软件学报 ISSN 1000-9825, CODEN RUXUEW

Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

面向 AI 的数据管理技术综述*

李国良, 周煊赫, 冯建华

清华大学计算机系

通讯作者: 李国良

摘　要: 人工智能技术因其强大的学习和泛化能力已经被广泛应用到各种真实场景中。然而，现有人工智能技术还面临着三大挑战。第一，现有 AI 技术使用门槛高，依赖于 AI 从业者选择合适模型、设计合理参数、编写程序，因此很难被广泛应用到非计算机领域；第二，现有 AI 算法训练效率低，造成了大量计算资源浪费，甚至延误决策时机；第三、现有 AI 技术强依赖高质量数据，如果数据质量较低，可能造成计算结果的错误。数据库技术可以有效解决这三个难题，因此目前面向 AI 的数据管理得到了广泛关注。本文首先给出 AI 中数据管理的整体框架，然后详细综述基于声明式语言模型的 AI 系统、面向 AI 优化的计算引擎、执行引擎和面向 AI 的数据治理引擎四个方面。最后展望未来的研究方向和挑战。

关键词: 数据管理技术; 人工智能; 声明性语言;

中图法分类号: TP311

A Survey of Data Management Techniques for Supporting Artificial Intelligence

Guoliang Li, Xuanhe Zhou

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract: Artificial intelligence has been widely used in various scenarios due to its powerful learning and generalization ability. However, most of the existing AI techniques are facing three major challenges. First, existing AI techniques are hard to use for ordinary users, which depends on AI experts to select appropriate models, choose reasonable parameters and write programs, so it is difficult to be widely used in non-IT fields. Second, the training efficiency of existing AI algorithms is low, resulting in a lot of waste of computing resources, even delaying decision-making opportunities. Third, existing AI techniques are strongly dependent on high-quality data. If the data quality is low, it will make error decisions. The database technology can effectively solve these three problems, and AI-oriented data management has been widely studied. Firstly, this paper gives the overall framework of data management in AI, and then gives a detailed overview of AI-oriented declarative language model, AI-oriented optimization, AI-oriented execution engine and AI-oriented data governance. Finally, we provide the future research directions and challenges.

* 基金项目: 国家自然科学基金(61925205, 61632016)

Foundation item: National Natural Science Foundation of China (61925205, 61632016)

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

Key words: data management technology; artificial intelligence; declarative language

1 前言

1.1 研究背景

人工智能技术已经渗透到各行各业中。2019 年，中国人工智能核心产业规模接近 570 亿元，涉及安防、金融、医疗、教育等诸多领域。面向不同的应用需求，人工智能技术已经衍生出了多种不同的机器学习算法，如深度学习、主动学习、强化学习等等。然而人工智能在落地过程中还面临着三个挑战性问题：

第一，AI 使用门槛较高。人工智能算法实际的落地情况并不容乐观。在不同领域下，我们需要构建独立的人工智能系统来执行操作。如 TensorFlow 框架中，我们首先需要用 Python 语言书写程序，然后调用相应的机器学习库实际执行模型。整个程序设计和执行流程都需要专门的人力和资源投入，开销比较大。如谷歌进行大规模神经网络学习时，曾需要用有 255 台计算节点的集群单独训练具有 5.57 亿个参数的 AmoebaNet-B 模型，每个节点上配备 1024 个 TPU 组成的芯片组。此外，这类机器学习框架难以与现有数据库兼容，存在数据转换和传输的额外开销。

第二，AI 训练效率较低。首先，现有 AI 系统缺少执行优化技术（如大规模缓存、数据分块分区、索引等），不仅会导致大量的计算、存储资源浪费，而且会提高程序异常的发生率（如内存溢出、进程阻塞等），严重影响单个任务的执行效率。其次，传统 AI 的执行方式缺少灵活性，芯片（如 CPU、ARM、GPU 等）、算法优化（如优化函数、评价指标等）都需要人为指定，不仅提高了对使用人员编程能力的要求，而且难以最大限度的发挥硬件资源优势，如没有动态调度机制，很多 AI 任务要被阻塞直到 GPU 资源满足要求为止，降低了整体的执行效率。

第三，AI 依赖高质量的训练数据。传统的数据治理工作非常繁杂，需要大量人的参与，并消耗大部分的资源和时间。首先，大规模机器学习算法需要大量的数据进行训练，一方面，来自真实场景的原始数据多不能直接使用，存在大量的缺失值、错误值和异常样本等；另一方面，一个训练集可能有多个数据源，数据源融合存在格式不一致、冗余信息多、连接开销大等问题。以图 1 为例，我们可以看出，现有机器学习的整个数据处理流程有很大的优化空间。

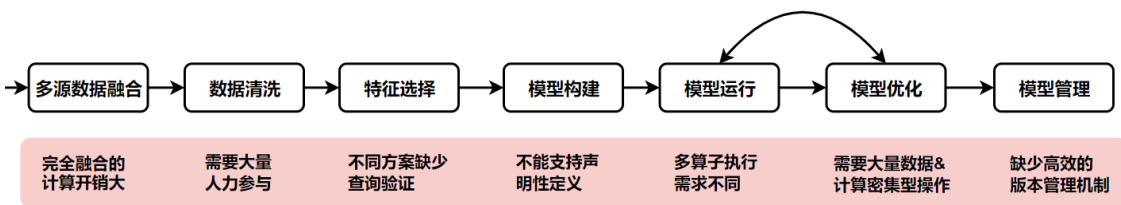

图 1 机器学习的生命周期及存在的问题

数据库管理系统经过近 60 年的发展，积累了很多较为成熟的数据查询和管理技术。以一条查询语句的处理流程为例。首先，基于声明式语言，用户只需要在查询语句中声明高层次条件（如数据列表，属性约束等）。

