间最长序列匹配的长度；我们用表名长度对该值进行归一化。如果列名包含在表名中，则将距离设为最大值 1。然后，我们按如下方式计算每个键候选的总体评分：

$$\begin{aligned}\text{key_score}(c) = {} & w_1 * d(tn, cn)\\ & + w_2 * c.distinct/c.count\\ & + w_3 * is_named_id(cn)\\ & + w_4 * is_named_key(cn)\end{aligned} \tag{3}$$

前三个组成部分的权重相同，即 $w_1 = w_2 = w_3 = 1$，$w_4 = 0.5$，最后一个组成部分的权重为其一半，即 ，但之后可以根据评估进行调整。最后两个组成部分用于检查列名是否以常见后缀结尾，例如 *id*、*key* 或 *nr*。最后，我们根据评分从候选项中选出一个或多个主键。如果存在明显的胜出者，即排名第一的候选项评分显著高于其他候选项，我们就将其选为主键。否则，我们保留所有主键候选项，并继续进入下一阶段。这样可以确保我们在检测包含依赖时考虑所有相似的键候选项。

5 推断包含依赖

经过规范化的数据库模式之间存在依赖关系。例如，订单 表可能有一个外键列客- 户_编号 ，它引用客户 表中的主键列编号 。此类依赖关系称为包含依赖 [3]，表示一项约束：某个列或列集合中的值必须是另一列或列集合中值的子集。识别不同列之间的此类包含依赖（IND），对于构建连接良好的上下文图至关重要。然而，与主键类似，IND 可能存在模糊性，难以识别，例如外键列中存在缺失值或错误值。此外，检查所有表对和列对之间的包含依赖是一项开销巨大的操作，在大型数据库模式下无法扩展。更糟的是，即使外键列中的值全部出现在主键列中，这种包含依赖在语义上仍可能无效，例如，两列的值域可能相同但语义不同，比如两个不同表中的自增序列号。

我们推断包含依赖的方法与推断主键的方法类似：首先利用轻量级统计信息和启发式方法缩小搜索空间，然后使用大语言模型裁定候选项。具体而言，我们首先从所有表中的非主键列抽取样本。随后，对于每一对表，我们考虑所有数据类型匹配的主键列-外键列组合。对于每个组合，我们检查外键列中的所有抽样值是否都存在于主键列中。如果是，则将该组合视为包含依赖候选。最后，我们根据各种启发式方法为每个候选项评分，并为每对表选出最佳候选项。下面将更详细地介绍每个步骤。

鉴于企业数据通常杂乱且不完整，我们首先清理从外键列抽取的样本值，去除空值、空字符串和明显的离群值，例如无符号列中的负值。然后，我们仅保留四分位距范围内的值，以筛选出典型值。对于字符串值，我们仅考虑长度处于四分位距范围内的值。

从所有非主键列中抽取 clean 样本后，我们会为数据类型匹配且值集合有效的所有主键列—外键列对计算包含依赖分数，方法与 [29] 类似。具体而言，我们将列 A 包含于列 B 的评分定义如下：

$$\begin{aligned}\text{ind_score}(A, B) = {} & w_1 * min(A.distinct/B.distinct, 1.0)\\ & + w_2 * 1/dep_count(A)\\ & + w_3 * ref_count(B)/max_ref_count\\ & + w_4 * seq_match(A, B)/max(len(A), len(B))\\ & + w_5 * is_named_id(A)\end{aligned} \tag{4}$$

其中，第一项检查外键列 A 中不同值的数量与主键列 B 中不同值的数量之比，并将其上限设为 1.0。第二项为参与较少包含依赖的外键列赋予更高评分，这表明它们更有可能是有效外键。第三项为被多个外键列引用的主键列赋予更高评分，并根据数据库模式中的最大引用数量进行归一化。第四项计算外键列名与主键列名之间的最长序列匹配，并根据两个列名中的较大长度进行归一化。最后一项检查外键列名是否以常见后缀结尾，例如 *id*、*key* 或 *nr*。所有项的权重相同，但之后可根据评估结果进行调整。由于使用的是样本且数据存在模糊性，我们会为每对表生成多个包含依赖候选项。不过，我们仍会将其限制在最低阈值以上，以避免假阳性。

在现实世界的企业数据中，表和列名会隐含地体现业务语义。例如，名为 Patient 的表表明这是医疗机构的数据库。如果还存在一个名为 Hospital 的表，就能进一步确认这一点。现在

如果同一数据库模式中还包含一个名为 Admissions 的表，那么 Patient 表与 Admissions 表之间极有可能存在外键关系。同样，如果 Admissions 表中有一个名为 patient_id 的列，那么它极有可能是引用 Patient 表主键的外键。

我们利用上述直觉，使用大语言模型对包含依赖候选项进行剪枝。我们根据表名、列名和样本值，使用大语言模型对这些候选项进行判定。这一步使我们能够利用大语言模型的语义推理能力，找出最佳的包含依赖；例如，即使样本值匹配，其语义也可能不同。

总之，我们将基于统计的剪枝与基于大语言模型的判定相结合，从而能够扩展到包含许多表和列的大型数据库模式，同时确保包含依赖的质量较高且假阳性较少。不过，这种方法的局限在于，如果样本不够具有代表性，我们可能会遗漏一些有效的包含依赖，即产生假阴性。如有需要，我们允许用户手动定义这些依赖，以处理此类情况。

6 连接树生成

包含依赖表明数据的不同部分之间存在怎样的关联。然而，盲目地利用它们连接表，可能导致数据重复和结果错误。例如，假设我们有两个表 ORDERS 和 ORDER ITEMS。Orders表_有一个名为 order_id 的主键，而 ORDER_ITEMS 表有一个名为 order_id 的外键。ORDERS 表包含订单总价，ORDER_ITEMS 表包含商品数量和商品价格。如果连接这些表时不考虑包含依赖，结果中可能会出现重复的订单行。为避免这个问题，我们生成左连接树，以系统化的方式组合表。具体来说，我们首先找出外键数量最多的事实表，即主键被引用次数最多的最左表。然后，我们从该事实表中找出最佳可达的维度表。接着，我们对外键数量第二多的事实表重复这一过程，依此类推。在此过程中，我们先构建更集中的连接树，再纳入较小的连接树。我们的目标是尽可能生成有效连接树，以便之后用于查询处理。查询处理超出了本文的讨论范围，但下文将更详细地介绍连接树生成。

我们按如下方式确定可到达的最佳维度表。首先，我们枚举从给定事实表可到达的所有 维度表，并按照其最长路径长度降序排列。此排序可确保先选取最长路径，同时避免与连接树后续部分中的路径重复。然后，对于每个维度表，我们枚举所有可到达的路径（从事实表出发），并选取使组合包含依赖分数最大化的路径，该分数定义为 $\pi_i(ind_score_i)$。其直观含义是，我们希望最大化该路径确实由包含依赖构成的概率。为给定维度表选定最佳路径后，我们会丢弃其他路径中不在最佳路径上的节点。这样可以确保在为其他维度表选取路径时不会产生环。

算法1展示了生成连接路径的伪代码。我们首先识别所有最左侧表，并按其引用计数降序排序

算法1： 生成连接路径

```
数据： 外键 fk
结果： 输出结果 y
1  连接_路径 = {};
2  根 = leftMost(fk);
3  roots.sort(key=lambda r: refCount(r), reverse=True);
4  对于root ∈ roots 执行
5      dims = reachable(root);
6      维度.sort(key=lambda d: maxPathLen(d), reverse=True);
7      对于dim ∈ dims 执行
8          paths = allPaths(root, dim);
9          最佳_路径 = 无;
10         最佳_评分 = 0;
11         对于路径 ∈ 路径 执行
12             路径_评分 = 1;
13             对于跳数 ∈ 路径 执行
14                 路径_评分 *= 跳数.ind.评分;
15             end
16             如果路径_评分 > 最佳_评分 则
17                 最佳_评分 =路径_评分;
18                 最佳_路径 = 路径;
19             end
20         end
21         顶部_最佳_路径 = 拓扑_排序(最佳_路径);
22         连接_路径.add(顶部_最佳_路径);
23         丢弃其他跳数(paths, 最佳_路径);
24     end
25 结束
26 返回连接_路径;
```

