Nexus：仅凭元数据，通过迭代低秩矩阵补全推断连接图

Tianji Cong*密歇根大学，美国密歇根州安娜堡 congtj@umich.edu

Yuanyuan Tian 微软 Gray Systems Lab 美国加利福尼亚州芒廷维尤 yuanyuantian@microsoft.com

Andreas Mueller 微软 Gray Systems Lab 美国加利福尼亚州芒廷维尤 Andreas.mueller.ml@gmail.com

Rathijit Sen 微软 Gray Systems Lab 美国华盛顿州雷德蒙德 rathijit.sen@microsoft.com

Yeye He 微软研究院 雷德蒙德，华盛顿州，美国 yeyehe@microsoft.com

Fotis Psallidas 微软 Gray Systems Lab 纽约，纽约州，美国 Fotis.Psallidas@microsoft.com

Shaleen Deep 微软 Gray Systems Lab 麦迪逊，威斯康星州，美国 shaleen.deep@microsoft.com

H. V. Jagadish 密歇根大学 安娜堡，密歇根州，美国 jag@umich.edu

arXiv:2602.08186v1 [cs.DB] 9 Feb 2026

摘要

自动推断连接关系是实现有效数据发现、集成、查询与复用的关键任务。然而，在庞大而复杂的数据库模式中准确且高效地识别这些关系颇具挑战，尤其是在数据值访问受限的企业环境中。本文提出仅在元数据可用时的连接图推断问题。我们对大量真实世界数据库模式开展实证研究，观察到当连接图表示为邻接矩阵时，呈现出两个关键性质：高度稀疏性与低秩结构。基于这些新观察，我们将连接图推断形式化为低秩矩阵补全问题，并提出 Nexus——一种仅使用元数据的端到端解决方案。为进一步提升准确性，我们提出一种新颖的期望最大化算法，在低秩矩阵补全与借助大语言模型优化连接候选概率之间交替进行。大量实验表明，Nexus 在包括一个真实世界生产数据集在内的四个数据集上显著优于现有方法。此外，Nexus 可在快速模式下运行，在获得可比结果的同时实现最高 6 倍加速，为真实世界部署提供实用且高效的解决方案。

1 引言

检测一组表之间的连接关系，对于数据管理任务至关重要，例如在关系数据库中识别主键和外键（PK/FK） [6, 18, 29, 31], 为分析型工作负载构建商业智能（BI）模型 [1, 23],以及在 Web 语料库 [3, 4,28, 30] 和数据湖 [2, 8, 11, 13–15, 32–34]等多种来源中查找相关数据。

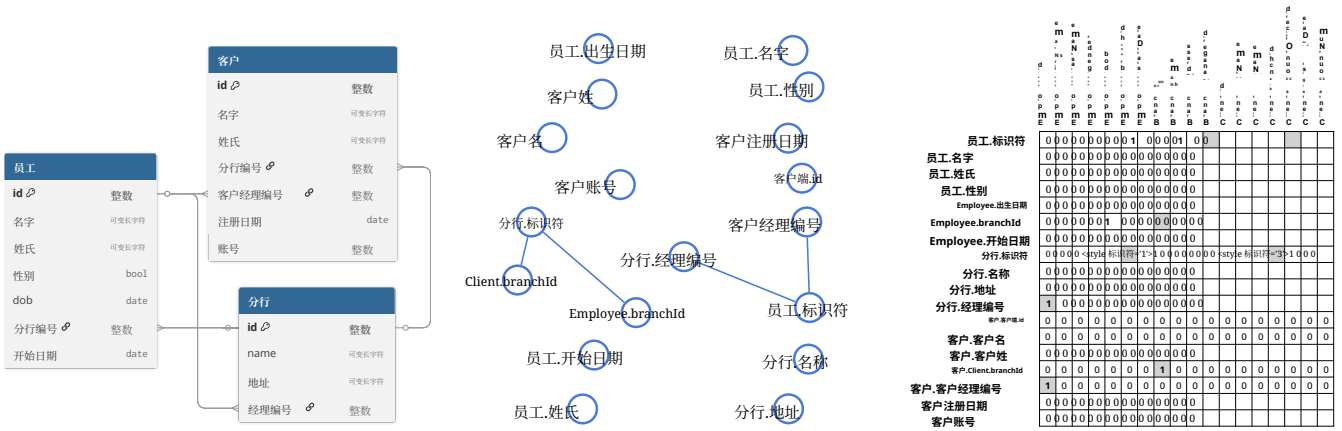
这项能力对于现代湖仓平台尤为重要。湖仓架构之所以具有吸引力，正是因为其设计原则（灵活性、可扩展性以及对多种数据的统一管理）；然而，这些原则也为连接检测带来了挑战。湖仓采用“宽松模式”，通常不支持或未声明 PK/FK 等引用约束。此外，将不同的数据集合并到单个存储库中，也模糊了数据库或数据库模式之间的正式边界。因此，推断完整的连接关系集合——我们称之为连接图——是湖仓的一项基础能力，可用于实现自动化数据编目、增强数据探索、机器学习特征工程、知识图谱构建和自然语言查询。图 1(b) 展示了一个简单数据库模式的连接图，其中每个节点代表一个表列，每条边表示两列之间的连接关系。

连接检测并非新问题，已有多种方法对此进行了研究，从传统启发式方法 [6, 18, 26],、基于嵌入的技术 [8, 14, 15],和机器学习模型 [1, 25, 29]，到近期的大语言模型（LLM） [12]。然而，现有解决方案主要依赖于访问数据值。无论是直接计算值重叠、生成嵌入，还是进行机器学习特征计算，利用数据值的计算开销都很大，尤其是对于包含大量大型表的数据集而言。更重要的是，假定连接检测算法能够自由访问数据，往往并不现实。对于在严格的隐私和安全约束下运营的数据服务提供商而言，访问客户数据往往不可行。除依赖数据外，现有解决方案通常只进行“局部”的成对检测，而忽略了全局连接图结构。尽管 Auto-BI [23] 通过全局图优化解决了这一局限，但它专为商业智能环境设计，适用于数据库模式遵循严格树状结构（例如雪花模式）的场景。这一设计假设在湖仓环境中并不成立，因为其中的数据库模式结构松散得多，约束也更少。

不同于可连接表搜索 [2, 8, 11, 13–16, 25, 33] （它旨在查找与给定输入表/列可连接的表），本文探讨的是一个更具挑战性的问题：连接图推断。给定一个

*在微软实习期间完成的工作。

Conference'17，美国华盛顿特区2026. ACM
ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(a) 实体关系图 (b) 连接图 (c) 连接图矩阵（邻接矩阵）

图 1：银行数据库模式示例及其对应的连接图和连接图矩阵。

表集合（具体而言，是其数据库模式和元数据），我们的目标是通过发现表之间的等值连接，尤其是主键/外键关系，推断相应连接图的邻接矩阵。尽管这两个问题密切相关，连接图推断从根本上更难：它需要发现整个表集合中所有成对的连接关系，因此相对于列数 n，最坏情况下的计算复杂度为二次方 $O(n^2)$；而可连接表搜索只需检查与指定输入表/列的可连接性，最坏情况下的时间复杂度为线性 $O(n)$。为此，我们开发了 Nexus，这是一种面向企业场景的精准且高效的端到端连接图推断方案。Nexus 专为仅使用元数据（详见第 2.1 节）而设计。不过，Nexus 也足够灵活，在查询日志和数据值可用时能够将其纳入，以进一步提高准确性（第 4.6 和 4.7 节）。

Nexus 的设计基于一个关键洞见：现实世界中的连接图具有高度结构化的特征。数据库模式通常围绕少数几个连接键构建，而大多数列从不参与连接。这种结构具有两个关键属性。首先，连接图的邻接矩阵（简称连接图矩阵）高度稀疏，如图 1(c) 所示。其次，除了图 1(c) 中显示的零行外，矩阵还存在显著冗余；例如，Employee.branchId 和 Client.branchId 列具有相同的可连接列集合，因此它们在矩阵中对应的行完全相同。这种冗余意味着，连接图结构可以由少数潜在因子来表示。从数学上看，这表明连接图矩阵具有低秩结构，可以用低维表示近似。

我们对 SchemaPile-Perm 数据集中 5,957 个非平凡的真实世界数据库模式进行了广泛分析 [12]，从实证角度证实了这一洞见：这些模式的连接图始终具有稀疏、低秩的特征。约 98% 的数据库密度低于 0.02，超过 95% 的数据库归一化排名低于 0.3。据我们所知，本研究详见第 2.2 节，是首个此类研究，也是规模最大的此类研究，因此本身就是一项重要贡献。证实连接图具有高度稀疏性和低秩特征，使其适用范围比那些局限于商业智能设置中常见的严格树形结构的方法更广 [23]。此外，我们的分析也为优先检测单列连接提供了依据。在 SchemaPile-Perm 中所有包含连接信息的数据库里，只有 7.5% 包含多列连接，这类连接仅占所有连接案例的 3.5%。因此，虽然当前版本的 Nexus 针对这一常见场景进行了优化，但其设计仍可扩展至多列连接，详见第 3.7 节。

基于这一关键洞见，我们将连接图推断表述为低秩矩阵补全 问题，目标是在低秩约束下，根据一组已知条目恢复连接图矩阵中的未知条目。我们通过将剪枝后的列对赋值为 0，并在有可用数据时将从查询日志中得出的已确认连接赋值为 1，来初始化矩阵；尽管我们的方法不依赖查询日志，但在有查询日志时，可以自然地将其纳入其中。对于剪枝后保留下来的其他候选条目，我们利用预训练机器学习模型或大语言模型为其是真实连接的可能性打分。随后，我们提出一种新颖的目标函数，在标准的低秩补全目标中加入两个额外项：(1) 一个损失项，鼓励恢复出的条目与机器学习模型/大语言模型给出的概率分数一致；(2) 一个正则化损失项，促进现实世界连接图中观察到的稀疏性。该表述使我们能够有效利用预训练模型的预测，同时遵循连接图固有的结构属性。