其次，数据库经过逻辑和物理优化生成执行计划，利用索引、内存计算、分布式处理等技术高效的执行查询处理操作。此外，关系型数据库基于关系表对数据集进行建模，利用外键等保证数据一致性。通过结合这些数据管理技术，我们可以有效解决以上三个难题：其一，声明式语言（类 SQL）可以降低 AI 使用门槛；其二，数据库优化技术（如索引、计划选择、视图缓存等）可以提升训练速度，降低资源使用率；其三，数据治理技术可以提升数据质量，提升 AI 训练质量。因此面向 AI 的数据库技术得到了广泛关注。

论文从四个层次介绍数据管理技术如何支持和优化人工智能技术（如图 2 所示）。

第一层，用声明性的语言模型简化 AI 的使用。其一，我们讨论如何用声明性的 SQL 语言代替传统 AI 高级语言，降低 AI 的使用门槛。其二，我们调研面向 AI 的 SQL 完备性相关的工作，分析如何细化 SQL 支持 AI 的粒度。其三，我们调研面向 AI 的 SQL 的智能推荐技术，即对 SQL 层进一步做逻辑封装，利用可视化等技术进一步提高 AI 的易用性（第二章）。

第二层，用算法优化引擎优化 AI 的执行逻辑。其一，我们调研面向 AI 的优化引擎，分析如何支持 AI 算法中不同的算子类型。其二，我们分析 AI 算子的代价估计技术，为 AI 算法选择提供基本资料。其三，基于声明性的语言模型，同一个 AI 问题的描述（如类 SQL 语句）可以被解析成多种不同类型的 AI 算法。因此这里我们调研 AI 算法的自动生成技术，分析如何根据 AI 问题选择、组装基本的 AI 算子，生成高效的执行逻辑。其四，我们调研 AI 模型的版本管理技术，利用数据库在管理时间序列数据上的经验，帮助数据分析师高效的组织历史训练结果（第三章）。

第三层，用异构执行引擎提高 AI 的物理执行效率。在第二层中我们调研了从逻辑层面对 AI 算子进行建模的方法，然而在物理层面（如芯片、加速器、单机架构）数据库技术仍然存在着很多局限性。其一，我们调研异构的 AI 计算方法，分析数据管理技术在硬件优化、优化函数优化、执行逻辑优化三个方面的工作。其二，我们调研数据库如何基于分布式加速架构（如 Hadoop、Spark 等）计算 AI，提供大规模的并行计算能力。其三，我们分别从数据库内核、AI 芯片、新型数据库架构三个方面调研 AI 执行的优化技术（第四章）。

第四层，用智能数据治理引擎优化数据的管理效率。因为 AI 算法严重依赖于数据，数据库在高效支持 AI 计算的同时，还需要解决 AI 数据收集、融合、清洗等多个方面的问题。其一，真实数据中经常存在大量的错误和不一致性，我们调研面向 AI 的数据发现技术。其二，繁杂的数据清洗工作需要消耗大量的人力，我们调研面向 AI 的数据清洗工作。其三，多源异构数据的融合开销比较大，我们调研面向 AI 的数据融合技术（第五章）。

最后，我们对未来基于数据库的人工智能优化技术的发展方向进行展望（第六章）。

不同于已有综述文献[1,2,3]主要关注人工智能在传统数据库技术上的应用，本文创新性的从语言模型、逻辑执行优化、物理执行优化、数据管理四个方面综述数据库技术如何优化人工智能的使用和执行效率，并给出未来人工智能技术与数据库内核结合的挑战和机遇。

图 2 面向人工智能的数据管理技术

2 声明性的语言模型

机器学习繁杂的编程语言已经成为普惠 AI 的重要障碍。相同的机器学习算法有多种不同的实现版本，用户如果没有足够的编程经验很难快速理解、使用、修改。而 SQL 作为高级编程语言，只需要声明输入数据之间的关系和需要的数据，可以方便的为用户使用。因此，如果可以为用户提供基于 SQL-like 接口调用机器学习模型的方法，用户使用 AI 技术的门槛便会大大降低。本章中，我们分别从三个方面分析如何更好地建立声明性的语言模型：1）高级语言-to-SQL，实现用类 SQL 语言声明机器学习算法；2）AI-SQL 的完备性，根据机器学习算法不同阶段的特点合理设计 SQL 语法，提高语言模型的扩容能力和执行效率；3）声明性的智能分析服务，在 SQL 上进一步封装操作，简化模型定义的同时提升对不同计算引擎的支持能力。

2.1 面向AI的高级语言-to-SQL

传统机器学习算法多基于 Python、R 等命令式的编程语言，需要程序员定义好完整的执行逻辑。用声明性语言 SQL 来简化人工智能技术使用的同时，也面临着一定的难度和挑战。首先，目前人工智能和数据查询有不同的执行引擎，我们需要设计一个统一的 SQL 解析引擎来兼容不同的执行平台，并提供跨计算引擎优化能力。其次，人工智能程序需要完整地给出算法逻辑，我们需要通过逻辑抽象简化人工智能技术的使用方法。

最直接的方法是将 SQL 翻译成不同语言的脚本，再分发到不同平台上执行。如 BigQuery ML[4]通过在 SQL 语法中直接增加“CREAT MODEL”字段，用于定义常见的机器学习模型（如线性回归、逻辑回归等）。然后对于带有“CREATE MODEL”字段的查询，BigQuery 直接将类 SQL 语句转换为等价的 TensorFlow 脚本执行。局限于传统 SQL 语法的表示能力（如张量操作通过嵌入 Python 脚本实现），BigQuery ML 目前能够支持的机