（第2–3行）。然后，对于每个最左侧表，我们枚举所有可达的维度表，并按最长路径长度降序排序（第4–6行）。对于每个维度表，我们枚举所有可达路径，并选取包含依赖分数总和最高的路径（第8–20行）。最后，我们对最佳路径进行拓扑排序，并将其添加到连接路径中（第21–22行）。我们还会丢弃其他路径中不属于最佳路径的跳转，以避免出现环（第23行）。算法最终返回所有连接路径（第25行）。

总而言之，我们使用包含依赖以系统化的方式生成左连接树。这确保我们在连接表时避免数据重复和错误结果。生成的连接树可供后续查询处理使用。

7 查询历史反馈

Tursio 默认根据数据推断连接。虽然这有助于识别显而易见的关系，但我们仍可能遗漏人们在工作负载中实际使用的关系。为弥补这一点，我们利用查询历史作为来自现有工作负载（例如仪表板、报告或专家查询）的反馈循环。我们的想法是，利用查询历史中记录的隐式人工反馈来调整连接推断。例如，仅根据数据推断连接时，可能会漏掉非主键列上的额外约束，例如日期列上的约束。

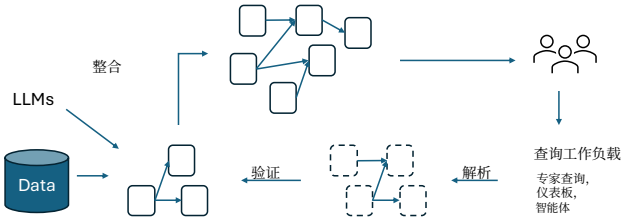


图3：Tursio 中的查询历史反馈循环。

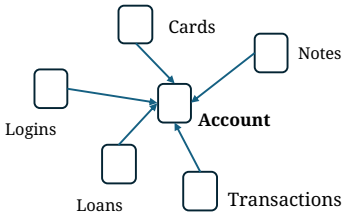


图 4：黑洞模式示例。

查询历史还提供了一种随时间演进推断连接的方式。例如，随着工作负载或语义发生变化，不同的连接路径可能会更受青睐。

图3展示了 Tursio 中基于查询历史的反馈循环。首先，我们收集同一数据库上所有工作负载的查询历史记录（步骤1）。随后，我们解析这些查询，以识别所使用的连接路径（步骤2）。接下来，我们结合大语言模型和在数据库上执行查询来验证这些连接路径（步骤3）。最后，我们将已验证的连接路径与现有的推断结果整合，以进一步完善它们（步骤4）。下面，我们将更详细地介绍这些步骤。

解析。 鉴于 Tursio 与平台无关，即支持多种数据库后端，我们使用 ANTLR 构建了一个通用查询解析器。由于 SQL语法可能因特定供应商扩展而变得非常复杂，我们只处理分析查询中常用的结构。我们还采用了一套较宽松的语法规则，以忽略与连接提取无关的复杂结构。具体而言，我们的语法规则不会解析数据定义语言语句、写操作、架构操作或用户管理操作。放宽语法规则使我们能够以较低延迟解析大量查询历史记录。

验证。 解析连接表达式后，我们会结合大语言模型和数据库执行来验证这些表达式，具体如下。请注意，Tursio 系统没有传统的绑定阶段。因此，如果列名未完全限定，我们会使用大语言模型来判断哪些表参与了连接条件。随后，我们在数据库上执行该连接条件，以检查它是否返回结果。如果连接返回结果，我们就将其视为有效连接表达式。

整合。 最后，我们将从查询历史中提取的连接表达式与通过统计信息推断出的表达式合并。在 Tursio 的默认版本中，对于一对表之间的连接，我们仅支持一个连接子句；它可以基于单个 key，也可以基于复合键。这一假设有助于简化系统后续的查询生成过程。如果一对表之间存在多个连接表达式，我们会将它们全部展示给用户，让用户选择最适合其业务用例的表达式。

8 处理细节

我们现在介绍 Tursio 如何处理其中的一些细微差别。

从信息架构进行引导。Tursio 会利用信息架构中的约束来启动连接推断。具体而言，我们从信息架构中提取主键和外键约束，并将其用作连接推断的初始候选项

用于连接推断。随后，我们使用前文所述基于统计信息的剪枝和基于大语言模型的判定方法对其进行细化。从数据库模式中提取连接关系，可以为大语言模型提供更好的候选项，从而更准确地识别包含依赖。这样一来，我们既能利用现有约束，又能确保连接推断的质量。

请注意，Postgres、MySQL、SQL Server 等传统数据库支持信息模式约束，并保证这些约束得到强制执行。然而，Snowflake、BigQuery、Redshift 等现代数据仓库也支持信息模式约束，但在数据摄取或查询处理期间不会强制执行这些约束。

支持复合键。Tursio 允许用户定义复合主键和外键。具体而言，用户可以指定多个列作为复合键的一部分。随后，Tursio 会将这些列的组合作为一个组合连接键。由于需要考虑的键组合数量众多，通过上述流程检测复合键仍是一个未解决的问题。Tursio 还会从数据库模式约束中提取复合键的参照约束，如上一节所述。

处理“黑洞”模式。一些数据库存在“黑洞”模式，其中一个维度表有来自多个事实表的多条入边。图 4 展示了一个此类黑洞模式示例：Account 表有来自 Transactions、Loans、Logins、Cards 和 Notes 表的入边。在这种情况下，如果查询从 Account 表开始，Tursio 将无法导航到其他任何表，因为该表没有出边。为处理此类情况，Tursio 会在连接推断期间识别黑洞模式，并从黑洞表向所有与其相连的事实表添加出边。这样，Tursio 就能从黑洞表导航到数据库模式中的其他表。

9 融会贯通

现在，我们来介绍 Tursio 中自动推断连接的产品体验。用户可以连接自己的数据库，并选择要导入 Tursio 的表。表导入后，Tursio 会使用本文所述的方法自动推断这些表之间的连接。随后，系统会在 Tursio 用户界面中将推断连接可视化，如图 5 所示。边最初以虚线显示，表示这些连接是推断得出的，尚未经过用户确认。用户可以点击每条边，查看推断连接的详细信息，包括主键列和外键列。随后，用户可以根据自己的领域知识和业务用例，选择确认或拒绝每条推断连接。

Tursio 会在后续的增量式 训练中记住此前已确认的连接，也就是用户确认过的那些表对；这些表对会

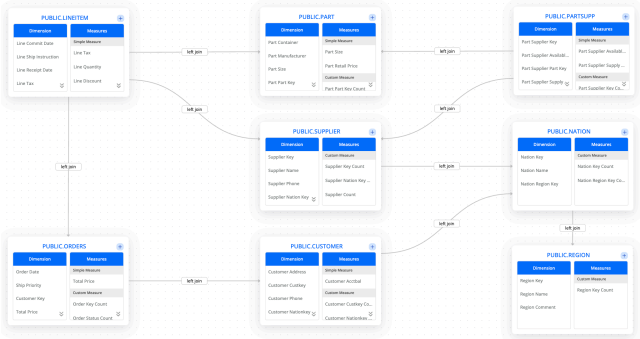


图5：Tursio 中 TPC-H模式的推断连接可视化示例。

表1：实验数据集及其统计信息。

数据集	# DB	# 表	# 维度	# 估计候选	# 主键	# 外键
TPC-H	1	8	61	214	8	7
TPC-DS	1	24	425	4,888	27	106
BIRD	11	75	806	29,822	93	TODO
生产环境	1	54	671	712	54	67