在实践中，预训练模型往往难以泛化到未见过的数据集，而强迫潜在条目遵循这些模型的不准确预测，可能会显著降低推断所得连接图的质量。为应对这一挑战，我们提出一种期望最大化（EM）算法，通过融入大语言模型提供的语义指导，迭代优化低秩矩阵补全。EM算法在两个步骤之间交替进行：（1）执行低秩矩阵补全；（2）使用大语言模型更新连接候选项的先验概率估计。该方法的一个关键

组成部分是利用大语言模型评估列实体类型的语义兼容性，这是实现有效连接的前提。我们将列实体类型定义为一种自由文本注释，用于概括某一列的值在其所属表上下文中的语义含义（例如，将 Employee 表中的 ID 列注释为"员工ID"）。与依赖固定实体类型集合或分类体系的现有注释方法不同，Nexus 利用大语言模型处理多样的列语义。我们利用大语言模型生成动态实体注释并确定其语义兼容性。根据评估结果，我们调整连接概率分数——提高语义匹配度高的候选项的权重，并对不匹配的候选项进行惩罚。这种迭代优化使 Nexus 能够纠正初始预测中的错误，并适应未见过数据集的独特语义。

总而言之，本文的贡献如下：

- 我们提出了 Nexus，这是一种准确、高效的端到端连接图推断解决方案，专为注重隐私的企业环境量身打造。Nexus 仅依赖元数据，并可在查询日志可用时选择性地利用它们，因此无需直接访问数据值也能有效运行。
- 我们对近 6,000 个真实数据库模式进行了广泛的实证分析，证实了一个关键且新颖的洞见：连接图既具有稀疏性，又始终呈现低秩特征。这一新发现构成了我们新颖的连接图推断算法的基础（第 2 节）。
- 我们将连接图推断形式化为一个低秩矩阵补全问题，其优化目标明确利用观测到的稀疏性和低秩结构，同时使用预训练模型提供初始连接似然度（第 3 节）。
- 为克服预训练模型先验预测较差的挑战，我们提出一种新颖的EM算法，在执行低秩矩阵补全与利用大语言模型迭代优化连接候选项的概率分数之间交替进行（第 3.5 节）。
- 我们在四个数据集（包括一个真实生产数据集）上展示了Nexus的有效性与效率。我们仅使用元数据的方法显著优于基线方法，包括那些使用数据值的方法（第 4 节）。

2 预备知识与观察

本节为我们的连接图推断方案奠定基础。我们将首先介绍关键概念并形式化定义该问题。随后，我们将展示对大量真实世界数据库模式进行实证研究的发现，这些发现直接推动了我们算法的设计。

2.1 问题陈述

我们首先给出连接图、连接图矩阵和连接图概率矩阵的定义。图1展示了一个简单示例数据库模式的连接图和连接图矩阵。

定义 1（连接图）。给定一组表 T，连接图是一个无向图 $G(T) = (V, E)$，其中：

- $V = \{c_1、 c_2、 \ldots\ldots c_n\}$ 表示 T 中所有表的列集合。

表 1：整理后的数据库模式的统计数据。

每个数据库的统计数据	Min	Max	Mean	中位数
表数	4	977	15	9
列数	17	4572	108	56
平均每个表的列数	4.04	37.46	6.73	6.00
密度	1.56×10^{-7}	0.032	0.005	0.003
归一化排名	5.58×10^{-4}	0.49	0.12	0.10

- E是连接关系集合。当且仅当两个列 c_i和 c_j之间存在连接关系时，它们之间才存在一条边（c_i、 c_j）。

当上下文明确时，我们将简称连接图 $G(T)$ 为 G。

定义 2（连接图矩阵）。设 $G = (V, E)$ 是表集合 T 的连接图。连接图矩阵 $A(T) \in \{0, 1\}^{n \times n}$；在不存在歧义时，也简单记作 A，它是连接图 G的邻接矩阵：

$$A_{i,j} = \begin{cases} 1 & \text{if } (c_i, c_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

由于我们考虑的是无向连接图，因此所得矩阵 A 是对称的（$A_{i,\ j} = A_{j,\ i}$），且对角线为零（$A_{i,\ i} = 0$）。

定义 3（连接图概率矩阵）。设 $G = (V, E)$ 为一组表 T的连接图。连接图概率矩阵$S(T) \in [0, 1]^{n \times n}$，在无歧义的情况下简称为 S，是一个 $n \times n$ 矩阵，其中每个元素 $S_{i,\ j}$表示列 c_i 与列 c_j 可连接的概率。

S 也是一个对称矩阵（$S_{i,\ j} = S_{j,\ i}$），并且由于连接图的无向性质，其对角线为零（$S_{i,\ i} = 0$）。

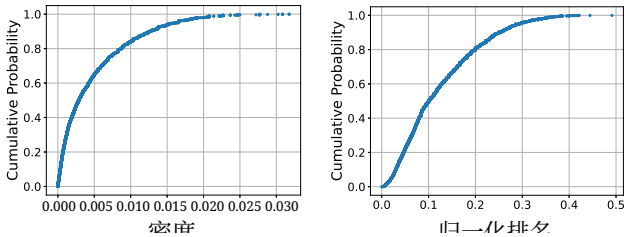
定义 4（仅基于元数据的连接图推断）。仅基于元数据的连接图推断问题，是指对于给定的一组表 T，仅使用 T的元数据 来重建连接图矩阵 A。

我们在本文中考虑的元数据包括表名、列名、列数据类型，以及简单的列统计信息（值总数、不同值的数量、空值的数量，以及最小值和最大值）。这些基本统计信息通常可在湖仓平台的目录服务中获得，用于查询优化。我们直接复用这些信息进行连接图推断。重要的是，为符合企业隐私与安全要求，我们的方法假设无法访问列中的实际数据值。

如前所述，我们的首要目标是检测 1:1 和 N:1 关系，它们对应于主键/外键连接。随后可以通过传递闭包推断多对多（N:N）关系。虽然我们专注于单列连接，但该方法也可扩展至多列连接，详见第 3.7 节。

2.2 实证分析

为了验证我们关于连接图结构的假设，我们开展了一项广泛的实证研究，以了解现实世界的连接图的特征。



(a) 密度的累积分布函数 (b) 归一化排名的累积分布函数

图 2：精选现实世界数据库模式的连接图矩阵的密度和归一化排名的累积分布函数。

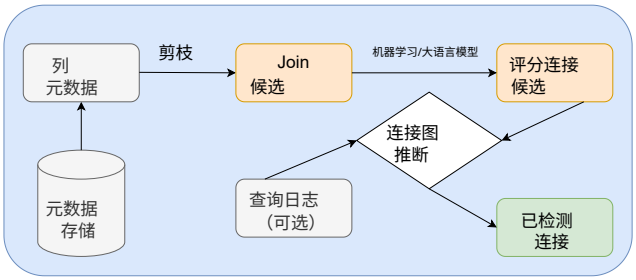


图 3：Nexus 流水线概览，该流水线仅根据元数据推断连接图。

在实证研究中，我们从 SchemaPile-Perm 集合 [12]中整理出一个包含真实感强且非平凡数据库模式的数据集。我们从 22,989 个数据库开始，首先筛选出其中包含连接关系的 11,809 个数据库（约占 51%）。随后，我们通过排除包含多列关系的 7.5% 的数据库模式，聚焦于单列连接。最后，我们过滤掉平凡数据库，移除表数量不超过三张或每张表的平均列数不超过四列的数据库。经过这一过程，我们得到一个包含 5,957 个数据库的精简数据集，用于开展分析。

如表 1 所示，这个精选数据集包含大量且多样化的现实世界数据库模式：每个数据库的表数量从 4 到 977 不等，每个数据库的列数从 17 到 4572 不等。除了基本统计信息，我们的分析还聚焦于连接图矩阵的两个关键属性：稀疏性和秩（见图 2）。

- 高稀疏性：我们使用矩阵密度来衡量稀疏性，其定义为 n^2 矩阵中的非零元素数量 $n \times n$。结果证实，现实世界的连接图极为稀疏，平均密度低于 0.005，中位数密度低于 0.003。如图 2(a) 所示，约 98% 的数据库密度低于 0.02。
- 低秩结构：我们使用归一化排名评估连接图的低秩性，其定义为矩阵 $n \times n$ 的秩$_{(A)}$ n。该指标表示矩阵距离满秩的程度。我们发现，平均和中位归一化排名分别仅为 0.12 和 0.10。如图 2(b) 所示，超过 95% 的数据库的归一化排名低于 0.3。

连接图的高稀疏性和低秩性这两个关键观察，证实了我们的直觉：模式中的大多数列不可连接，而且整体连接结构包含显著的冗余，因此可以用低维空间来表示。下一节将介绍我们基于这两个关键观察提出的新型连接图推断算法。

3 Nexus 设计

本节概述我们的连接图推断方案 Nexus，并详细介绍各个核心系统组件。

3.1 概述

图 3 展示了 Nexus 整体流程流程从元数据存储中检索给定表集合的架构信息和列级元数据。这些元数据随后用于筛除不太可能的连接列对，从而显著减少潜在连接候选项的数量。接下来，使用预训练模型为每个候选项评分，该模型可以是机器学习模型或大语言模型。这些经过评分的候选项（可选地加入从查询日志中提取的已知连接）构成连接图概率矩阵。在该矩阵中，非候选对的概率设为 0，来自查询日志的已知连接的概率设为 1.0，其余条目则填入预训练模型给出的分数。随后，将概率矩阵输入矩阵补全算法。该算法基于我们对连接图矩阵具有高度稀疏性和低秩结构的观察，并通过迭代式期望最大化（EM）过程进一步增强，以缓解预训练模型可能存在的不准确性。最终输出为推断得到的连接图矩阵，其中标识出检测到的连接关系。