器学习算法有限（如线性回归、二元逻辑回归、多项式逻辑回归等）。

针对上述问题，SQLFlow[1]在 SQL 解析引擎（如 MySQL、Hive、SparkSQL 等）和人工智能的算法库（如 TensorFlow、Keras 等）之间建立一套灵活、开放的语言翻译机制。如图 3 所示，其一，他们主要使用 SELECT 语句，通过扩展选项关键词定义 TensorFlow 估计器（“TRAIN DNN”），使系统与 SQL 引擎之间保持松耦合，避免增加如“CREATE MODEL”这样的一级关键词。其二，他们提供更加开放的模型定义方式，允许用户设置估算器属性（如 Python 类的构造函数的参数；模型训练、评估、预测的参数等）。用户可以选择使用默认值，或者将超参数估计的研究成果集成到系统中，以自动优化参数的默认值。其三，SQLFlow 翻译机制（SQL -> SQL Parser -> extended SQL -> code generator -> execution engine）默认用从 tf.estimator.estimator 派生出来的估计器估计不同张量流图的开销，从而自动生成较优的数据流图。SQLFlow 并不限制底层机器学习工具包必须是张量流，只需要估算器提供训练、评估和预测的方法。这种方式下，SQLFlow 就不需要关心它是否调用 TensorFlow[5]或 xgboost[6]框架，从而更好的兼容不同的机器学习框架。

```
SELECT  * FROM kaggle_credit_fraud_training_data
LIMIT 1000
TRAIN DNNClassifier                 -- a pre-defined estimator, tf.estimator.DNNClassifier
WITH layers=[50, 100]               -- a parameter of the Estimator class constructor
     train.batch_size = 20          -- a parameter of the Estimator.train method
COLUMN *,
     cross(v1, v9, v28)             -- plus a derived (crossed) column
LABEL class
INTO sqlflow_models.my_model_table; -- save trained model parameters and features into a table
```

图 3 在 SQLFlow 上定义一个 DNNClassifier 分类器

2.2 面向AI的SQL的完备性

面向 AI 的 SQL 的完备性主要包括两方面的内容。一方面， SQL 语言模型需要有独立的描述 AI 问题的能力，不需要用户额外嵌入其他语言脚本。另一方面，支持全过程的 AI 操作，即基于 SQL 语句就可以完成所有的数据和模型操作，不需要在底层转换为其他语言再执行。

MADlib 系统[7]将 AI 算子嵌入到数据库系统中，提供完备的 SQL 翻译机制。MADlib 核心是将常用的机器学习库兼容到数据库系统（如 PostgreSQL）中。一方面，用户可以完全靠 SQL 语句调用机器学习算法，包括数据的获取、连接、抽样和模型的定义、训练、调用等。另一方面，由于传统数据库一般不支持在矩阵上进行大量的线性变化操作，MADlib 针对性的提供面向 AI 的数据操作和训练模式：

- **面向 AI 的数据操作**：在 SQL 级别上，MADlib 智能地将矩阵划分成块，将这些块加载到各个节点的内存空间。一旦进行了分区，就可以使用 SQL 来协调这些块在一台或多台机器上的内存进出。传统查询优化器或数据库设计工具不具有智能划分矩阵模式，因此 MADlib 通过在 SQL 语法中增加数据划分功能，使数据库引擎可以精心安排分区的数据移动和各阶段的中间结果。此外，MADlib 提供新的用户

1 https://sql-machine-learning.github.io/

自定义聚合函数。由于数据库系统主要使用聚合函数处理并行的流式数据，基于聚合函数定义的机器学习算法可以更高效的被执行。

- **训练模式**：为了更好地支持不同机器学习算法的训练模式，MADlib 通过迭代训练对模型进行优化，引入如梯度下降、马尔可夫链、蒙特卡洛等优化算法：1）通过虚拟表计算迭代次数。声明一个具有 n 行的虚拟表（例如，通过 PostgreSQL 的 generate_series_table 函数），并将其与表示单个迭代的视图连接起来；2）递归查询。使用 SQL 的递归特性来执行任意停止条件下的迭代计算；3）驱动函数。上述方法不具备通用性和可移植性。因此，在 MADlib 中，我们通过在 Python 中编写驱动程序 UDF 来实现更复杂的迭代方法，从而控制迭代和迭代传递状态。类似工作还有 SAP PAL[8]、MLlib[9]、MLog[10]等。

2.3 面向AI的SQL智能推荐

对于 SQL 语言本身，其核心“SELECT…FROM…WHERE…GROUP BY…”是一种非常强大的工具，可以支持跨一个或多个处理器和磁盘协调数据的批量处理。但是使用 SQL 仍然有一定的学习和使用成本，而且对于普通用户来说仍然不够直观。所以另一类研究进一步将 SQL 封装成动态图、电子表格等，利用可视化技术提供更加智能的推荐服务。