在后续的连接推断中排除在考虑范围之外。这样，Tursio 就能专注于其余表对，并随着收到更多用户反馈而逐步改进连接推断。不过，查询历史反馈仍可提供已推断表之间的其他连接（显示为表之间的多重边候选）。此外，如果需要，用户仍可运行完整训练，重新考虑所有表对。

在查询处理过程中，我们首先根据查询确定连接图中的起点，即理解用户意图。随后，我们根据推断连接扩展到连接图中所有相关节点。最后，我们使用大语言模型从语义上选出最佳路径，以生成最终的 SQL 查询。有关这方面的更多详情将留待未来工作。

10 评估

现在，我们评估所提方法的质量和效率，并对超参数进行消融研究。鉴于发现连接关系的关键信号是主键和包含依赖，我们报告这些子任务的准确率、精确率、召回率和 F1 分数，以及整体指标。我们还研究超参数对主键和包含依赖评分的影响，以及它们对外键关系检测的影响。

实验设置。 我们使用表1中总结的四个数据集开展实验，涵盖合成基准测试和真实生产工作负载。一个数据集可以包含多个数据库；我们分别评估其中的每个数据库，并报告该数据集的平均性能。

10.1 主键推断

现在，我们介绍在表1所列的四个数据集上对主键（PK）检测进行的评估。对于基准测试数据集（TPC-H、

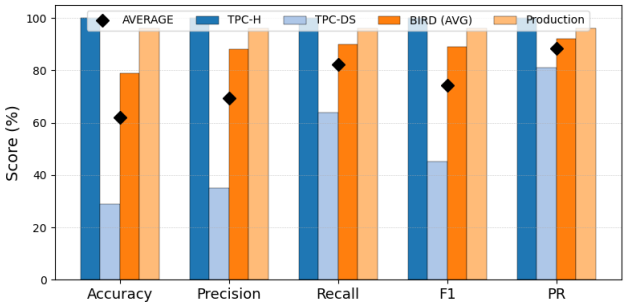


图 6：主键检测性能比较。

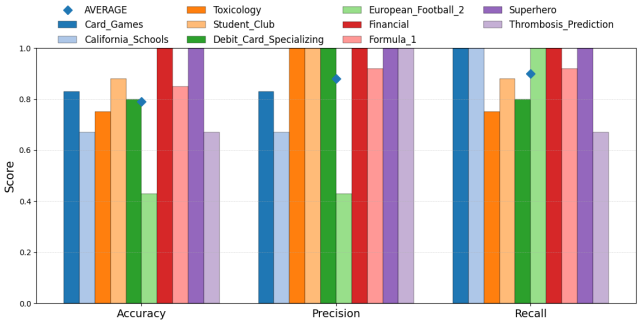


图 7：主键推断（BIRD）。

TPC-DS、BIRD），我们将检测出的主键与官方数据库模式元数据中定义的真实主键进行比较。对于生产数据集，我们请领域专家根据其对数据和模式设计的了解，手动验证检测出的主键是否正确。我们报告准确率、精确率、召回率和 F1 分数；如果检测出的主键与该表的真实主键完全一致，则认为预测正确。

结果： 图 6 总结了实验结果。在 TPC-H 中，所有主键均被正确识别，准确率、精确率和召回率均达到完美水平。对于 BIRD-Dev 数据集，在 11 个 BIRD-Dev 数据集上，我们的方法在主键检测方面的平均准确率为 79%，精确率为 88%，召回率为 90%。图 7 进一步展示了这 11 个 BIRD-Dev 数据集各自的结果。大多数数据集的评分都达到完美或接近完美；少数漏检出现在键命名不常见的模式中，或涉及复合主键，而我们当前的单列检测尚未覆盖这些情况。

对于 TPC-DS，图 6 显示，主键检测的平均准确率为 29%、精确率为 35%、召回率为 64%。在这种情况下，假阴性数量较多，主要是因为命名约定不标准，以及存在不符合常见模式的代理键。此外，该数据集包含当前方法无法检测的复合主键（用户仍可手动配置）。假阳性主要是指并非主键、但具有相似命名约定或数据特征的 key，例如基数和唯一性较高的 key。例如，在 TPC-DS 数据集中，表 *Customer* 的 *C_CUSTOMER_SK* 是主键，而我们预测的是 *C_CUSTOMER_ID*，后者表现出

表2：主键检测的基线比较。

HoPF-召回率	Tursio-召回率	Tursio-完美召回率
88%	88%	92%

相似特征，因此产生了假阳性。在我们的生产数据集上，主键检测的平均准确率、精确率和召回率均达到 96%。

基线： 我们将方法与 HoPF、主键检测 [13] 进行了比较。这项工作与我们的方法最为接近，都是利用统计信息和启发式方法来检测主键。由于他们在不同的数据集上进行了评估，我们将其工作中报告的各数据集上的性能平均值作为评估的基线。对于主键检测，基线报告在五个数据集上的平均召回率为 88%。我们的方法在这些数据集上达到了 88% 的平均召回率，与基线的 88% 召回率持平（表2）。

此外，请注意，在 Tursio 预测的主键与真实值不完全匹配的情况下，真实主键始终包含在排名靠前的主键候选项中，且其评分与预测主键的评分相近。我们通过检查真实值主键是否包含在排名前三的候选项中来评估主键候选项的召回率；这些候选项的置信度分数与排名第一的预测结果相差不超过 0.9 的阈值。我们将此指标记为 PR（完美召回率）。表2显示，Tursio 在所有数据集上的平均完美召回率为 92%，优于基线召回率。

完美召回率对 Tursio 来说是一个更相关的指标，因为只要真实值主键包含在 PK 键候选项中，且其置信度分数与排名第一的预测结果相差不超过 0.9 的阈值，下游连接依赖推断就不会受到影响。

这些结果表明，我们的方法实现了稳健的主键检测性能。未来的改进可以在适用时加入复合主键检测，以进一步减少边缘情况下对人工修正的需求。

10.2 包含依赖检测

现在，我们讨论表1所列数据集上的包含依赖（IND）检测评估。Tursio 会识别与已识别主键候选项相关联的初始包含依赖集合，并利用非主键列的数据样本对其进行细化。我们根据公式4对这些包含依赖进行评分，评估其构成有效外键关系的可能性，并依据阈值进行剪枝，以保留高质量候选项供后续连接推断使用。在当前设置中，我们使用的阈值为 0.4。我们还采用基于大语言模型的推理步骤，进一步验证剩余候选项，并生成一组最终的可能外键候选项。表3汇总了发现的初始包含依赖集合、经阈值筛选后保留的候选项数量，以及经过基于大语言模型的推理后得到的最终候选项集合。我们还评估了 BEAVER-DW 数据集上的包含依赖检测，以测试其在大型基准数据集上的表现。

即使初始包含依赖集合很大，阈值筛选和基于大语言模型的推理步骤也能有效地将候选项剪枝至便于人工审查的规模，同时保留可能对应于有效

表3：不同数据集中的包含依赖

数据集	初始	阈值	LLM
TPCH	33	18	9
TPCDS	327	97	19
生产环境	6185	2291	21
BIRD-借记卡_卡_专门化	8	8	3
BIRD-欧洲_足球_2	74	58	7
BIRD-公式 1_	220	127	19
BIRD-学生俱乐部_	8	6	6
BIRD-加利福尼亚学校_	1	1	1
BIRD-毒理学	5	5	5
BIRD-纸牌_游戏	29	12	1
Bird代码库_社区	112	55	7
Bird-超级英雄	30	27	9
bird_financial	45	32	8
bird_血栓_预测	1	1	0
Beaver-DW	1002	297	56