3.2 元数据与连接候选项剪枝

穷举评估每一对列，将其视为潜在连接候选项，所需成本会随列数呈二次增长，因此对于拥有数千列的大型数据集而言并不切实际。为解决这一问题，我们从元数据存储库中收集列级元数据，并应用低成本筛选器，尽早排除大量不太可能的连接候选项。

我们首先剔除不太可能用作连接键的列，例如嵌套类型（如 JSON、ARRAY、MAP）、度量类型（如 FLOAT、DOUBLE、DECIMAL），以及 BLOB 和 BOOLEAN 类型。对于剩余的列，我们利用其数据类型和元数据存储库中的基本统计信息，包括基数（唯一值的数量）、空值比例以及最小值/最大值。这些统计数据通常由元数据存储库为查询优化而维护。数据类型不兼容的列对（如 INTEGER 与 VARCHAR）会立即被剪枝。此外，在推断 N:1 或 1:1连接时，我们还会应用进一步的剪枝条件，包括确保“1”侧的值唯一、验证“N”侧的基数不超过“1”侧的基数，以及确认“N”侧的值域是“1”侧值域的子集。这些剪枝条件可以用 SQL 语法实现，并在数据库系统中高效执行。

尽管存在更复杂的分析和剪枝技术，但它们通常会带来更高的计算成本。我们选择这些更简单、更高效的仅基于元数据的过滤器，因为

它们只需极低的开销，就能有效排除很大比例的真负例。

3.3 连接候选项评分

初始剪枝步骤产生的连接候选项可能仍包含许多假阳性。例如，用户 ID 列和产品 ID 列可能具有兼容的数据类型和值范围，但它们在语义上并不可连接。先前的工作 [1, 23, 29]通常采用机器学习模型，结合多种信号，例如列名的嵌入相似度和列值的 Jaccard 相似度，来预测连接关系并估计相应的概率分数。然而，先前的工作依赖于计算需要访问实际数据值的机器学习特征。此类计算代价高昂，对于大型表而言往往难以承受；在企业环境中，这些表通常包含数百万甚至数十亿行。与大多数先前工作不同，我们假设无法访问数据值。我们采用两种基于元数据的策略来估计连接候选项的概率分数：

• 预训练机器学习模型：我们仅使用元数据衍生特征对机器学习模型进行预训练。这些特征包括根据列名得出的成对特征（最大包含度、Jaccard 相似度、编辑距离与最大长度之比、Jaro-Winkler 相似度，以及加权嵌入的余弦相似度），以及列级特征（数据类型编码、基数比和空值数量比）。

• 大语言模型：我们可以将大语言模型用作预训练模型，直接提示它给出预测结果及相应的置信度等级，并将置信度分为低、中、高。这些定性的置信度等级随后会映射为预定义概率分数（例如，低 $\rightarrow 0.1$、中 $\rightarrow 0.6$、高 $\rightarrow 0.9$）。这种间接获取置信度的方式，源于大语言模型在生成可靠且经过良好校准的置信度分数方面存在已知局限性 [9, 24, 27]。

完成此步骤后，我们使用预训练机器学习模型或大语言模型的预测分数构建连接图概率矩阵；如果有可用数据，还可以加入从查询日志中提取的已确认连接列对。

尽管预训练模型很有用，但它们仍面临两个显著局限：（1）其预测本质上是局部的，只考虑单个列对的特征，而忽略连接图的全局结构；（2）在一个数据集上训练的模型可能无法有效泛化到其他数据集，而且其置信度分数可能具有误导性。我们将在第 3.4 节和第 3.5 节分别应对这些关键挑战。

3.4 通过低秩矩阵补全进行连接图推断

经过初步剪枝和评分后，我们得到一个连接图概率矩阵 S。该矩阵是部分观测的：一些条目被视为确定值，其中被剪枝的候选值为 0，来自查询日志的已知连接值为 1，其余条目则包含预训练模型给出的概率分数。我们将 S 中确定条目的索引集合定义为 Ω：$\Omega = \{(i, j) \mid (c_i, c_j)$ 已被剪枝或来自查询日志$\}$。

连接图推断的目标是恢复一个矩阵 $M \in [0, 1]^{n \times n}$，使其能够高质量地近似未知的真实连接图矩阵 A。这一恢复过程只影响 S中的不确定条目。基于我们的关键观察——现实世界的连接图高度稀疏且为低秩——我们将这一推断任务建模为低秩矩阵补全问题。

低秩矩阵补全的标准表述旨在最小化 M 的秩，同时保留已观测到的值。形式上，优化目标可写为：

$$\begin{aligned} &\text{minimize} \quad \operatorname{rank}(M) \\ &\text{subject to} \quad M_{i,j} = A_{i,j} \text{ for } (i, j) \in \Omega \end{aligned} \tag{2}$$

其中 $\operatorname{rank}(\cdot)$ 表示矩阵的秩。

由于秩函数的存在，这是一个非凸优化问题，并且已知属于 NP难问题 [7]。一种成熟的凸松弛方法是用矩阵的核范数近似秩函数 [5]，，从而得到以下优化目标：

$$\begin{aligned} &\text{minimize} \quad ||M||_* = \sum_{k=1}^{n} \sigma_k(M) \\ &\text{subject to} \quad M_{i,j} = A_{i,j} \text{ for } (i, j) \in \Omega \end{aligned} \tag{3}$$

其中，$\sigma_k(\cdot)$ 表示矩阵中第 k 大的奇异值，而核范数 $||\cdot||_*$ 则是所有奇异值之和。

虽然还存在其他低秩矩阵补全方法，例如矩阵分解 [17, 20]，，但这些方法需要预先指定待恢复矩阵的秩。在我们的场景中，这类关于矩阵秩的先验知识并不容易获得。另一方面，核范数松弛通常被认为不如矩阵分解具有可扩展性，因为它需要估计与潜在条目对应的所有参数，而这些参数会随着矩阵维度增大而呈二次增长。不过，在我们的场景中，一个关键发现大大缓解了这一顾虑：现实世界的连接图极其稀疏。具体而言，我们观察到，检测到的连接候选项（剪枝后）在待恢复矩阵中所占比例不到 5%。因此，优化中的参数数量大幅减少，使核范数方法既可行又高效。这一关键发现不仅解决了可扩展性问题，也促使我们在恢复目标中明确加入稀疏性诱导约束，以确保最终解体现连接图的这一基本属性。

此外，我们还可以获取预训练模型提供的连接图概率矩阵 S中连接候选项的置信度分数。这些分数作为先验信息，用于指导恢复 M中的潜在条目。

令 $\bar{\Omega}$ 表示 Ω的补集，其中包含概率矩阵 S中不确定条目的位置集合。为整合观测到的稀疏性和先验信念，我们提出以下优化目标：

$$\begin{aligned} &\text{minimize} \quad ||\mathrm{P}_{\bar{\Omega}}(S - M)||_F^2 \; + \; \lambda_1 \, ||M||_* \; + \; \lambda_2 \, ||M||_1 \\ &\text{subject to} \quad M_{i,j} = A_{i,j} \text{ for } (i, j) \in \Omega \end{aligned} \tag{4}$$

这里，$\mathrm{P}_{\bar{\Omega}}$ 是一个投影算子，它保留位置属于 $\bar{\Omega}$ 的条目，并将所有其他条目置零。$||\cdot||_F$和 $||\cdot||_1$ 分别表示矩阵的弗罗贝尼乌斯范数和1-范数。

上述目标函数可分解为以下三个部分：• 置信度分数正则化：项$||\mathrm{P}_{\bar{\Omega}}(S - M)||_F^2$鼓励最终矩阵 M 与

算法1： 用于迭代连接图推断

输入： $S^{(0)}$，初始概率矩阵；
$\Omega^{(0)}$， $S^{(0)}$中已知确定条目的初始集合；
Γ，最大迭代次数；
ϵ，提前停止的容差
输出： M，恢复矩阵

/* 列到其实体类型的映射（缓存） */
col_entity_type ={};
次 $t = 0, 1, \ldots, \Gamma - 1$ **执行**
  /* E 步：恢复潜在矩阵 $M^{(t+1)}$ */
  $M^{(t+1)}$ = **低_秩_矩阵_补全**($S^{(t)}, \Omega^{(t)}$);
  /* 检查是否收敛 */
  如果 $t > 0$**则**
    /* 没有剩余候选项或已到最后一次迭代 */
    如果 $\bar{\Omega}^{(t)} == \emptyset$ **或** $t == \Gamma - 1$ **则**
      跳出;
    end
    如果 $\|M^{(t+1)} - M^{(t)}\|_F \leq \epsilon$**则**
      跳出;
    end
  end
  /* M 步骤：更新概率矩阵 */
  $S^{(t+1)}, \Omega^{(t+1)}, col_entity_type$ = **更新_概率_矩阵**($M^{(t+1)}, \Omega^{(t)}, col_entity_type$)
结束
返回 $M^{(t+1)}$