- **AI 工作流的可视化推荐**：对于人工智能来说，数据工作流的主要任务是让机器对已有数据进行发现、清洗、融合和分析，从而使机器能对新的数据进行合理的判断。数据工作流设计是一个复杂的过程，涉及诸多数据工程技术，即使使用 SQL 语句也需要很高的专业经验和时间投入。因此，通过将数据工作流的设计过程可视化，我们可以进一步简化 AI 的使用。代表性工作如 BigQuery ML。首先，**数据融合控制中心**允许用户将数据工作流拖放到控制中心统一管理，无需编码就可以基于可视化界面浏览和管理所有数据集和数据工作流。Telus Digital 的研发架构中，BigQuery ML 提供：1）预置的连接器，用于同步不同的数据源（如用户网上行为、用户交易数据、经济社会环境数据等）；2）数据转换集合（如数据收集技术、数据转换技术等），作为生成数据工作流的基础工具库，方便用户快速的获取结果。此外，通过提供直观的可视界面和工作流抽象，相比于 SQL 语言，进一步降低了数据融合的门槛。这种增强的可访问性，它允许数据分析师和科学家“自助服务”，而无需云或软件工程专家的帮助。其次，**数据分析**方面，面向谷歌云上 EB 级的智能数据分析。每天谷歌的用户都会上传 PB 级的数据到 BigQuery 上。因此 BigQuery 需要在 EB 规模的、无服务器模式的数据仓库上提供智能分析服务。BigQuery ML 采用一种简单的方法对海量数据进行机器学习分析，分为 Cloud Dataflow SQL 和 Dataflow FlexRS 两部分。Cloud Dataflow SQL 用于启动数据流水线和调度作业，还可以自动检测批处理或流数据处理的需要：1）Cloud Dataflow SQL 使用的 SQL 语法与 BigQuery 中使用的相同。这允许数据分析师使用 BigQuery UI 中的 Dataflow SQL，将云发布/子流与数据基础结构中的文件或表连接起来，然后直接实时查询合并的数据。这意味着用户可以生成实时查询分析并创建“仪表板”以可视化结果；2）Dataflow FlexRS 则通过灵活的调度，批处理要执行的作业。它可以根据用户优先级、需求程度动态的调度任务，如若用户正在处理非时间敏感的数据，它的执行优先级会被自动后调，但是能从可抢占的资源定价中获益。此外，FlexRS 考虑了成本效益，允许隔夜执行作业。

- **交互式的 SQL 生成技术**：大量的业务用户将电子表格作为数据分析不可或缺的工具。因此 BigQuery ML 提供一种虚拟电子表格技术 Connected Sheets。一方面，它结合了电子表格界面的简单性和机器学习算法的拟合能力；另一方面，Connected Sheets 没有行限制，可以使用来自 BigQuery 的完整数据集。不管是数百万还是数十亿行的数据都能以一种虚拟电子表格的方式呈现给用户。用户在表格上的进行的数据操作都会被自动转化成 SQL 语句发送到数据库执行，并基于分布式处理技术优化查询效率。它的优势包括：1）方便使用：用常规工作表功能（包括公式、数据透视表和图表）即可进行分析，并将结果数据可视化为工作表呈现给用户。如亚洲航空公司（AirAsia）就用这套方法实现数据平民化，即分析师和业务用户能够直接创建图表，并利用他们在海量数据集上的现有技能（如审计、数据统计等方法）访问 BigQuery 中的基础数据，并且可以实现最细粒度数据（列/行级）的访问，而完全不需要调用 SQL 语句；2）安全：基于可信数据管道，允许用户与组织中的任何人安全的共享电子表格上的数据，进行实时的多人在线分析。

3 AI 算法优化引擎

传统数据管理系统中，优化引擎根据查询树生成实际使用的执行计划。随着声明性 AI 语言模型的快速发展，同一个 AI 问题（如图片分类、人脸识别等）也可能会有多个算法、参数的组合供选择，而且不同组合对执行性能和表现可能有很大的影响。因此本章中我们调研数据库优化引擎中的技术，分别从面向 AI 的优化引擎、AI 算子的代价估计、AI 算法的自动生成、AI 模型的版本管理四个方面进行综述和分析。

3.1 面向AI的优化引擎

面向 AI 的优化引擎主要指根据查询树生成统一的执行计划，对于异构的计算平台，根据计划中的算子类型下发到指定的计算平台上分别执行；否则，由数据库计算引擎直接执行。下面我们分别综述这两类技术。

- **异构的计算平台**：前面提到的机器学习工具 SQLFlow 的代码生成器（Code Generator）负责执行计划的生成和转发。首先，代码生成器根据解析结果（SQL 关键字匹配）判断该 SQL 是 AI 操作还是数据库查询。如果是 AI 操作，代码生成器先通过 ODBC 驱动程序将查询中的标准 SELECT 部分传递给 MySQL，获取需要的数据；然后，它进入一个循环，反复从 SELECT 语句的运行中读取输出结果，如果是训练阶段，则用于优化模型，否则用经过训练的模型进行预测。此外，在每个训练阶段，SQLFlow 都要依赖 TensorFlow 算法包来更新 TRAIN 子句中指定模型的参数。如果是数据库查询，则直接下发到优化器继续执行。SQLFlow 的核心优势包括：1）基于 SQL 降低机器学习算法的使用门槛；2）一条扩展的 SQLFlow SQL 就可以包含数据查询和机器学习操作，并行执行可以提高处理效率。然而，缺点是 1）数据传输开销没有改善，数据和模型计算仍然集成在不同的平台上；2）半自动化的问题声明，仍然需要用户定义大部分的执行操作，代码生成器对于执行算法的优化空间较小。

- **统一的数据库计算引擎**：MADlib 根据机器学习算法流程做针对性的计划优化。以 k-means 的一个实现为例[11]。MADlib 主要使用驱动函数，迭代的调用用户定义的聚合函数（User Defined Aggregate, UDA）执行操作。UDA 中有两个状态：迭代间状态代表 UDA 最终函数的输出，迭代内状态代表由 UDA 的转换和合并函数维护的状态。在聚合期间，转换状态包含两个迭代内状态，但只修改迭代内状态。第

一步，它存储 k 个分类中心在两种状态中，并将训练点到各个分类中心的赋值视为默认给出的。第二步，在迁移函数中，它首先计算出当前点与迭代状态开始运算时最接近的分类中心。为了简化运算，首先，它需要计算上一次迭代中最接近的分类中心，然后计算当前迭代中最接近的中心。如果它显式地存储最近的点，我们可以避免最近质心计算的一半。然后在迭代内部状态更新分类中心。只有作为聚合的最后一步，迭代内状态才成为新的迭代间状态。第三步，将点存储在一个关系表中，该表包含该点的坐标属性和当前最近的分类中心。迭代状态将每次所有的分类中心的位置存储在一个矩阵中。MADlib 提供了一个 UDF：close_column(a, b)，表示在 a 矩阵中距离向量 b 最近的向量，用 SQL 语句表示为：“UPDATE points SET centroid_id = closest_column(centroids, coords)”。第四步，数据库会逐个处理查询（并且不执行跨语句优化），因此每 k 均值迭代需要对数据进行两次传递，直到确定一个样本的分类中心。