外键关系的高质量候选项。由于大多数数据集都没有可用的包含依赖真实值，我们无法直接评估包含依赖检测的精确率和召回率。不过，我们发现 TPC-H 中的包含依赖数量与文献 [30] 和 [13],中报告的数量相当，这有力地表明了我们进行包含依赖检测方法的质量。

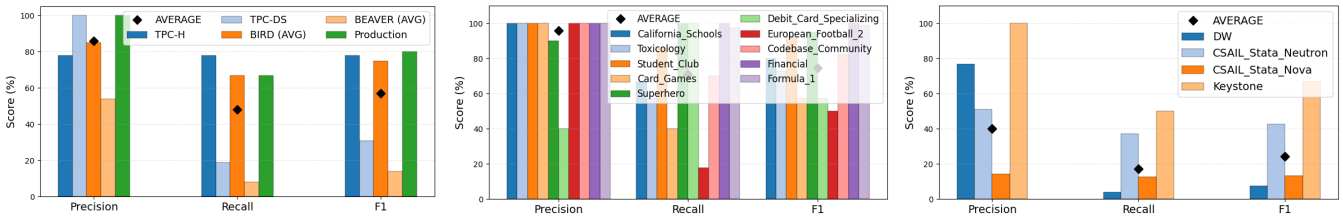
10.3 连接路径发现

现在，我们评估 Tursio 在构建上下文图时发现表之间关系的能力。我们报告精确率、召回率和*F1* 分数，并考察五类数据集：(1) BEAVER，(2) BIRD-Dev，(3) TPCDS，(4) TPC-H，以及 (5) 生产工作负载。对于 BIRD-Dev、TPCDS 和 TPC-H，真实值由官方模式元数据中定义为主外键约束的全部连接关系组成。对于 BEAVER 基准，我们使用两个数据仓库提供的所有已标注连接对。对于内部生产工作负载，连接标注由领域专家手动整理。

评估时，我们将上下文图推断出的连接关系集合与真实值中的连接对进行比较。如果预测的连接与真实值中的主外键关系完全匹配，则视为正确。对于错误预测，我们同时考虑假阳性（预测的连接并不存在于真实值中）和假阴性（真实值中的连接未被预测）。我们在图 8a 中报告所有数据集的总体结果，并分别在图 8b 和图 8c 中报告 BIRD 和 BEAVER 基准中每个数据集的结果。

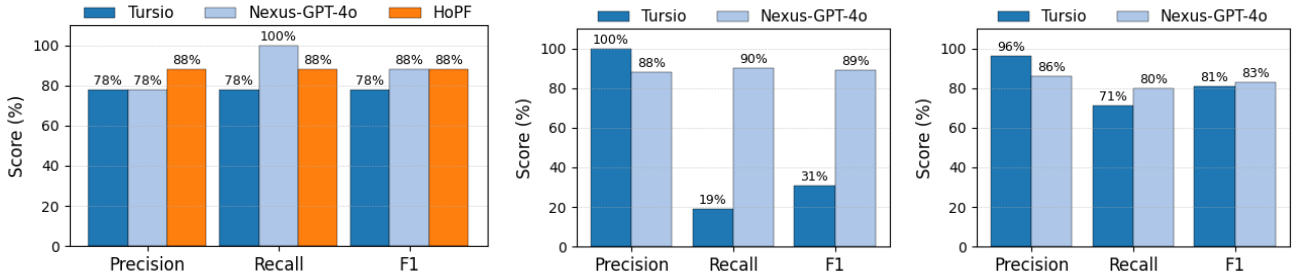
基线。 我们将方法与两个基线进行比较：HoPF（一种基于启发式的方法） [13],，以及 Nexus（将连接推断作为低秩矩阵补全问题来解决） [6]。HoPF 在五个数据集上的平均召回率为 91%，而 Nexus 在四个数据集上的平均召回率为 87。5%。此外，我们还在 TPCH、TPCDS 和 BIRD 数据集上将方法与 Nexus 进行比较。在 TPCH 上，Nexus 的 F1 分数、精确率和召回率分别为 88%、78% 和 100%；在 TPCDS 上分别为 89%、88% 和 90%；在 BIRD 数据集上则分别为 83%、86% 和 80%。

结果。我们在下文讨论每个数据集系列的结果。



(a) 摘要 **(b) BIRD** **(c) BEAVER**

图 8：各数据集上的连接推断性能。



(a) TPC-H 基线比较 **(b) TPC-DS 基线比较** **(c) BIRD 基线比较**

图 9：不同数据集上的基线比较。

TPC-H。 Tursio 的 F1 分数、精确率和召回率均为 78%。有趣的是，我们还发现了依赖关系 *lineitem.l_partkey → part.p_partkey* 以及*lineitem.l_suppkey → sup- plier.s_suppkey*，这些都是语义上有效的连接，但在 TPC-H 中并未声明为直接外键。TPC-H 定义明确的数据库模式以及表之间清晰的关系，有助于提高准确率和召回率，并表明我们能够在结构良好的模式中识别有意义的语义连接路径，而不局限于严格声明的外键约束。不过，如图 9a 所示，与基线相比，我们的精确率和召回率较低，这主要是因为我们采用了较为保守的连接分数阈值和基于大语言模型的验证。

TPC-DS。 该基准测试包含 106 个外键关系，显著高于其他数据集。我们在 TPC-DS 上取得了 31% 的 F1 分数、100% 的精确率和 19% 的召回率。与基线相比，我们的精确率更高，但召回率更低，如图 9b 所示。我们倾向于以较低的召回率为代价保持高精确率，因为假阳性会导致错误的连接路径，并可能显著影响下游任务，例如预测要查询的数据模型。

BIRD. 该基准测试包含一组多样化的数据集，其数据库模式设计质量和连接复杂度各不相同，因此我们能够在不同的真实场景中评估方法的稳健性。由于 BIRD 基准数据集具有明确的关系结构，我们的算法在该数据集上的表现显著提升。在 BIRD-Dev 上，我们取得了 81% 的平均 F1 分数、96% 的精确率和 71% 的召回率。如图 9c 所示，与基线相比，我们的精确率高出 10 个百分点，但召回率较低。较高的精确率表明大多数推断连接都是正确的，而召回率主要受限于诸如 *European Football 2* 等数据集中不完善的归一化

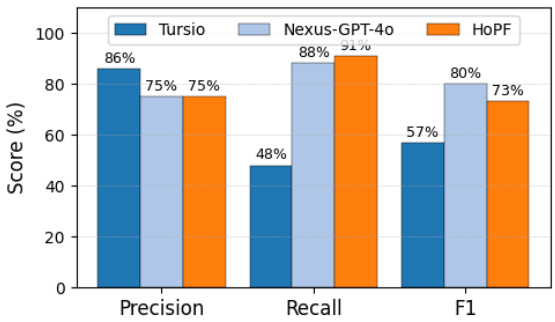


图 10：连接推断的平均性能。

以及诸如 *Card Games* 等数据集中的非主键-外键键约束。*debit_card_specialization* 数据集尽管具有清晰的模式语义，精确率仍然较低，因为部分推断连接（例如 *(transac- tions_1k.productid, products.productid), (transactions_1k.customerid, customers.customerid)*）在语义上有效，但未在数据库模式中声明为外键。