预训练模型给出的置信度分数保持一致，但仅针对 $\bar{\Omega}$ 中的不确定条目。

• 低秩正则化：项 $\lambda_1 ||M||_*$ 使用核范数来促使解 M 呈现低秩特性，这与我们的关键观察一致。

• 稀疏性正则化：项 $\lambda_2 ||M||_1$ 使用 L1 范数来促进 M的稀疏性，从而有效地将不太可能的连接的概率推向零。

λ_1 和 λ_2 分别是对低秩项和稀疏项的正则化强度进行加权的超参数。

请注意，恢复矩阵 M 在 $[0, 1]$, 中包含实数值，而不是二元指示变量。为生成最终的二元决策，我们对 M的潜在条目应用阈值 θ。所得的决策矩阵$\hat{M} \in \{0,1\}^{n \times n}$ 定义如下：

$$\hat{M}_{i,j} = \begin{cases} M_{i,j} & \text{if } (i,j) \in \Omega \\ \mathbf{1}(M_{i,j} \geq \theta) & \text{otherwise} \end{cases} \tag{5}$$

其中，$\mathbf{1}(\cdot)$是指示函数。

算法 2: 更新 the 概率 矩阵（**更新_概率_矩阵**）

输入： S，概率矩阵；
Ω， S中的已知确定条目；
col_entity_type，列到实体类型映射；
$low_threshold$，用于提示大语言模型的阈值实体类型和类型匹配检查；
$high_threshold$，将潜在条目提升为强正例的阈值；
δ，实体类型不匹配的惩罚因子
输出： 已更新 S、 Ω 和 col_entity_type

$join_candidates$ = {(i<style 标识符='3'>, j) | (i<style 标识符='8'>, j) $\in \bar{\Omega}$ 和 $i < j$};
/* 遍历连接候选项 */
对于 $(i, j) \in join_candidates$ **执行**
  如果 $S[i][j] < low_threshold$**则**
    继续;
  end
  如果 $i \notin$列_实体_类型 **则**
    $col_entity_type[i]$ = **提示_大语言模型_用于_实体_类型**（col_i）；
  end
  如果 $j \notin$列_实体_类型 **那么**
    $col_entity_type[j]$ = **提示_大语言模型_用于_实体_类型**（col_j）；
  end
  /* 提升类型匹配条目的概率 */
  如果 $col_entity_type[i] == col_entity_type[j]$ **或** **提示_大语言模型_用于_软_类型_匹配**（col_i, col_j）**then**
    如果 $S[i][j] \geq high_threshold$**那么**
      $S[i][j] = S[j][i] = 1.0$;
      /* 添加新的已知条目 */
      $\Omega.add(\{(i,j),(j,i)\})$;
    else
      $S[i][j] = S[j][i] = high_threshold$;
    end
  else
    /* 未匹配条目的衰减概率 */
    $S[i][j] *= \delta$; $S[j][i] *= \delta$;
  end
结束
返回 $S, \Omega, col_entity_type$

3.5 基于期望最大化算法的迭代连接图推断

如公式4所示，我们将恢复矩阵与预训练机器学习/大语言模型确定的概率矩阵进行（部分）对齐。这种方法假设该模型能够准确估计

连接候选项的概率分数。然而，将预训练模型应用于与训练数据集分布迥异的未见数据集时，这一假设可能并不成立。因此，强制恢复矩阵中的潜在条目与可能不准确的估计值相匹配，会降低结果质量。为解决这一问题，我们提出了一种期望最大化（EM）算法（算法1），利用大语言模型提供的指导，迭代改进低秩矩阵补全的质量。

EM 是机器学习领域广泛采用的一种方法，用于计算含有不完整变量或潜变量的模型的最大似然估计 [10]。该算法如算法1所示，通过两个主要步骤迭代运行。期望步骤 （E 步骤）涉及低秩矩阵补全，用于估计潜在条目的值。此估计基于观测数据 Ω 和机器学习/大语言模型给出的概率分数 $S^{(t)}$。这一 E 步骤可解释为：根据观测数据以及机器学习/大语言模型提供的先验知识，确定潜在条目的后验概率。随后，在最大化步骤 （M 步骤）中，我们通过使用大语言模型评估列实体类型的语义兼容性，来优化概率矩阵。具体而言，我们利用大语言模型的语义能力，确保有效连接满足一项关键条件：可连接的两列必须具有语义等价的实体类型。

我们将列实体类型定义为一种自由文本注释，用于描述某一列在其关系表中所表示的具体语义。例如，食品订单表中的ID列最好标注为“食品订单ID”，而用户表中的ID列则应标注为“用户ID”。借助大语言模型强大的语义理解能力，我们使用大语言模型对列实体类型进行标注，并确定两个实体类型之间的语义等价性。实体类型匹配是有效连接的必要条件，并能大幅减少虚假连接候选项的数量（例如，两个ID列可能存在偶然的值重叠，但实际上指向不同类型的实体）。需要特别指出的是，我们的方法不依赖固定的实体类型词汇表，因为列语义可能因不同的表和数据库而异，且现有实体类型词汇表都无法全面覆盖细微多样的列语义范围。

如算法 2 所详述，M 步骤会遍历潜在条目（连接候选项），并提示大语言模型根据列名及其对应的表名识别列的实体类型。对于得分超过预定义低阈值的每个候选对，若其尚无实体类型，我们会提示大语言模型推断其实体类型。为降低大语言模型调用的计算和货币成本，我们会缓存已识别的实体类型。随后，我们逐字符比较这些实体类型。若比较未通过，则进一步提示大语言模型进行软检查，以判断这些实体类型在语义上是否匹配。实体类型的对齐是有效连接的前提。我们发现，大语言模型特别擅长进行此类细致的语义比较。若实体类型对齐，且相应的概率得分超过预定义的高阈值，我们就将该候选对提升为强正例，将其概率得分设为 1，并将其从潜在条目集合中移除。若类型匹配但概率得分低于高阈值，我们则提高该概率得分至该阈值，确保该候选对在下一次 EM迭代中再次接受大语言模型检查。这种设计考虑到了预训练模型对该候选对给出的初始概率估计较低，以及大语言模型输出固有的随机性。若实体类型检查未通过，我们会按特定系数 δ降低该候选对的概率得分；在所有实验中，该系数均设为 0.5。在实现中，为进一步降低大语言模型调用的延迟，我们会在一个提示中批量处理多个请求，每批 24 个。

EM算法的迭代过程会持续进行，直到达到预设的最大迭代次数 Γ，或实现收敛。收敛由恢复矩阵的变化情况决定（以弗罗贝尼乌斯范数进行量化）；当该变化低于指定容差 ϵ时，即判定为收敛。我们在所有实验中均将 ϵ 设置为 $1e^{-5}$。

对于延迟敏感场景，我们可以将系统配置为仅执行一轮 EM算法（仅进行低秩矩阵补全）。这种快速模式称为 Nexus-Fast，尤其适用于预训练模型能够提供强先验的情况。它跳过 EM算法中的所有大语言模型调用，从而确保推理速度达到最大值，同时仍能优于预训练模型，并取得与完整 EM算法相比具有竞争力的结果。在下文中，我们将完整的 EM算法称为 Nexus-EM，以区别于 Nexus-Fast。

3.6 通过核心子矩阵缩减进行优化

上述过程会恢复一个大小为 M 的矩阵，其中 $n \times n$ 是表集合中列的总数。对于大型表集合， n 可能非常大。然而，由于我们的主要目标是推断由连接候选项确定的潜在条目，而这些候选项通常仅涉及所有列中的一小部分，因此我们可以改为恢复一个更小的矩阵，我们称之为 核心子矩阵：

定义5 （核心子矩阵）。给定一个对称的 $n \times n$ 矩阵M，核心子矩阵 M' 定义为从 M 中删除所有仅包含零元素的行和列后得到的子矩阵。形式化地，令 $I = \{i \mid \exists j, M_{i,j} \neq 0\}$ 为对应于非零行的索引集合（由于 M 是对称的，M' 这些行也等价地对应于非零列）。那么，核心子矩阵可表示为 $M[I, I]$。其维度为 $n' \times n'$，其中 $n' = |I|$。

由于大多数列都不是连接键，尤其是分析型工作负载中无处不在的度量导向列，因此核心子矩阵 M' 通常远小于 M（即 $n' << n$）。重要的是，我们基于低秩矩阵补全的方法仍然适用，因为我们发现 M' 也具有较高的稀疏性和类似于完整邻接矩阵的低秩结构。

恢复较小的核心子矩阵 M' 具有显著的实践优势。从经验来看，其维度通常不到 M的一半，因此运行时间更短，内存使用量也更少。主要缺点是，连接候选项剪枝并不完美，某些相关连接键可能会被排除在 M' 之外，且无法恢复。通过实验，我们发现，仅运行低秩矩阵补全（即 Nexus-Fast）时，这项优化几乎能带来 25% 的加速，因此我们默认在 Nexus-Fast 中启用该优化。对于 Nexus-EM，这项优化的效果不那么显著，因为主要的性能瓶颈在于大语言模型调用（来自远程服务）。因此，我们选择恢复完整矩阵，以免遗漏任何相关连接键。

表2：实验数据集及其统计数据。

数据集	# 模式	# 表	#列	# 候选	密度	归一化排名
TPC-H	1	8	61	1,830	0.004	0.20
TPC-DS	1	24	425	90,100	0.001	0.08
BIRD-SQL	11	75	806	324,415	0.003	0.10
REAL	3	30	1156	667,590	0.001	0.02

3.7 扩展至多列连接

虽然我们是在单列连接的背景下讨论这一方法，但它很容易适配多列连接场景。其核心思想保持不变：我们将表中的每一组 k 列视为一个概念单元。随后，这些单元将应用第 3.2 节所述的相同剪枝规则。此扩展会使低秩矩阵补全任务的概率矩阵变大。具体而言，对于一组表 T，矩阵维度变为 $m \times m$，其中 $m = \sum_{\tau \in T} \binom{n_\tau}{k}$ 和 n_τ 是表 τ中的列数。矩阵中的每个条目都表示一对 k列单元的连接概率。