我们可以看到，MADlib 完全将人工智能算法的执行逻辑内嵌到数据库中，提供运行时优化。此外，计算和数据传递都在同一个数据库系统中进行，也进一步提升了执行效率。但是，缺点是逻辑优化比较复杂，不方便迁移到新的数据库系统上。

3.2 AI算子的代价估计

AI 算法中存在大量不同类型的算子，如标量、向量、大规模张量等操作。面对声明性的 AI 问题，如果有多个实际的 AI 算法可以解决，那么评估这些 AI 算法的执行代价对于提高解决问题的效率就有很大的意义。目前这方面的工作比较少。如在 Kumar 等人提出的基于线性回归模型优化数据源的连接操作[12]中，他们主要研究用于连接操作的混合哈希算法，快速估计所有方法的 I/O 和 CPU 成本。这种成本估计的思想很简单。如当关系表 R 的散列表可以容纳在主内存中时，它分别估计基于线性回归模型的索引连接、传统排序合并算法的执行成本：

$$\text{Hybrid: } \begin{cases} \{R\} * [hash + move] + \\ \{S\} * [hash + comp * F]. \end{cases} \tag{1}$$

$$\text{Sort: } \begin{cases} \{R\} * [comp + (log_2\{R\}) * (comp + swap)] + \\ \{S\} * [comp + (log_2\{S\}) * (comp + swap)]. \end{cases} \tag{2}$$

其中，R 和 S 是要进行连接操作的两张关系表；hash 指在内存中给一个属性建立索引的时间；move 指将一行数据拉入内存的时间；comp 指在内存里比对两列属性的时间；swap 是将以行数据换出内存的时间；F 是增量系数。因为当 R 的散列表可以容纳在实际内存中时没有 I/O 成本，索引很明显混合哈希将优于排序合并算法。进一步扩展，由于任何 AI 操作主要包括数据搜索、运算等基本操作。我们可以先用计算开销（CPU 成本）和数据移动开销（I/O 成本）估计基本操作的代价，再综合起来对不同的 AI 操作进行评估。但是这种基于经验公式的方式局限性很大，未来需要有普适性更强的模型来代替这套方法。

3.3 AI算法的自动生成

基于需求自定义 AI 算法是一件麻烦的事情。通过向用户暴露机器学习模型的数学结构，TensorFlow、Theano 和 Caffe 等机器学习框架可以构造出各种灵活而且具有表现力的机器学习模型。然而，一方面，用户需要掌握大量的数学、算法方面的知识才能进行程序书写；另一方面，由于没有声明性的数据管理层，用户需要处理

这些系统中的数据加载、移动和批处理等繁琐且往往容易出错的任务。因此人们推出了 Keras、PyTorch 等更高层的机器学习库，用户只需要声明性的给出数据之间的逻辑关系，而不再需要具体解释为了得到最终模型需要做哪些数据操作，如 Json 格式转换、张量计算操作等。然而，他们只解决了前一个问题。他们仍然不能为像 numpy 数组这些数据提供智能管理，用户仍然需要处理低层编程细节，例如需要考虑这些数组是否适合内存，需要自定义经典数据库功能，像跨进程重用计算、基于时间戳的数据快照管理等等。此外，Keras 不能集成到承载大多数企业数据的标准关系数据库生态系统中。因此，我们综述数据管理技术在 AI 算法自动选择和组装中的应用。

首先，基于数据库的关系代数自动生成人工智能算法。在 Mlog[10]中，他们用数据库系统管理所有的数据移动、数据一致性、数据持久化和机器学习相关的优化方法（如 SGD，Adam 等）。其一，他们基于传统数据库（如 SciDB）构建标准数据模型，避免重复数据库系统在数据管理上进行的工作，包括构造 relational table schema，建立索引、外键等。因为传统数据库缺少针对大规模张量数据的索引结构，Zhang 等人[10]提出了一种新型的数据结构（Linear Array B-tree， LAB-tree）。他们主要利用线性函数，针对不同维度之间的关系建立 B-tree，然后将 B-tree 映射到一维向量上，再对这个一维向量进行索引，从而在索引中同时体现了数据结构和数据特征；其二，Mlog 通过扩展查询语法支持用户利用传统关系型视图和关系型查询来声明机器学习模型。首先，他们提供一种基于张量视图的新型查询语言，它具有与现有的关系型数据模型和查询语言兼容的形式语义。1）数据模型：Mlog 的数据模型是基于张量的，所有操作都是张量上线性代数的子集。在 Mlog 中，张量与关系模型密切相关；实际上，从逻辑上讲，张量被定义为一种特殊的关系类型。设 t 为维数 dim（T）的张量，每个维数 j 的索引范围为{1，…，dom（T，j）}。逻辑上，每个张量 T 对应一个关系 R；2）关系代数：我们在张量上定义了一个简单的关系代数，并就关系 R 定义了它的语义，这使得我们能够将张量上的操作紧密集成到关系数据库中，并生成统一的语义。这个关系代数与支持线性代数操作的扩展的数据立方体非常相似。比如，对于切片算子，用于选择输入张量的子集，包括不同的维度区域等；对于线性代数算子，它支持一系列线性代数运算符，包括矩阵乘法和卷积。这些操作符都有 op（T_1，T_2）的形式等。这套关系代数将 SQL 转换成 Mlog 程序，然后他们使用文本静态分析技术自动将 Mlog 程序编译成本机 TensorFlow 程序，生成实际的执行逻辑。这类工作还有 DataRobot[1]、Zylotech[2]和谷歌的 AutoML 等等。他们不仅能够基于用户需求自动生成或重用机器学习模型，而且能够自动匹配合适的优化方法。