BEAVER。该数据集代表了最具挑战性的评估设置，主键-外键规范有限。在这种设置下，模型的平均表现较低，F1 分数为 14。4%，精确率为 54%，召回率为 8%。由于我们的连接路径发现算法旨在识别主键-外键关系，因此难以在 BEAVER 模式中可靠地识别给定的连接路径。

生产工作负载。 我们的生产数据集来自某企业客户的多表分析模式，包含约 54 张表和数百列，涉及房地产领域。该数据库模式没有定义明确的主键。这种情况反映了真实的企业部署环境，在这种环境中，

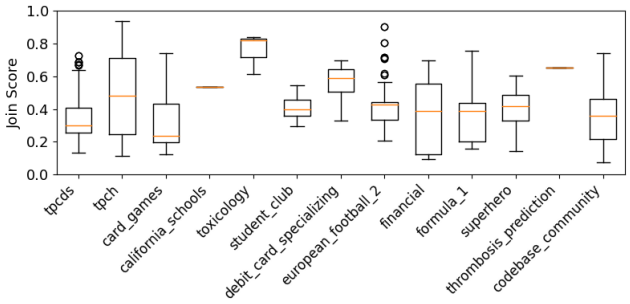


图 11：各数据集的连接得分分布。

出于性能或遗留系统方面的原因，通常会省略参照约束。我们的 F1 分数达到 80%，精确率达到 100%，召回率为 67%。完美的精确率表明，所有推断连接在语义上都有效，因此我们的方法在现实企业环境中具有很高的可靠性。然而，由于存在多列连接关系，召回率较低。目前的实现侧重于单列包含依赖，尚不能自动检测复合键连接。我们的方法能够避免错误的连接关系，并进一步扩展连接关系集合，使其符合实际业务用途和语义，因此适用于企业级复杂模式。

我们的方法比基线更为保守，优先考虑精确率而非召回率。这适用于现实场景，因为假阳性可能导致错误的连接路径，并对数据模型预测和 SQL 生成等任务产生负面影响。我们仅支持识别主键-外键关系，因此在包含非主键-外键连接或多属性连接的数据集中，召回率会受到限制。不过，较高的精确率确保了通过我们的算法推断出的连接更加可靠。事实上，如图 10 所示，我们的方法平均比基线的精确率高出 11 个百分点。假阴性仍可通过人工审查逐步找回。总体而言，Tursio 的连接推断性能因数据集类别而异，反映出我们的算法对模式复杂度较为敏感。结构良好的模式始终能得到高质量的推断，而不规则或约束较弱的模式仍然难以处理。

10.4 连接评分函数

为了更好地理解连接推断评分函数中各项信号的影响，我们针对 Tursio 引擎在剪枝前识别出的包含依赖（IND）集合进行了消融研究。在这项分析中，我们考察了所有已计算连接分数、但尚未经过剪枝阈值（默认值为 0。4）筛选的候选包含依赖。目前，每个包含依赖的连接分数是通过一组归一化特征分数的算术平均值计算得出的，每个特征分数都反映了外键合理性的不同方面。直接影响连接分数的特征详见第 5 节。

连接得分分布。我们以箱线图的形式分析了 TPC-DS、TPC-H 和 BIRD 数据集的连接分数分布（图 11）。每个箱体概括了某个数据集中连接候选项评分的中位数、四分位距（IQR）和整体离散程度。

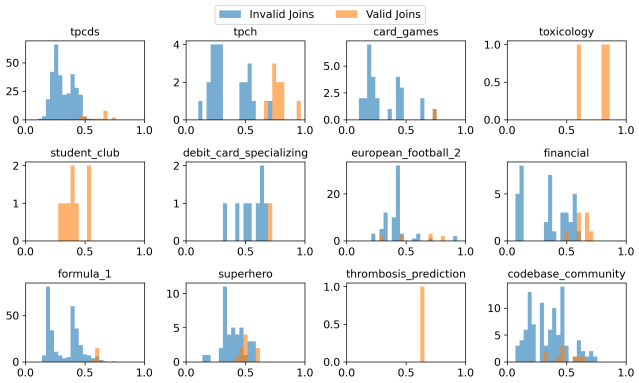


图 12：按有效性划分的连接得分分布。

显然，各数据集的评分存在显著差异。BIRD 数据集中的 california_schools、toxicology 和 thrombosis_prediction 等数据集候选项很少，因此分布范围有限。我们不会在后续研究中考虑这些数据集。BIRD 数据集 student_club、debit_card_specialization 和 superhero 的四分位距较窄，且没有离群值。其分布表明连接评分稳定，模式语义也更清晰。这也反映在连接推断的表现上：这些数据集具有较高的精确率和召回率。大多数 BIRD 数据集的中位评分集中在 0.35 至 0.45 之间。TPC-H 和 financial 数据集的四分位距相对较宽，表明连接分数的变化更大。TPC-H 的四分位距大约从 0.25 延伸至 0.70。这可能是因为其数据库模式更复杂，导致连接可能性的范围更广，推断也更具挑战性。TPC-DS 和 european_football 数据集存在若干离群值。这些离群值可能对应少数明显高于大多数候选项、极有可能成立的连接候选项。在 european_football 数据集中，这些离群值可能反映了模式设计中的不规则之处，导致某些候选项的连接分数异常偏高。因此，有必要深入分析连接分数及其与底层特征的关系，并考察它们与推断连接的有效性之间的相关性。

按有效性划分的连接得分分布。 我们研究了特征分数的分布，并比较了这些特征是否促成了各数据集中的有效连接关系。分布情况如图 12 所示。理想的连接得分应呈现两个完全不同的分布：无效连接集中在 0.0 附近，而有效连接集中在 1.0 附近，且两者完全不重叠。然而，在实际观察中，我们发现了更复杂的可分性模式。*TPCH* 的无效连接呈现明显的双峰分布，峰值位于 ~ 0.25 和 ~ 0.55 附近；而有效连接几乎全部位于 ~ 0.6 以上。对于 TPCH，将阈值设为 0.65 可获得近乎完美的精确率和召回率。*TPC-DS* 也表现出良好的判别能力，绝大多数无效连接低于 0.5，而有效连接集中在 0.5 – 0.7 之间。*formula_1* 表现出严重的类别不平衡（无效连接占比较高），但分离程度中等。无效连接的峰值较低（位于 0.1 和 0.4 附近），而有效连接则集中在较高值（0.5˜0.6）处。对于*formula_1*，约为 0.55 的阈值可以分离大部分类别。数据集*codebase_community*、*financial*、*superhero*、*debit_card_specializing* 以及*european_football_2* 中的有效连接和无效连接存在显著重叠。这表明

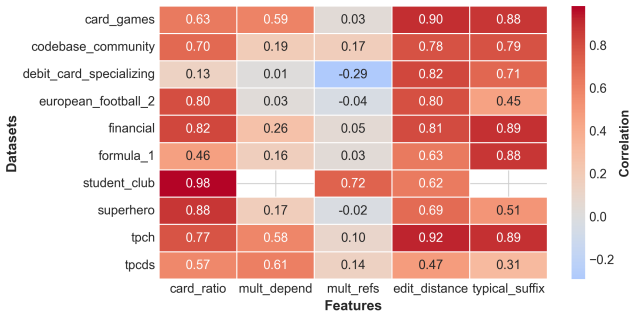


图13：连接得分特征的相关性。

当前的连接得分特征及其权重仍需进一步分析和调整，才能可靠地区分有效连接与无效连接。数据集*toxicology*和*thrombosis_prediction*仅识别出一个包含依赖，且该依赖对应的是有效连接。数据集*student_club*仅识别出 8 个包含依赖，且全部都是有效连接。另一方面，数据集*card_games*识别出 29 个包含依赖，但其中只有 1 个是有效连接。这些数据集中的严重类别不平衡限制了连接得分分布的可解释性。