至关重要的是，这个扩展后的矩阵同样具有高稀疏性和低秩结构。因此，我们可以直接应用相同的矩阵补全和 EM 算法（第 3.4 节和第 3.5 节）来检测多列连接。事实上，单元中包含多列会增大矩阵规模，而多列连接的总数通常更少。这往往会使连接图矩阵具有更高的稀疏性和更低的秩，从而有利于我们的算法。此外，第 3.6 节中的核心子矩阵约简优化同样适用，可用于进一步提升效率。

尽管具备这种能力，我们对 SchemaPile-Perm 数据集的分析显示，多列连接仅占真实世界数据库模式的一小部分（7.5%）。因此，我们的实验评估侧重于更常见的单列连接。

4 实验

本节介绍实验结果，涵盖质量和效率比较，以及超参数消融研究。除非另有说明（如第 4.7 节所述），所有实验均采用仅元数据设置。

4.1 实验设置

数据集。 表2列出了用于评估 Nexus 的数据集，涵盖标准基准测试和复杂的多源场景。我们纳入了 TPC-H 和 TPC-DS，遵循当前最优（SOTA）工作 [23]。由于这些基准测试代表单模式环境，我们还通过合并 BIRD-SQL [22] 开发集中的 11 个数据库创建了一个多源场景。此外，我们引入了 REAL，这是一个跨越三个不同来源的生产数据集，其中仅有元数据和查询日志可用。对于真实值，我们对合成数据集依赖已定义的主键/外键约束，而对 REAL 则从查询日志中推导关系，因为它缺乏显式约束。REAL 反映了生产环境的"混乱性"，为连接检测提供了严格测试。尽管我们使用 SchemaPile-Perm 数据集来分析连接图结构，但由于它缺少实际数据值和列统计信息，我们无法将其用于评估。相反，我们将该数据集用于机器学习模型训练（见下文）。最后，表2的第5列报告了最坏情况下的候选对数量，反映了随着数据集规模增大而增长的计算复杂度。

基线方法。 首先，我们评估独立的机器学习模型：XGBoost 和 GPT-4o。对于 XGBoost，我们使用来自 SchemaPile-Perm（第 2.2 节）的大规模数据库模式语料库进行训练，并利用来自 [1] 的元数据特征超集[1]。对于 GPT-4o，我们使用基于元数据的零样本提示。这两个模型也作为我们系统的先验（分别记为 Nexus-XGBoost 和 Nexus-GPT-4o）。

其次，我们将系统与 SemPy [26]（来自微软 Fabric）和 Auto-BI（当前最优方法）进行比较。除了使用数据值的原始版本外，我们还针对主要的仅元数据评估对二者进行了适配。对于 SemPy，我们禁用了基于值的重叠检查，转而依赖列名相似度（Ratcliff/Obershelp）及其他启发式方法。对于将连接检测定义为图优化问题的 Auto-BI，我们用基于元数据特征训练的 XGBoost 模型替换了其原始的 XGBoost 模型。

最后，我们还纳入了 Aurum [15] 和 DeepJoin [14]。这些方法关注相关的"可连接表搜索"问题，主要依赖数据值；因此，我们仅在允许访问数据的实验中评估它们（第 4.7 节）。我们还尝试评估 JOSIE [33]，但由于它始终超时而将其排除（多源数据集需要 >1 小时），这一局限在先前的工作中也曾提及 [11]。

指标。 由于检测到的连接候选项存在类别不平衡，我们使用 F1分数、精确率和召回率评估所有方法的质量。所报告的数值代表我们针对每个数据集尽力调优后，各方法取得的最佳结果。为比较效率，我们以秒为单位报告运行时间。

硬件。 所有实验均在一台 Azure虚拟机（标准 NC12s v3）上进行，该虚拟机配备 12 个 vCPU 和 224 GiB 内存。

4.2 质量比较

表3展示了四个数据集上所有仅使用元数据的方法的 F1分数、精确率和召回率。Nexus 始终以显著优势超越所有基线方法。例如，Nexus-GPT-4o（使用 GPT-4o 先验的 Nexus）在 TPC-H、TPC-DS、BIRD-SQL 和 REAL 数据集上分别比此前的当前最优方法 Auto-BI 高出 32、37、51 和 53 个 F1 分数。类似地，在这些数据集上，SemPy 的表现分别比 Nexus-GPT-4o 低 47、46、54 和 35 个 F1 分数。

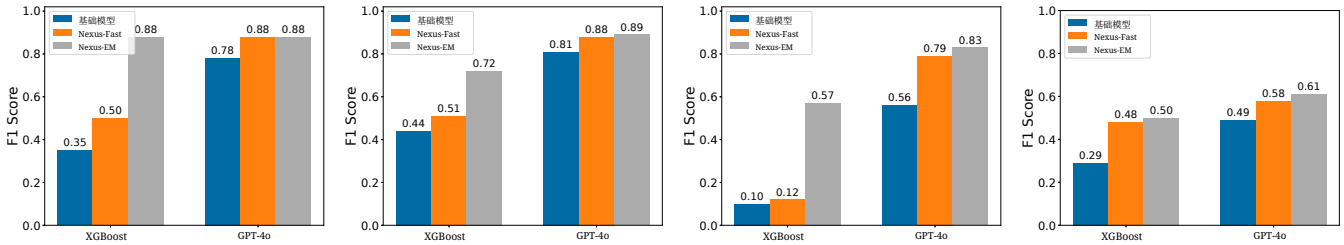
仅使用元数据特征并在 SchemaPile-Perm 数据集上预训练的 XGBoost 难以泛化到测试数据集，在 TPC-H 和 BIRD-SQL 上取得了最低的 F1 分数。相比之下，即使先验较弱，Nexus-XGBoost 仍显著提升了 F1 分数：在 TPC-H 上提升 53 个百分点，在 TPC-DS 上提升 28 个百分点，在 BIRD-SQL 上提升 47 个百分点，在 REAL 上提升 21 个百分点。这一显著改进归因于 Nexus 的迭代优化（算法1中的 EM算法），该优化利用了连接图的低秩结构以及基于大语言模型的列实体类型检查。

即使 GPT-4o 展现出强劲的开箱即用表现，Nexus 仍能改进 GPT-4o 的先验。GPT-4o 在公开的 TPC-H 和 TPC-DS 基准测试中取得出色的结果质量，这并不意外，因为它很可能在训练期间记住了这些数据集的连接基准真值。尽管

[1]这些特征包括拼接后的表名称和列名称词元的包含关系、Jaccard 相似度、编辑距离比率和 Jaro-Winkler 相似度，列数据类型（整数和字符串）的布尔型指示变量，以及 [1] 中使用的特征。

表3：四个数据集上仅元数据方法的质量比较。

	TPC-H			TPC-DS			BIRD-SQL			REAL		
	F1	精确率	召回率	F1	精确率	召回率	F1	精确率	召回率	F1	精确率	召回率
XGBoost	0.35	0.21	1.0	0.44	0.34	0.62	0.10	0.05	0.99	0.29	0.17	0.89
GPT-4o	0.78	0.64	1.0	0.81	0.68	0.99	0.56	0.39	0.98	0.49	0.34	0.88
SemPy	0.41	0.26	1.0	0.43	0.50	0.38	0.29	0.22	0.47	0.26	0.29	0.23
Auto-BI	0.56	0.39	1.0	0.52	0.61	0.45	0.32	0.42	0.26	0.08	0.15	0.05
Nexus-XGBoost（我们的方法）	**0.88**	0.78	1.0	0.72	0.58	0.95	0.57	0.44	0.85	0.50	0.42	0.63
Nexus-GPT-4o（我们的方法）	**0.88**	0.78	1.0	**0.89**	0.88	0.90	**0.83**	0.86	0.80	**0.61**	0.49	0.80



TPC-H **TPC-DS** **BIRD-SQL** **REAL**

图4：基线模型、Nexus-Fast 和 Nexus-EM 在四个数据集上的 F1 分数对比。

BIRD-SQL 也是公开的，但将全部 11 个数据库合并为一个数据集，会导致 GPT-4o 产生数量不少的假阳性（其精确率低于近乎完美的召回率便是明证）。对于未见过的 REAL 数据集，GPT-4o 能正确预测部分真实连接，但假阳性率很高。

预训练模型作为 Nexus 中的模块化组件。虽然我们使用 GPT-4o 来代表一种强先验，但它可以替换为任何先进的机器学习模型。Nexus-XGBoost 和 Nexus-GPT-4o 的结果凸显了我们框架的鲁棒性：更准确的先验自然会带来更好的结果，但即使初始化时使用显著较弱的模型，Nexus 仍能提供足够的质量。特别是在 TPC-H 上，Nexus-XGBoost 达到了与 Nexus-GPT-4o 相当的高质量结果，尽管其底层的 XGBoost 先验明显弱于 GPT-4o 先验（F1 分数分别为 0.35 和 0.78）。

Auto-BI 和 SemPy 在所有数据集上的表现都较差。这一结果并不令人意外，因为这两种方法最初的设计都假设可以访问数据值。无法访问数据导致无法使用包含依赖检查来剪枝候选，从而带来更高的假阳性率。此外，Auto-BI 的性能依赖于相对有效的基线模型，而由于缺少从数据值中提取的特征，其结果受到显著影响。如第 4.7 节所示，在获得完整的数据访问权限后，这两种方法的结果都有大幅改善。然而，对于仅根据元数据推断连接图这一具体任务，Auto-BI 和 SemPy 都不具备竞争力。