其次，基于内置的 AI 算子对人工智能算法进行选择。LogicBlox 提供了类 SQL 语言 LogiQL[14]，基于对传统 SQL 语句封装更好的支持机器学习等数据分析技术。下面我们分别以规定性分析和预测分析为例，分析 LogicBlox 如何利用 LogiQL 生成机器学习算法：

- **规范性分析**（Prescriptive Analysis）：通过添加一个语言特性可以将 LogiQL 扩展支持数学优化和规范性分析。其思想是，谓词 R[x_1，…，x_n]=y 可以声明为自由二阶变量，这意味着系统负责用元组填充它，从而满足完整性约束。此外，可以将 R[*]=y 形式的派生谓词声明为最小化或最大化的目标函数。假设我们希望自动计算股票金额，以实现利润最大化，将以下内容添加到程序中就足够了：

1 https://www.datarobot.com/

2 https://www.zylotech.com/

“lang:solve:variable(‘Stock)；lang:solve:max(‘totalProfit).”。其中，第一句是一个二阶存在量词的简写，它指出股票的谓词 Stock 应该被当作我们正在解的一个自由二阶变量，而第二句则指出谓词总利润 totalProfit 是一个需要最大化的目标函数（服从完整性约束）。在算法优化引擎中，程序被转换成线性规划（LP）问题，并传递给适当的解算器[15]。LogicBlox 通过自动合成另一个 logiQL 程序（将变量谓词上的约束转换为可由求解器使用的表示），以类似于马尔科夫链[16,17]的方式将问题实例中的量词排除。此外，它利用 LogicBlox 系统中的所有查询评估机制（例如查询优化、查询并行化等），提高了模型落地的可扩展性。系统调用适当的解算器，并用结果填充存在量化谓词的值（将未知值转换为已知值）。此外，逻辑增量地保持对解算器的输入，使系统可以增量地（重新）解决受输入更改影响的问题部分。如果更改示例应用程序，使股票谓词现在定义为从产品到整数的映射，则 LogicBlox 将检测更改并重新编写问题，以便调用不同的解算器（支持混合整数编程（MIP））。

- **预测分析（Predictive Analysis）**：预测分析主要指从已有数据中对未来事件进行预测。LogicBlox 中的预测分析最初是通过一组**内置的机器学习算法**来支持的。这是通过特殊的“预测型”P2P 规则实现的。该规则有两种模式：学习模式（在学习模型的地方）和评估模式（在应用模型进行预测的地方）。假设我们希望预测分公司产品的月销售额。我们有一个谓词 sales[sku，store，wk]=amount，以及一个谓词 feature[sku，store，feature name]=value，它与每个 sku，store 和 feature name 关联一个对应的 feature 值。学习规则（如：SM[sku, store] = m←predict << m = logist(v|f) >> Sales[sku, store, wk] = v, Feature[sku, store, n] = f）为每个 sku 及其分支学习一个逻辑回归模型，并将生成的模型对象（它是模型表示的句柄）存储在谓词 sm[sku，store]=model 中。

此外，面对快速变化的人工智能技术，LogicBlox 还支持实时编程。在这种情况下，传统的编辑-编译-运行周期被放弃，取而代之是对程序运行时行为进行实时反馈，提供更具交互性的用户体验[18]。例如，在零售计划应用程序中，我们允许用户动态定义和更改模式和公式。这些更改会触发对数据库服务器中应用程序代码的更新，而挑战是需要快速的更新用户视图以同步这些更改。

3.4 AI模型的版本管理技术

构建机器学习模型是一个迭代的过程（时间序列）。数据分析师在达到既定的表现标准前，可能要设计大量的模型进行尝试（如，线下覆盖面积，准确度等），而这些模型和相关数据在构建新的模型方面有很大的参考价值。然而，当前模型的版本管理依赖于用户手动组织，浪费人力和学习资源。因此我们调研基于数据库的 AI 模型的版本管理技术。

Vartak 等人提出了一套管理机器学习模型的端到端系统 ModelDB[19]。建立机器学习模型是一个迭代的过程。一个数据科学家在达到某些验收标准（例如 AUC 截止曲线）之前需要建立数十到数百个模型。然而，当前的模型构建方式是比较随意的，数据科学家缺少科学的工具来管理随时间构建的模型。这导致了三个问题：1）想要复现历史版本的模型或结果非常耗时，有时难以实现；2）要求数据科学家“记住”模型先前版本的结果和参数。例如，数据科学家必须记住已测试的参数或特征的组合及其结果，以便得出见解并通知下一个实验；3）数据科学家很难回答有关不同版本的具体差别。例如，数据科学家经常在某一点上发现代码或数据中的差异，然后必须重新运行代码或数据下游的所有分析。然而，在缺乏模型版本控制的情况下，识别实验非常困难。为了解决以上问题，ModelDB 基于数据库系统，在训练模型时接收模型和相关的元数据，以结构化的格

式存储模型数据，并自动跟踪本机环境中构建的机器学习模型，智能地为其编制索引，允许通过 SQL 和可视化界面灵活地浏览模型。这样 ModelDB 可以帮助数据科学家高效的进行历史版本管理，提高模型构建和重现的效率。