我们观察到，连接分数对模式复杂度和领域高度敏感，因此进一步研究了单个特征对连接得分分布的影响，以及这些特征与推断连接的有效性之间的相关性。我们还采用监督学习方法对连接分数进行建模，以便更深入地了解这些特征与推断连接的有效性之间的关系。

特征对连接分数的影响：我们通过研究各项特征在基准数据集中的相关性，分析单个连接分数特征与整体连接分数之间的关系。直接影响连接分数的特征如下：

- *card*_比例表示外键候选项中的不同值数量与被引用的主键中的不同值数量之比。此特征施加了必要的基数约束，对值多样性超过主键的候选项进行惩罚。

- *mult*_依赖度通过计算一个外键候选项可包含于多少个主键的倒数，来衡量其排他性。该值越高，表示该外键在结构上越符合单个主键，这也是有效参照约束的一个重要特征。

- 多重_引用表示某个主键相对于数据集中被引用次数最多的主键，被引用的频率有多高。此特征体现了这样一种直觉：真正的主键通常会被同一数据库模式中的多个外键引用。

- 编辑_距离使用归一化编辑距离量化外键与主键属性名称之间的词汇相似度。此特征体现了命名一致性和模式级语义对齐。

- 典型_后缀是一个二元指标，用于表示是否外键候选项以常用的标识符后缀结尾，例如*id*、*key*或*nr*，从而体现了模式设计惯例。

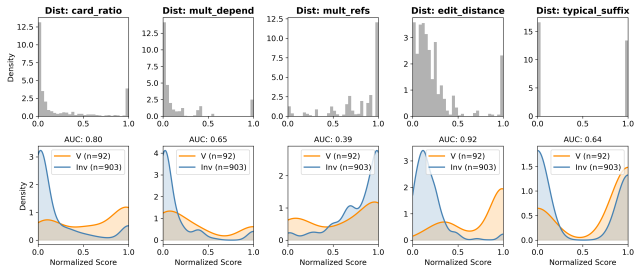


图14：连接得分特征的分布。

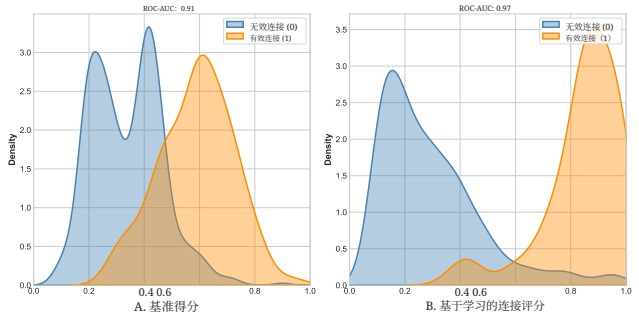


图15：基线与学习式评分函数之间连接分数的可分性。学习式评分有效地实现了更高的 ROC-AUC，并缩小了有效连接与无效连接之间的不确定区域。

图 13 展示了连接得分特征的热力图。我们发现，在大多数数据集中，编辑距离和典型后缀都呈现强正相关，这表明这些词汇相似度信号具有稳健性，且不受数据集影响。编辑距离呈现出最稳定的相关性，范围约为 0.62 至 0.92；在 TPCH、Card Games 和 Financial 中，相关性尤其强。同样，典型后缀在大多数数据集中也表现出较高的正相关性，多数情况下超过 0.70，表明它能有效捕捉模式层面的规律。

相比之下，card_ratio 总体呈现中等至强的正相关，但在不同数据集之间的差异较大。学生俱乐部和 Superhero 等数据集显示出较高的相关性，而其他数据集（例如借记卡专业化）的相关性则明显较弱。其余指标 mult_depend 和 mult_refs 的贡献不够稳定。它们的相关性通常较低，有时甚至为负（借记卡专业化和 European Football 2 尤为明显），这表明它们的效用有限或取决于具体情境。学生俱乐部数据集中的缺失值进一步表明，这些指标可能并不适用于所有情况。

连接评分特征的分布分析。我们进一步直观展示了不同特征在有效和无效包含依赖中的分布及类别条件密度。我们使用归一化特征分数，并绘制所有已识别包含依赖中特征值的直方图，以及依据真实值判定相应包含依赖有效或无效后得到的特征值核密度估计（KDE），从而深入了解整体分布。

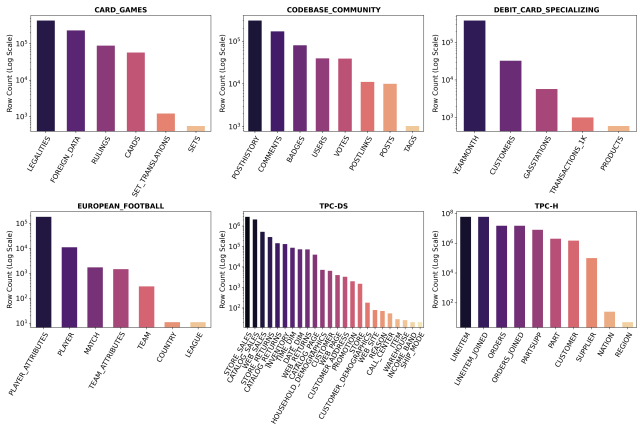


图16：包含元组数超过 100K 的表的数据集。

图14展示了分析结果。能够有效提升连接评分的特征呈现出明显分离的条件密度曲线：有效连接集中在特征空间的不同区域，与无效连接的重叠有限。尤其是词法相似度特征，例如 编辑_距离 和典型_后缀，表现出明显的区分度：有效包含依赖的评分始终较高，而无效候选项则集中在接近零的位置。相比之下，多重_依赖 和多重_引用等结构指标的有效与无效分布存在大量重叠，尽管其整体分布有时呈现偏斜。单独使用多重_引用 （AUC：0。39）时，表现甚至不如随机猜测。这表明这些特征单独不足以区分连接是否有效，但与更强的信号结合时仍可提供补充证据。基数_比率 则表现居中：虽然有效连接通常取值较高，但密度区域的重叠表明该特征容易受数据集特定属性影响，因此更适合作为辅助特征，而非主导特征。

尽管存在领域特定的噪声，全局AUC指标（大多为 > 0.60 ）仍证明，这些特征从根本上说是可靠的指标，能够在广泛的数据库类型中判断连接是否有效。这些模式表明，聚合异构信号是合理的：区分能力很强的词汇特征可由较弱但具有结构意义的指标加以强化，从而实现稳健的连接排序。

基于学习的连接评分。除了基于均值聚合特征分数外，我们还尝试采用监督学习方法对连接分数进行建模。具体而言，我们训练了一个逻辑回归模型，其中 IND 特征分数作为自变量，而目标变量表示某个 IND 是否根据真实模式对应于有效的外键关系。由此得到的模型会生成概率连接评分，以反映某个 IND 有效的可能性。图15展示了所有已识别 IND 的学习所得连接分数分布。对于所有已识别的 IND，我们根据其是否对应于数据库模式中提及的实际外键关系，将其标记为 *valid* 或 *invalid* 。