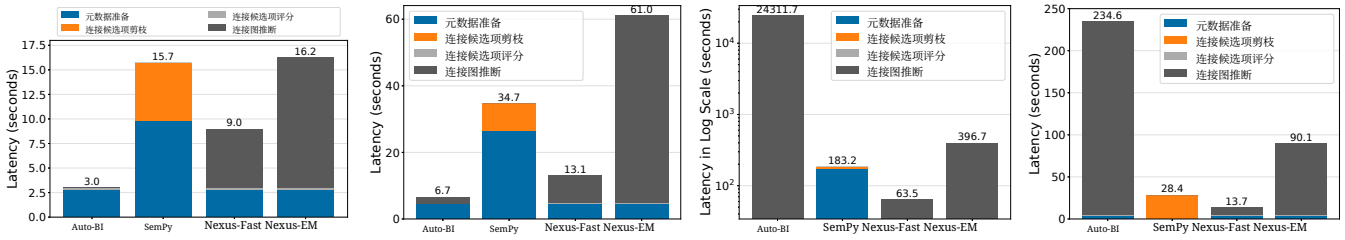
最后，我们观察到一个一致的趋势：随着数据集复杂度增加，从简单的单一模式设置（TPC-H 和 TPC-DS），到多源场景（BIRD-SQL），再到复杂的生产环境（REAL），所有方法的结果质量都在下降。具体而言，所有方法在多源数据集上都面临较高的假阳性率（因此精确率较低）。REAL 数据集并未像合成数据那样呈现一幅乐观的图景，而是如实衡量了现有方法在实际生产环境中（即数据湖或湖仓中存储的来自多个来源的数据集）所能达到的水平。

4.3 快速模式推断

默认情况下，Nexus 在其 EM 算法中使用大语言模型，推断列的语义实体类型并执行软类型匹配。尽管功能强大，这些大语言模型 API 调用会增加延迟，并产生相当可观的费用。对于对延迟或预算敏感的场景，我们提供 Nexus-Fast 这一快速模式，它仅在核心子矩阵上使用低秩矩阵补全（不使用迭代式 EM 算法），如第 3.6 节所述。

为便于比较，我们将采用 EM 迭代的默认算法称为 Nexus-EM，以区别于 Nexus-Fast。在图4中，我们比较了 Nexus-Fast 和 Nexus-EM 的 F1分数，并使用 XGBoost 和 GPT-4o 作为基线模型，为连接候选项评分。总体而言，在四个数据集上，Nexus-Fast 都能显著提升两种基线模型的结果质量，而 Nexus-EM 的结果最佳。

当使用较弱的基础模型（如 XGBoost）时，由大语言模型增强的 Nexus-EM 比 Nexus-Fast 取得更强的结果，尽管其运行时间更长（如下一节所示）。具体而言，Nexus-EM 在 TPC-H、TPC-DS 和 BIRD-SQL 数据集上分别比 Nexus-Fast 高出 38、21 和 45 个 F1 分数。当弱模型提供的先验较差时（如 BIRD-SQL 所示），Nexus-Fast 仅将结果质量提高了 2 个 F1 分数。然而，当同一基础模型具有一定预测价值时



TPC-H **TPC-DS** **BIRD-SQL** **REAL**

图5：TPC-H、TPC-DS、BIRD-SQL 和 REAL 数据集上的效率对比。Auto-BI 采用了与 Nexus 相同的元数据准备、连接候选项剪枝和连接候选项评分实现。

如在 TPC-H、TPC-DS 和 REAL 上，Nexus-Fast 分别提升了 15、7 和 19 个 F1 分数。

相比之下，采用 GPT-4o 这类强大的基础模型时，Nexus-Fast 的结果质量与 Nexus-EM 相当。质量差距显著缩小，在 TPC-DS 上仅为 1 个 F1 分数，在 BIRD-SQL 上为 4 个，在 REAL 上为 3 个，而在 TPC-H 上没有差异。与仅使用基础模型相比，Nexus-Fast 在这四个数据集上的结果分别提升了 10、7、23 和 9 个 F1 分数。下一节将详细介绍 Nexus-Fast 在延迟方面的优势。

4.4 效率对比

图5展示了四种方法在所有数据集上的效率对比。此处报告的是采用 XGBoost 基础模型时 Nexus-Fast 和 Nexus-EM 的运行时间。

我们将端到端延迟分解为四个关键组成部分：元数据准备、连接候选项剪枝、连接候选项评分和连接图推断，如流水线所示（图 3）。Auto-BI、Nexus-Fast 和 Nexus-EM 均包含这四个组成部分，而 SemPy 仅涉及元数据准备和连接候选项剪枝。为确保在仅使用元数据的适配中进行公平比较，我们用 Nexus 中效率更高的实现替换了 Auto-BI 的前三个组成部分。

在所有方法中，Nexus-Fast 展现出出色的效率，尤其是在包含大量表和列的多源数据集上。在 BIRD-SQL 和 REAL 数据集上，Nexus-Fast 的速度比 Nexus-EM 快 6 倍以上。这种性能差距在预期之中，因为 Nexus-Fast 仅对核心子矩阵执行低秩矩阵补全，而 Nexus-EM 则需要在每次迭代中额外调用大语言模型 API，对语义列类型进行检查。Nexus-EM 的延迟也取决于算法运行的迭代次数。本文中，我们对 Nexus-EM 运行五次迭代，这足以在每个数据集上取得最佳结果。尽管如此，即使面对大型多源数据集，Nexus-EM 也能在几分钟内完成连接检测。Auto-BI 在 BIRD-SQL 和 REAL 数据集上的延迟最高，分别比 Nexus-Fast 慢 382 倍和 17 倍。这表明，Auto-BI 的基于图的优化技术难以扩展到更大的多源数据集。

综合来看，图4和图5表明，在使用一个相当准确的基础模型时，Nexus-Fast在质量与效率之间提供了颇具吸引力的权衡。

4.5 超参数消融研究

我们进行消融研究，以分析关键超参数对F1分数的影响。

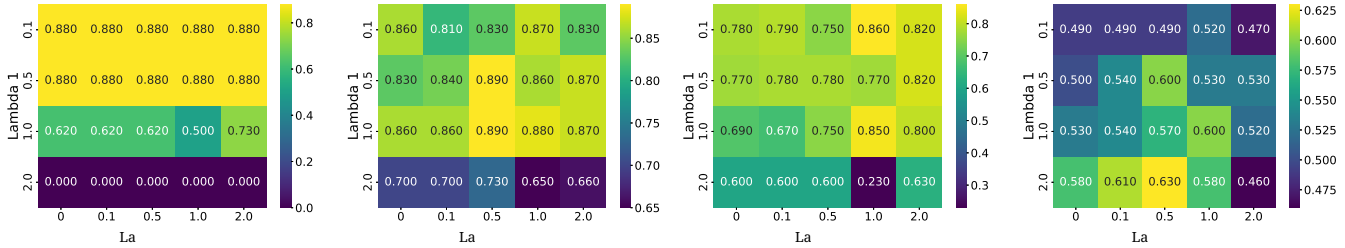
4.5.1 权重 λ_1 和 λ_2 在低秩矩阵补全目标函数中的作用。参数 λ_1 和 λ_2 用于平衡低秩矩阵补全优化目标（公式4）中的低秩正则化与稀疏性正则化。我们评估了20种组合构成的网格，其中 λ_1 取四个值（0.1、0.5、1.0、2.0），λ_2 取五个值（0、0.1、0.5、1.0、2.0）。图6展示了 Nexus-GPT-4o 的 F1分数热力图。总体而言，Nexus-GPT-4o 在正则化强度适中的情况下表现最佳。具体来说，$\lambda_1 \in \{0.5、1.0\}$在 TPC-H、TPC-DS 和 BIRD-SQL 上表现最佳，而对于 REAL，较大的 λ_1 最为理想。这与数据集的特性相符：REAL 的归一化排名最低（为 0.02，而 TPC-H 为 0.20，见表2），因此需要更强的低秩正则化。由于篇幅所限，我们省略了 Nexus-XGBoost 的热力图，其呈现出类似的规律。

*4.5.2 EM*算法中的阈值。我们的 EM 算法引入了两个参数：*low_threshold* 和 *high_threshold*。由于我们观察到基线模型的假阴性率较低，因此将 *low_threshold* 固定为 0.5，这也是基线模型的判定阈值。换言之，如果某个基线模型将候选对预测为负例，我们就会在 EM 算法中跳过对该候选对进行耗时的大语言模型语义检查。这样，Nexus-EM 就能专注于预测为正例的候选对，确保即使面对较大的数据集，也能在几分钟内完成执行。另一方面，*high_threshold* 控制我们多快将类型兼容的候选对标记为确定的正例。图7展示了 Nexus-XGBoost 和 Nexus-GPT-4o 的 F1分数对 *high_threshold* 的敏感性。总体而言，这两种方法对 *high_threshold*的变化都相对稳健。

4.6 有查询日志时

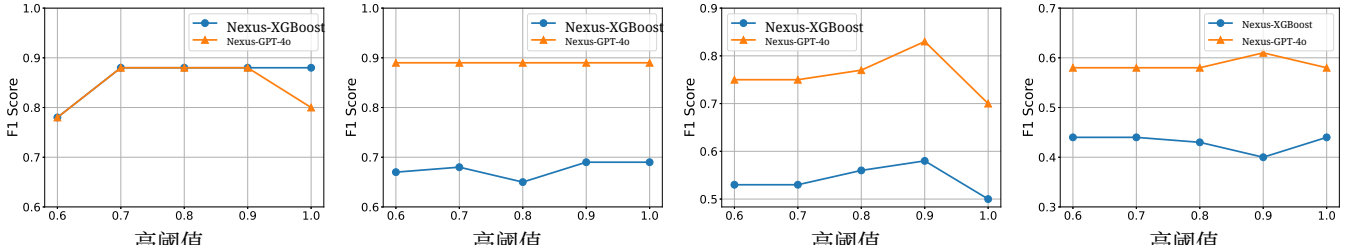
在实际应用中，查询日志通常是可用的，并且可以提供部分乃至完整的真实连接关系集合。我们评估了在可用连接比例各异的情况下 Nexus 的结果质量；这些连接是从真实情况中随机抽样得到的。图8展示了在四个数据集上，随着已知连接比例的增加，Nexus-EM 的 F1分数变化。