另一种方法则是基于链式结构进行版本控制。SimSQL[17]是一个基于 SQL 语法规范的数据库管理系统，它提供大规模分布式的模拟数据库值（整个数据库数据的抽象）的马尔科夫链，链的值在任何时间节点上都包含整个数据库的内容。SimSQL 采用了许多最初在蒙特卡洛数据库系统（MCDB）[20]中提出的想法，允许用户定义除普通数据库表之外的随机数据库表。在随机表中，有些条目是具有相关概率分布的随机变量（不一定以闭合形式已知），而其他条目则是普通的、确定性的数据值。从概念上讲，当发出包含随机表的 SQL 查询时，SimSQL 使用伪随机数生成技术来实例化每个随机数据值。查询可以在生成的数据库实例上运行；此过程以蒙特卡洛方式重复多次，以获得查询结果的经验分布。SimSQL 通过使用“元组包”方法共享公共成本，同时以高效的方式实例化和处理许多可能的数据库实例。

4 AI 异构执行引擎

上一节中，我们主要从逻辑层面介绍了数据库优化技术在 AI 领域的应用。而在物理层面，为了原生支持 AI 计算，我们仍然面临着一些挑战。传统数据库执行引擎用于实际的对指定数据进行处理，目前已经可以实现行级并行处理。但是传统执行引擎支持的算子运算比较单一，如基本的 Aggregate、Hash Join 等数据库算子。所以，为了用数据库执行引擎原生的支持 AI 操作，我们分别从算子实现（异构的 AI 计算引擎）、并行优化（分布式加速架构）、执行优化（AI 执行优化技术）、智能数据分析四个方面综述执行引擎对 AI 计算的支持。

4.1 异构AI计算引擎

本节中，我们分别介绍如何基于数据管理技术更好的支持人工智能算法中异构的操作算子。AI 算子主要有两类特点：1）种类多样，包括标量、向量、大规模张量等不同操作；2）计算密集型，大规模的张量计算对芯片（算力）和存储（数据）的性能要求很高。针对以上特点，我们分别从调研数据管理技术在硬件优化、优化函数优化、执行逻辑优化三个方面的工作。

首先，一些数据库系统集成了异构的计算框架和大规模内存计算能力，可以有效提高处理人工智能算法的能力。如 ColumnML[21]系统可以在执行 ML 任务之前方便地对复杂模式进行预处理和非规范化。首先，ColumnML 集成专门的硬件加速器，用 FPGA，GPU 等新硬件对常用子运算符（如哈希、分区、排序、高级分析等）专门化处理，从而提高 DBMS 中的处理效率。其次，在大多数这些系统中，将数据移动到加速器需要高效的 I/O；否则，性能优势将不复存在。所以 ColumnML 基于内存型列式存储，面向列的处理使从内存中移动数据更容易，同时仍然能够利用硬件中的可用并行性，特别是在压缩列时，可以提高 I/O 效率。最后，这种列式存储的方式不完全适用于机器学习算法，特别像随机梯度下降（SGD）等优化函数，它们经常同时访问一个元组的所有属性；逐行处理元组，存在大量的“行操作”。但是，他们仍然可以在列存储上执行 SGD：由于 DRAM 的行缓冲区位置的原因[22]，为了使从内存中的远程地址读取更有效，必须批量读取：从一列读取多个缓存线，将它们存储在缓存中，从下一列读取。这实际上是一种动态的列存储到行存储转换，这是不利的。所以它需要与数据集中的功能数量成比例的缓存空间和较大的批处理大小，以便尽可能按顺序读取[23,24]。

为了解决 ColumnML 等列式存储系统中行式计算低效的问题，Shalev Shwartz 等人[25]引入了一种新的优化函数，随机坐标下降算法（Stochastic Coordinate Descent, SCD），并在运行时提供一个边界以进行聚合。SCD 的核心思想是保持一个包含模型与样本之间内积结果的向量。然后可以始终将所做的更改应用于模型的一个坐标，也可以应用于该内积向量，这将在所有样本和对应的优化梯度之间保持最新的内部积。该模型要求对所有样本只访问一个坐标，这等于按列访问模式。我们通过在每次迭代中随机（不替换）选择一个特性来执行 SCD。模型 x 初始化为 0，因此内积向量 z 也从 0 开始。一个 epoch 对应于处理整个数据集：完全访问每一列以计算 Lasso[23]逻辑回归的部分渐变。然后，利用局部梯度更新模型的相应坐标。最后，将进行内积向量更新，以使 z 中的内积保持最新，通过对损失函数的估计，对优化问题进行多个阶段的求解，直至收敛。此外，他们还使用了 SCD 的分区版本（(Partitioned Version of SCD)），灵感来自于 Jaggi 等人引入的 Cocoa 算法[27]。Cocoa 的目标是在执行分布式双坐标提升时降低通信频率。PSCD 的主要目标是减少中间状态的保留量。由于缓存位置的存在，可以降低内存访问的复杂性。

此外，数据管理技术可以提供细粒度的数据操作优化。前面我们介绍了 MADlib 是如何在逻辑层面上实现 AI 算法自动化生成的，而在物理层面上，一方面，MADlib 的数据库引擎可以利用线性代数例程来处理其获取的核心数据片段。为此，它需要快速地调用经过优化的线性代数方法。除了执行块的粗粒度管理之外，数据库引擎还必须有效地调用执行算术计算的单个节点上的代码。对于在行级操作的 UDF（可能每行多次），标准方法是在 C 或 C++中实现它们。当计算密集矩阵运算时，这些函数会对开放源代码库进行本地调用，如 Lapack[27]或 Eigen[28]。另一方面，稀疏矩阵在标准数学库中没有很好地处理，需要进行更多的定制，以便在磁盘和内存中有效地表示。他们选择用 C 语言为 MADlib 编写自己的稀疏矩阵库，利用 C 语言中的基本算术循环进行编码。