如图15所示，学习得到的权重显著提升了连接分数的判别能力。尽管有效和无效包含依赖之间仍存在一些重叠，但学习得到的连接

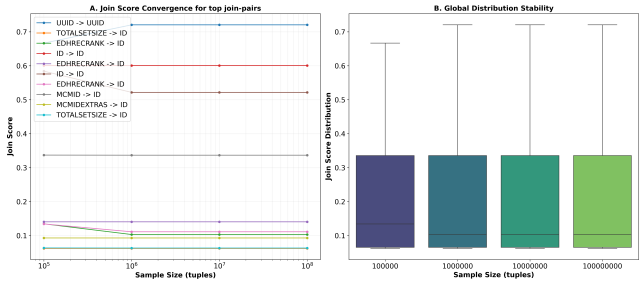


图17：样本量对连接分数的影响（Card Games）。

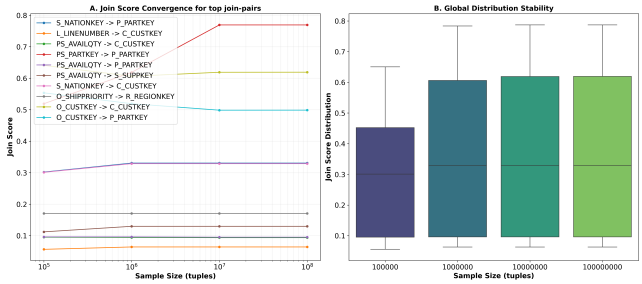


图18：样本量对连接分数的影响（TPC-H）。

分数呈现出更清晰的区分，尤其是为大量无效连接赋予较低分数，同时为大多数有效连接保留较高分数。在我们的场景中，这种行为是理想的，因为连接分数主要用于剪枝机制，而非作为最终决策规则。将学习得到的权重纳入生产系统将是未来

10.5 样本量的影响

我们首先考察了基准测试中各数据集的表规模分布。其中有六个数据集包含的表，其总行数超过 10^5 （图16）。对数刻度纵轴突显了高容量事务表与低容量维度表之间的巨大差异。鉴于这种结构差异，我们进一步分析了评分函数对数据采样的敏感性。

包含依赖的连接分数与候选主键和外键（FK）列中的不同值数量等统计信息密切相关。遗憾的是，在大型表上计算精确的不同值计数需要大量计算。因此，Tursio 利用基于采样子集计算的近似统计量。为了评估评分函数对这些近似值的敏感性，我们将样本量设为 S，覆盖 $\{1, 10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8\}$ 个元组。我们分析了 TPC-H 和 BIRD 基准测试中的包含依赖评分，并总结出三个关键观察结果。

1. 对检测到的包含依赖数量的影响。样本量较小时，系统会识别出更多包含依赖候选。这是意料之中的，因为样本有限往往会低估不同值的基数，从而导致包含依赖检测过于乐观。随着样本量增加，这些估计会变得更加准确，筛选也会更加严格，因此识别出的包含依赖更少。

2. 随着样本量增加，连接分数的稳定性。单个包含依赖的连接分数在不同采样方案下可能有很大差异。在大多数情况下，随着样本量增加，评分会下降

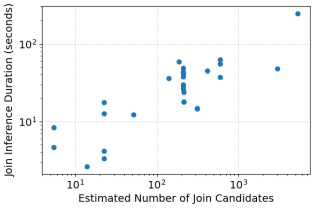


图19：连接推断时间与估计的连接候选项数量。

并最终在观察到足够的数据后趋于稳定。例如：图17和图18分别展示了卡牌_游戏数据集和*TPC-H*数据集中的这种行为，其中连接分数在收敛前单调下降。这是因为初始采样会高估外键在主键集合中的“覆盖率”。观察到足够的数据后，评分便会稳定下来，这表明样本已捕捉到该关系潜在的真实分布。

总体而言，这些结果表明，虽然较小的样本量会在连接评分中引入方差和乐观偏差，但一旦样本量达到约 10^6 个元组，连接分数便趋于稳定。超过这一点后，增加样本量对检测到的包含依赖数量及其相关连接分数的影响均可忽略不计。因此，我们选择 10^6 作为实用的默认值，以在计算效率与稳健、稳定的连接得分估计之间取得平衡。

10.6 可扩展性

最后，我们评估连接推断算法的可扩展性。我们将连接推断时长 衡量为针对给定数据集枚举并评估所有连接候选项所需的总墙钟时间。我们在图19中展示了连接推断时长（以秒为单位）与所探索的连接推断候选项数量（采用第1节所述方法，并同样报告于表1）的双对数散点图。我们展示了具有不同数量的表、列和连接关系的模式。从图19可以看出，连接推断时间大致随连接候选项数量呈线性增长。

总体而言，我们发现 Tursio 具有良好的可扩展性，基于大语言模型的剪枝能够有效缓解连接候选项的组合爆炸，即无需穷举便可高效筛除无效连接。通过降低对穷尽式统计信息的依赖，并利用基于大语言模型的语义推理，我们的解决方案在大型复杂数据库模式下仍能保持合理的执行时间，从而支持真实世界的部署场景。

11 经验总结与结论

在 Tursio 的多个生产环境数据库模式中构建并部署可扩展的连接推理，带来了若干实践经验，我们认为这些经验普遍适用于连接发现系统。

可扩展性需要分阶段剪枝。 随着表和维度列数量增加，连接推断的候选空间会迅速扩大，在真实世界的数据库模式中可达到数十万个候选项。在我们的方法中，轻量级统计筛选器会在进行任何大语言模型调用之前，将候选集缩小数个数量级，从而使延迟和成本都保持在可控范围内。

统计信息与大语言模型互为补充，而非彼此替代。 纯统计方法的召回率很高，但会产生许多假阳性，尤其是在中等连接分数区间（0.3–0.5），因为仅凭结构特征难以作出明确判断。相反，大语言模型具备语义理解能力，但在缺乏上下文且候选项过多时容易产生幻觉。这种双管齐下的设计——用统计信息进行剪枝，用大语言模型进行判定——充分发挥了二者互补的优势：统计信息将搜索空间缩小到大语言模型能够有效推理的范围，而大语言模型则解决统计信息无法消除的歧义。

词汇信号出奇地稳健。我们的消融研究发现，编辑距离相似度和命名约定指标（例如，_id、_key 后缀）是不同数据库模式中最可靠的有效连接预测指标。这些简单的词汇特征始终优于依赖项数量和引用数量等结构指标；后者因数据集不同而表现出较大的差异。这表明，尽管模式设计者在规范化方面的严谨程度各不相同，他们通常仍遵循较为一致的命名约定——连接发现系统应利用这一规律。

模式质量决定推断质量。我们的评估揭示了显著差异：结构良好的数据库模式（TPC-H、BIRD-Dev）精确率达到 82–97%，而规范化程度较低的模式（BEAVER）精确率则降至 40%。这并非任何特定算法的局限，而是一项根本性挑战：当数据库模式缺乏明确的主键-外键规范时，真实值本身就会变得模糊不清。实际上，这意味着连接推断系统应向用户呈现置信度，并支持增量式优化，而不是追求完全自主运行。

反馈循环对于生产环境部署至关重要。在实践中，静态的一次性推断并不足够。工作负载不断演变，数据库模式也会发生变化，而领域专家掌握着无法仅凭数据获取的知识。将查询历史作为隐式反馈、将用户确认证作为显式反馈纳入系统，可以使系统不断改进，而无需重新训练。在我们的部署中，查询历史反馈发现了一些连接路径，例如日期列上的非主键-外键连接，而数据驱动推断会遗漏这些路径。

采样能够以极小的精度损失实现可扩展性。 对大型表计算精确统计信息的代价高得难以承受。我们的实验表明，当样本量达到约 10^6 个元组时，连接分数趋于稳定；此后增加样本量几乎不会带来进一步提升。这一实用阈值使我们能够高效地接入大型表，即无需扫描整个数据集，同时保持可靠的连接得分估计。

总之，我们提出了一种可扩展连接推断系统，它将统计剪枝与基于大语言模型的判定相结合，从结构化数据库构建上下文图。我们的两阶段流水线推断主键和包含依赖，构建左连接树以避免数据重复，并通过查询历史反馈持续优化连接。在 TPC-DS、TPC-H、BIRD-Dev 和生产工作负载上的评估表明，该方法能够以高精确率扩展到大型模式。展望未来，我们看到了将该框架扩展到多列和跨数据库连接发现的机会。