总体而言，通过查询日志获得更多真实连接，可以提升 Nexus 的结果质量。这是因为采样得到的真实



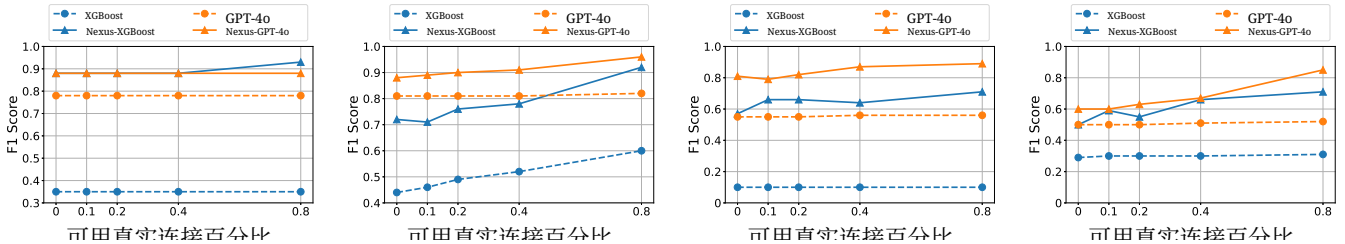
TPC-H **TPC-DS** **BIRD-SQL** **REAL**

图6：F1分数热力图 o 在低秩矩阵补全目标函数中，Nexus-GPT-4o 关于损失权重 λ_1 和 λ_2 n.



TPC-H **TPC-DS** **BIRD-SQL** **REAL**

图7：Nexus-EM 算法中高_阈值对应的 F1分数。



TPC-H **TPC-DS** **BIRD-SQL** **REAL**

图8：ba 基础模型的 F1分数 （虚线）与 Nexus-EM（实线）随可用真实连接比例的变化 s.

连接提供了有关底层真实邻接矩阵连通性的宝贵结构信号。例如，在 TPC-DS 上，当已知 40% 的真实连接时，Nexus-GPT-4o 的 F1分数达到 0.91，Nexus-XGBoost 达到 0.78。当真实连接的已知比例提高到 80% 时，这些分数分别升至 0.96 和 0.92。已知连接在 REAL 数据集上的益处也十分显著：当已知 80% 的真实连接时，Nexus-XGBoost 和 Nexus-GPT-4o 的 F1 分数均提高了 20 多分。相比之下，基线模型 XGBoost 和 GPT-4o 从已知连接增加中获得的提升微乎其微，如图8中的平坦虚线所示。这是因为基线模型在所有数据集上都已达到（几乎）完美的召回率（见表3）。这有力地证明，观察到的质量提升正是由我们基于低秩矩阵补全的 EM算法带来的。

4.7 当数据值可用时

虽然 Nexus 是为数据访问受限的企业环境而设计的，但在数据值可用时仍然非常有效。我们在 TPC-H 和 TPC-DS 数据集上，将 Nexus-EM 与 Aurum、DeepJoin、XGBoost、Auto-BI 和 SemPy 进行比较。出于篇幅考虑，我们不纳入多源数据集。在拥有完整数据访问权限时，对于所有相关方法，我们改用 Auto-BI 的 XGBoost 模型，该模型利用数据值中的额外特征进行训练，以取代先前实验中的仅元数据模型。

图 9 展示了六种方法的 F1 分数。与仅使用元数据的设置不同，XGBoost、Auto-BI 和 SemPy 现在表现不相上下，这凸显了它们对数据值的高度依赖。这种对比进一步印证了我们先前的观察：缺少数据信号时，利用全局连接图结构对于推断可连接性至关重要。尽管如此，Nexus 仍然表现最佳，

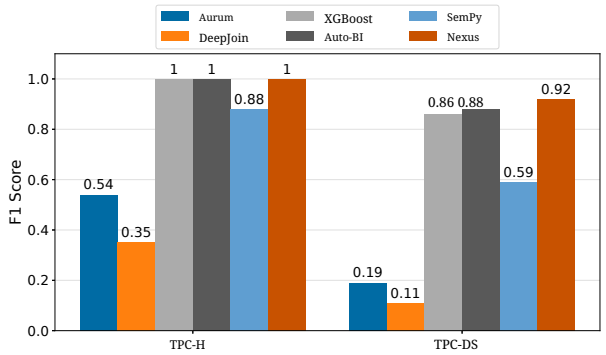


图 9：当数据值可用时，TPC-H 与 TPC-DS 上 F1分数的比较。

在 TPC-H 和 TPC-DS 上均是如此。尽管这些数据集都符合 Auto-BI 针对其优化的严格雪花模式，Nexus 在 TPC-DS 上的表现仍优于 Auto-BI。

除了利用基于值的特征提升机器学习预测外，访问数据值还能进行包含依赖测试，这是一种有效的连接候选项剪枝技术。该测试能有效排除许多假阳性；若仅依据元数据，这些假阳性可能会被认为具有很高的可连接性。在 TPC-DS 上，连接候选项数量从 3351 个降至 1232 个。在 TPC-H 上，降幅更为显著：从 183 个候选项降至仅 30 个。这种激进的剪枝是 XGBoost、Auto-BI 和 Nexus 都能在 TPC-H 上取得完美 F1 分数的关键原因之一。

作为"可连接表搜索"问题的代表性方法，Aurum 和 DeepJoin 的表现落后于所有其他方法。它们要求用户指定查询列，并且在原始评估中仅关注一部分真实连接键。在推断完整连接图的设置下，若将所有具有唯一值的列都用作查询列，这两种方法都会产生大量假阳性。DeepJoin 还要求为 top-k 搜索设置一个固定的 k，且其最初是为字符串类型列设计的，而测试数据集中超过 80% 的连接涉及数值列。

5 相关工作

连接检测在文献中得到了广泛研究，涉及关系数据库中的主键/外键检测 [6, 18, 29, 31],、面向分析型工作负载的商业智能模型构建 [1, 23], 、Web 语料库中的相关表搜索 [3, 4, 28, 30] 以及数据湖中的相关表搜索 [2, 11, 13– 16, 25, 32–34]。以往工作通常针对两种场景之一：（1）可连接表搜索，即一种"点查询"设置，用于查找可与给定输入表/列连接的表；（2）连接图推断，用于查找表集合中的所有连接关系。本工作聚焦于后一种场景，且无需用户输入。

传统的连接检测方法依赖名称相似度和值重叠等启发式方法 [6, 26]。例如，Chen 等人 [6] 结合表名、列名和数据值等信号，借助包含关系及其他约束来剪除不太可能的连接候选，从而区分真实关系与虚假候选。类似地，Jiang 等人 [18] 利用一套全面的剪枝规则，同时检测主键和外键。然而，这些方法往往计算开销大，而且会漏掉语义匹配。Zhang 等人 [31] 通过计算列之间的分布相似度来检测主键/外键关系，所用指标为推土机距离，但这种方法扩展性较差，处理 10 GB 数据需要 2.5 小时以上 [6]。

基于嵌入的方法将表和列的元数据表示为向量，并使用近似最近邻（ANN）搜索来查找连接候选项 [2, 8, 14]。虽然这些方法能够发现不明显的语义连接，但其相似度指标无法保证结果正确，因此往往会产生大量假阳性。

近期研究已转向基于机器学习的方法。Rostin 等人 [29] 提出利用各种机器学习模型，根据已知外键学习主键/外键关系。然而，后续研究表明，这类预训练模型往往无法泛化到未见过的数据集 [6, 23]。在 Nexus 中，我们通过采用 EM算法来解决这一局限：该算法在大语言模型的指导下，迭代改进预训练机器学习模型的预测结果。

我们的工作与 Bharadwaj 等人 [1] 和 Lin 等人 [23], 近期提出的方法最为相关；这些方法旨在以最少的用户输入，自动且高效地构建完整的连接图。与早期工作 [29], 类似，Bharadwaj 等人 [1] 的方法依赖于在大规模数据上训练的机器学习模型。虽然该方法在分布内数据集上效果良好，但并未解决分布外泛化问题，也忽视了连接图的全局结构。Auto-BI [23] 针对特定的商业智能场景，通过在机器学习预测的基础上应用 k-最小成本树形结构优化来构建连接图。然而，对于模式较宽松的数据集而言，强制采用雪花模式的限制过于严格。相比之下，我们的方法适用范围更广，且不对工作负载特征作任何假设。我们的方法基于这样一些关键观察：现实场景中的连接图高度稀疏，并呈现低秩结构。利用这一洞见，我们证明了该方法在各种不同的数据集上都具有稳健的表现。

最后，除 [1], 外，大多数先前的工作都假设可以访问数据值，但在大型表上进行基于值的特征计算，计算成本高昂，而且在企业环境中，出于合规和审计要求 [1], 通常并不现实。相比之下，Nexus 专为仅依赖元数据运行而设计，同时仍能高效地提供高质量结果。

6 结论

在这项工作中，我们提出了 Nexus，这是一种新颖的、仅依赖元数据的连接图推断方案，专为数据隐私要求严格的企业环境而设计。我们的方法基于对大量真实数据库模式进行分析后得出的关键洞见：连接图不仅高度稀疏，而且始终呈现低秩结构。我们将连接图推断建模为低秩矩阵补全问题，以利用这些特性。为克服现有预训练模型泛化能力不足的问题，我们提出一种新颖的 EM 算法，通过引入大语言模型对语义兼容性的评估，迭代地优化连接概率。包括真实生产数据集在内的大量评估结果表明，Nexus 在效果和效率方面均取得了显著提升。

参考文献

[1] Sagar Bharadwaj、Praveen Gupta、Ranjita Bhagwan 和 Saikat Guha。2021。大规模发现相关数据。《VLDB学报》14卷8期（2021），1392–1400。[2] Alex Bogatu、Alvaro A. A. Fernandes、Norman W. Paton 和 Nikolaos Konstantinou。2020。数据湖中的数据集发现。载于第36届 IEEE 国际数据工程会议（ICDE 2020），美国得克萨斯州达拉斯，2020年4月20日至24日。IEEE，709–720。[3] Michael J. Cafarella、Alon Y. Halevy 和 Nodira Khoussainova。2009。关系型网络的数据集成。《VLDB学报》2卷1期（2009），1090–1101。[4] Michael J. Cafarella、Alon Y. Halevy、Daisy Zhe Wang、Eugene Wu 和 Yang Zhang。2008。WebTables：探索网络表格的力量。《VLDB学报》1卷1期（2008），538–549。[5] Emmanuel J. Candès 和 Benjamin Recht。2009。通过凸优化实现精确矩阵补全。《计算数学基础》9卷6期（2009），717–772。[6] Zhimin Chen、Vivek R. Narasayya 和 Surajit Chaudhuri。2014。在 Microsoft SQL Server PowerPivot f or Excel 中快速检测外键。《VLDB学报》7卷13期（2014），1417–1428。[7] Alexander L. Chistov 和 Dima Grigoriev。1984。代数闭域理论中量词消去的复杂度。载于《计算机科学数学基础 1984》，捷克斯洛伐克布拉格，1984年9月3日至7日，会议论文集（《计算机科学讲义》第176卷），Michal Chytil 和 Václav Koubek（编）。Springer，17–31。[8] Tianji Cong、James Gale、Jason Frantz、H. V. Jagadish 和 Çagatay Demiralp。2023。WarpGate：面向云数据仓库的语义连接发现系统。载于第13届创新数据系统研究会议（CIDR 2023），荷兰阿姆斯特丹，2023年1月8日至11日。www.cidrdb.org。[9] Tianji Cong、Fatemeh Nargesian、Junjie Xing 和 H. V. Jagadish。2025。OpenForge：概率元数据集成。《VLDB学报》18卷9期（2025），2914–2927。[10] Arthur P Dempster、Nan M Laird 和 Donald B Rubin。1977。通过 EM 算法从不完整数据中求最大似然。《皇家统计学会杂志：B辑》（方法论）39卷1期（1977），1–22。[11] Yuhao Deng、Chengliang Chai、Lei Cao、Qin Yuan、Siyuan Chen、Yanrui Yu、Zhaoze Sun、Junyi Wang、Jiajun Li、Ziqi Cao、Kaisen Jin、Chi Zhang、Yuqing Jiang、Yuanfang Zhang、Yuping Wang、Ye Yuan、Guoren Wang 和 Nan Tang。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。《VLDB学报》17卷8期（2024），1925–1938。[12] Till Döhmen、Radu Geacu、Madelon Hulsebos 和 Sebastian Schelter。2024。SchemaPile：大型关系数据库模式集合。《ACM数据管理论文集》2卷3期（2024），172。[13] Yuyang Dong、Kunihiro Takeoka、Chuan Xiao 和 Masafumi Oyamada。2021。数据湖中的高效可连接表发现：一种基于高维相似性的方法。载于第37届 IEEE 国际数据工程会议（ICDE 2021），希腊哈尼亚，2021年4月19日至22日。IEEE，456–467。[14] Yuyang Dong、Chuan Xiao、Takuma Nozawa、Masafumi Enomoto 和 Masafumi Oyamada。2023。DeepJoin：利用预训练语言模型发现可连接表。《VLDB学报》16卷10期（2023），2458–2470。[15] Raul Castro Fernandez、Ziawasch Abedjan、Famien Koko、Gina Yuan、Samuel Madden 和 Michael Stonebraker。2018。Aurum：数据发现系统。载于第34届 IEEE 国际数据工程会议（ICDE 2018），法国巴黎，2018年4月16日至19日。IEEE计算机学会，1001–1012。[16] Javier Flores、Sergi Nadal 和 Oscar Romero。2021。迈向可扩展的数据发现。载于第24届国际数据库技术拓展会议（EDBT 2021）论文集，塞浦路斯尼科西亚，2021年3月23日至26日。OpenProceedings.org，433–438。[17] Prateek Jain、Praneeth Netrapalli 和 Sujay Sanghavi。2013。使用交替最小化进行低秩矩阵补全。载于计算理论研讨会（STOC'13），美国加利福尼亚州帕洛阿尔托，2013年6月1日至4日，Dan Boneh、Tim Roughgarden 和 Joan Feigenbaum（编）。ACM，665–674。[18] Lan Jiang 和 Felix Naumann。2020。整体式主键和外键检测。《智能信息系统杂志》54卷3期（2020），439–461。[19] Youri Kaminsky、Eduardo H. M. Pena 和 Felix Naumann。2023。发现相似性包含依赖。《ACM数据管理论文集》1卷1期（2023），75:1–75:24。doi:10.1145/3588929[20] Yehuda Koren、Robert M. Bell 和 Chris Volinsky。2009。推荐系统中的矩阵分解技术。《计算机》42卷8期（2009），30–37。[21] Sebastian Kruse 和 Felix Naumann。2018。近似依赖的高效发现。《VLDB学报》11卷7期（2018），759–772。[22] Jinyang Li、Binyuan Hui、Ge Qu、Jiaxi Yang、Binhua Li、Bowen Li、Bailin Wang、Bowen Qin、Ruiying Geng、Nan Huo、Xuanhe Zhou、Chenhao Ma、Guoliang Li、Kevin Chen-Chuan Chang、Fei Huang、Reynold Cheng 和 Yongbin Li。2023。大语言模型是否已能充当数据库接口？面向大规模数据库事实依据文本到 SQL 的 BIg Bench。载于《神经信息处理系统大会进展》第36卷：2023年神经信息处理系统大会年会（NeurIPS 2023），美国路易斯安那州新奥尔良，2023年12月10日至16日。[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。《VLDB学报》16卷10期（2023），2578–2590。

[24] Matéo Mahaut、Laura Aina、Paula Czarnowska、Momchil Hardalov、Thomas Müller 和 Lluís Màrquez。2024。大语言模型的事实置信度：当前估计器的可靠性和鲁棒性。载于计算语言学协会第*62*届年会论文集（第*1*卷：长篇论文），*ACL 2024*，泰国曼谷，*2024*年*8*月*11*日至*16*日。计算语言学协会，4554–4570。[25] Marc Maynou、Sergi Nadal、Raquel Panadero、Javier Flores、Oscar Romero 和 Anna Queralt。2024。FREYJA：数据湖中的高效连接发现。*CoRR* abs/2412.06637（2024）。[26] 微软。2024。*SemPy Python* 库。https://learn.microsoft.com/en-us/fabric/data-science/semantic-link-power-bi?tabs=sql 访问日期：2025-4-24。[27] Yudi Pawitan 和 Chris Holmes。2024。大语言模型推理的置信度。*CoRR* abs/2412.15296（2024）。[28] Rakesh Pimplikar 和 Sunita Sarawagi。2012。使用列关键词回答 Web 上的表查询。*Proc. VLDB Endow.* 5, 10（2012），908–919。[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。载于第*12*届 *Web* 与数据库国际研讨会，*WebDB 2009*，美国罗得岛州普罗维登斯，*2009*年*6*月*28*日。[30] Anish Das Sarma、Lujun Fang、Nitin Gupta、Alon Y. Halevy、Hongrae Lee、Fei Wu、Reynold Xin 和 Cong Yu。2012。查找相关表。载于 *ACM SIGMOD* 数据管理国际会议论文集，*SIGMOD 2012*，美国亚利桑那州斯科茨代尔，*2012*年*5*月*20*日至*24*日。ACM，817–828。[31] Meihui Zhang、Marios Hadjieleftheriou、Beng Chin Ooi、Cecilia M. Procopiuc 和 Divesh Srivastava。2010。关于多列外键发现。*Proc. VLDB Endow.* 3, 1（2010），805–814。[32] Yi Zhang 和 Zachary G. Ives。2020。在数据湖中查找相关表以开展交互式数据科学。载于*2020*年数据管理国际会议论文集，*SIGMOD Conference 2020*，线上会议 *[*美国俄勒冈州波特兰*], 2020*年*6*月*14*日至*19*日。ACM，1951–1966。[33] Erkang Zhu、Dong Deng、Fatemeh Nargesian 和 Renée J. Miller。2019。JOSIE：用于在数据湖中查找可连接表的重叠集合相似性搜索。载于*2019*年数据管理国际会议论文集，*SIGMOD Conference 2019*，荷兰阿姆斯特丹，*2019*年*6*月*30*日至*7*月*5*日。ACM，847–864。[34] Erkang Zhu、Fatemeh Nargesian、Ken Q. Pu 和 Renée J. Miller。2016。LSH Ensemble：互联网规模的域搜索。*Proc. VLDBEndow.* 9, 12（2016），1185–1196。