参考文献

[1] Ziawasch Abedjan、Lukasz Golab 和 Felix Naumann。2015。关系数据剖析：一项综述。*VLDB Journal* 24, 4 (2015), 557–581。[2] Jana Bauckmann、Ulf Leser、Felix Naumann 和 Veronique Tietz。2007。高效检测包含依赖。见 *ICDE*。1448–1450。[3] Marco A. Casanova、Ronald Fagin 和 Christos H. Papadimitriou。1982。包含依赖及其与函数依赖的相互作用。见 *PODS*。171–176。[4] Yanping Chai、Nan Sun、Yuyu Wang、Xiaoyuan Du 和 Guoliang Li。2024。LakeBench：用于在数据湖中发现可连接表和可并集表的基准测试。*PVLDB* 17, 8 (2024), 1925–1938。[5] Zhimin Chen、Vivek Narasayya 和 Surajit Chaudhuri。2014。Microsoft SQL Server PowerPivot for Excel 中的快速外键检测。*PVLDB* 13 (2014), 1417–1428。[6] Tianji Cong、Yuanyuan Tian、Andreas Mueller、Rathijit Sen、Yeye He、Fotis Psallidas、Shaleen Deep 和 H. V. Jagadish。2026。Nexus：通过迭代低秩矩阵补全仅从元数据推断连接图。(2026)。arXiv: 2602 08186 [ ] . cs.DB Zihui [7] Cong 和 Raul Castro Fernandez。2023。WarpGate：面向云数据仓库的语义连接发现系统。见 *CIDR*。[8] Raul Castro Fernandez、Ziawasch Abedjan、Famien Koko、Gina Yuan、Samuel Madden 和 Michael Stonebraker。2018。Aurum：一种数据发现系统。见 *ICDE*。IEEE, 1001–1012。[9] Raul Castro Fernandez、Aaron J. Elmore、Michael J.Franklin、Sanjay Krishnan 和 Chenhao Tan。2023。大语言模型将如何颠覆数据管理。*PVLDB* 16, 11 (2023), 3302–3309。[10] Jaya Gupta 和 Ashu Garg。2025。上下文图：人工智能的万亿美元机遇。检索于2025年12月30日，来自 https://foundationcapital.com/context-graphs- ais-trillion-dollar-opportunity/ [11] Alon Halevy、Anand Rajaraman 和 Joann Ordille。2006。数据集成：青春期。见 *PVLDB*。9–16。[12] Lan Jiang 和 Felix Naumann。2020。整体主键与外键检测。*Journal of Intelligent Information Systems* 54, 3 (2020), 439–461。[13] Lan Jiang 和 Felix Naumann。2020。整体主键与外键检测。*Journal of Intelligent Information Systems* 54, 3 (2020), 439–461。[14] Alekh Jindal、Shi Qiao、Shivani Tripathi、Niloy Debnath、Kunal Singh、Pushpanjali Nema、Sharath Prakash、Aditya Halder、Ronith PR、Sadiq Mohammed、Abdul Hameed、Karan Hanswadkar、Ayush Kshitij、Sarthak Bhatt、Rony Chatterjee、Jyoti Pandey、Christina Pavlopoulou 和 Ravi Shetye。2026。借助深度上下文使数据库可搜索。(2026 2602 08320 [ ])。arXiv: . cs.DB [15] 2003 Theodore Johnson、Amit Marathe 和 Tamraparni Dasu。. 数据库探索与 Bellman。(2003), 34–39。[16] Eduardo H. M. Kaminsky、Eduardo H. M. Pena 和 Felix Naumann。2023。发现相似性包含依赖。见 *SIGMOD*。ACM, 1–26。[17] Jaewoo Kang 和 Jeffrey F. Naughton。2003。关于不透明列名与数据值的模式匹配。见 *SIGMOD*。205–216。

[18] Mohamed Kayali、Michael Cafarella、Peter Baile Chen、Çağatay Demiralp 和 Michael Stonebraker。2025。Mind the Data Gap：弥合大语言模型与企业数据集成之间的鸿沟。发表于 *CIDR*。[19] Henning Koehler 和 Sebastian Link。2025。以高精确率和召回率挖掘有意义的键和外键。*PVLDB* 18，12（2025）。[20] Jan Kossmann、Stefan Lindner、Felix Naumann 和 Thorsten Papenbrock。2022。工作负载驱动的惰性数据依赖关系发现，用于查询优化。发表于 *CIDR*。[21] Christos Koutras、George Siachamis、Andra Ionescu、Kyriakos Psarakis、Zihan Brons、Jerry 和"；Miao、Nikolaos Bastian、Ziawasch Abedjan、Maurice van Keulen 和 Vassilis Christophides。2021。Valentine：评估用于数据集发现的匹配技术。发表于 *ICDE*。IEEE，468–479。[22] Peng Li、Yeye He、Denny Yashar、Weiwei Cui、Song Ge、Haidong Zhang、Danielle Rifinski Fainman、Dongmei Zhang 和 Surajit Chaudhuri。2024。Table-GPT：面向多样化表格任务的表格调优 GPT。发表于 *ACL*。[23] Yiming Lin、Yeye He 和 Surajit Chaudhuri。2023。Auto-BI：利用局部连接预测和全局模式图自动构建商业智能模型。arXiv:2306.12515 [cs.DB] https://arxiv.org/abs/2306.12515[24] Thorsten Papenbrock、Tanja Bergmann、Moritz Finke、Jakob Zwiener 和 Felix Naumann。2015。使用 Metanome 进行数据剖析。*PVLDB* 8，12（2015），1860–1863。[25] Thorsten Papenbrock、J ohannes Ehrlich、Jannik Marten、Tommy Neuber、Jan-Peer Rudolph、Martin Schönberg、Jakob Zwiener 和 Felix Naumann。2015。函数依赖发现：对七种算法的实验评估。发表于 *PVLDB*，第 8 卷。1082–1093。[26] Thorsten Papenbrock、Sebastian Kruse、Jorge-Arnulfo Quiané-Ruiz 和 Felix Naumann。2015。基于分治的包含依赖发现。*PVLDB* 8，7（2015），774–785。[27] Thorsten Papenbrock 和 Felix Naumann。2016。一种函数依赖发现的混合方法。发表于 *SIGMOD*。ACM，821–833。[28] Erhard Rahm 和 Philip A. Bernstein。2001。自动模式匹配方法综述。*VLDB Journal* 10，4（2001），334–350。[29] Alexandra Rostin、Oliver Albrecht、Jana Bauckmann、Felix Naumann 和 Ulf Leser。2009。一种外键发现的机器学习方法。。发表于 *WebDB*。[30] Alexandra Rostin、Oliver Albrecht、Jana Bauckmann、Felix Naumann 和 Ulf Leser。2009。一种外键发现的机器学习方法。发表于 关于 ***Web*** 与数据库的国际研讨会。https://api.semanticscholar.org/ CorpusID:[31] 11659431 Aamod Santos、Juliana Freire、Aecio Bessa 和 Yurong Deng。2025。Magneto：结合小型和大型语言模型进行模式匹配。*PVLDB* 18, 8 2025( )。[32] Peng Yi 等。2024。KGFabric：面向企业数据互联的可扩展知识图谱仓库。*PVLDB* 17，12（2024），4033–4045。[33] Meihui Zhang、Marios Hadjieleftheriou、Beng Chin Ooi、Cecilia M. Procopiuc 和 Divesh Srivastava。2010 3，1 2 2010 805 814。关于多列外键发现。*PVLDB* –（ ），–。