4.2 分布式AI加速架构

传统单机上的 AI 优化方法已经不能满足大规模机器学习的需求。分布式计算引擎可以从两个方面进一步优化机器学习的执行效率。首先，通过水平扩展，我们可以将机器学习的训练任务下发到多个物理节点上，利用物理并行方式加速 AI 算法的执行效率。其次，由于很多机器学习的优化算法需要频繁读写数据，如批量随机下降算法每轮训练需要计算所有样本的损失值，我们通过数据压缩，将尽可能多的数据放到内存中进行计算，从而进一步提升整体执行效率。因此下面我们分别从分布式任务并行和分布式内存计算两个方面综述这类技术。

- **分布式并行计算**：单机数据库的性能面临着瓶颈，越来越多的数据库系统支持分布式架构，如 MPP、NUMA，解决临界资源带来的瓶颈问题，为 AI 技术提供**大规模的并行数据处理**功能。同时，很多研究考虑如何利用分布式数据库系统加速机器学习算法，如 Kinetica[29]基于 MPP 架构支持大规模的机器学习训练。首先，它在分布式节点上进行大规模的统计和分析查询；其次，在这个规模下，机器学习模型可以同时基于原始数据的多个子集进行训练。对每个子集，模型学习到一种状态之后就将这部分数据丢弃在数据仓库中（节省空间）；最后，基于自定义函数（CDF），用户可以定义模型的训练方式。另一种方法是重新定义分布式机器学习库，如 MLlib 基于 Spark 的分布式执行框架执行机器学习算法。首先，它提供一套开源的分布式机器学习库，算法包括朴素贝叶斯、决策树、k-Means 等[30]。该库还

提供了许多用于凸优化、分布式线性代数、统计分析和特征提取的低级原语和基本实用程序，并支持各种 I/O 格式，包括对 libsvm 格式的本机支持、通过 spark-SQL 进行数据集成用于模型导出的 MLlib 的内部格式。在最底层，Spark Core 提供了一个通用的执行引擎，有超过 80 个算子来转换数据，例如进行数据清理和特征化。相似工作还有 RHEEM[31]，从更细粒度层次支持分布式的执行算子。为了实现无缝的分布式执行，RHEEM 的逻辑层算子（UDF 中）经过优化（任务并行化、部署细节）转换成实际执行算子，再分发到各个节点上执行。它将数据单位细化到行/列级，允许我们在更高的并行度上设计操作并获得更好的表现。

- **分布式内存计算**：分布式内存计算是将尽可能多的数据经过压缩存储在主内存中，通过避免频繁的 I/O 操作**加速 AI 计算效率**。在分析和商业智能数据库（Business Intelligent DB）中广泛的采用这种模式来**加速数据分析的效率**。如 MLWeaving 进一步提供了一种紧凑的内存表示，它通过将具有数据拆分到比特级别，对具有指定值的不同的比特位在大规模内存上独立存储，从而方便灵活的进行并行数据处理。然后数据分析平台 Kx 通过结合基于内存的时间序列数据库 kdb+，提高数据处理效率的同时减少资源开销。首先，Kx 具有管理、吸收、存储和分析大型数据集的能力，是深度神经网络的理想执行引擎。其次，基于内存中的流分析功能，适合快速进行数据集的采样、聚合和连接操作。

4.3 AI执行优化技术

在当今数据中心里，机器学习已经成为一种重要负载，如广告定位、内容推荐等。如上一节介绍的，目前已经有很多支持人工智能技术的平台。但是，传统系统多把 AI 技术当成一种数据服务型负载的扩展，错过了重要的表现优化的机会。比如，研究型系统 Clipper、商业系统 Amazon Sagemaker 和微软的 AzureML 等系统，将模型使用当成黑盒使用，或只提供普通的系统优化手段，如输入缓存、批量处理等等。像 Pretzel 则只能为传统编译器提供端到端的优化方法。然而，提高机器学习模型的训练效率对于 AI 技术的落地和运行时优化至关重要。这个部分包括模型的训练、优化、管理等诸多方面，有很大的性能提升空间。现有数据库技术主要从三个方面加快机器学习的训练效率：1）基于数据库内核执行 AI 操作，提高执行效率；2）基于 AI 芯片优化 AI 算子的执行，提高计算效率；3）基于新型数据库系统支持 AI 技术，对整个执行流程进行优化。

- **GPU+数据库技术**：GPU 数据库是最早兴起的一批 AI 数据库系统，相关工作包括 Kinetica[29]、MapD[32]、PG-Storm[1]、Blazegraph[2]等等。Kinetica[29]是一款基于 GPU 加速执行引擎的数据库系统。传统 CPU 的每个核心具有取指和调度单元构成的完整前端，因而其核心是多指令流多数据流(Multiple Instruction Multiple Data，MIMD)，每个 CPU 核心可以在同一时刻执行自己的指令，与其他的核心完全没有关系。但这种设计增加了芯片的面积，限制了单块芯片集成的核心数量。GPU 的每个流多处理器才能被看作类似于 CPU 的单个核心，每个流多处理器以单指令流多线程方式工作，只能执行相同的程序。尽管 GPU 运行频率低于 CPU，但由于其流处理器数目远远多于 CPU 的核心数，我们称之为“众核”，其单精度浮点处理能力达到了同期 CPU 的十倍之多。因此，集成了上千个处理核（目前最多能达到 6000 核并行）的 GPU 芯片可以提高大规模并行计算的效率。为了充分发挥 GPU 的优势，Kinetica 支持用

1 http://heterodb.github.io/pg-strom/

2 https://www.blazegraph.com/

户定义函数（User Defined Function，UDF）框架，允许自定义算法和代码直接在数据库中的数据上运行。UDF 使在数据库中运行自定义计算和数据处理成为可能。这提供了一种高度灵活的方法，可以通过网格计算在规模上执行复杂的高级分析，将网络中每台计算机的空闲处理能力连接起来，以承担一个被划分为多个任务的作业。数据库业务和 AI 工作负载可以在同一个 GPU 加速数据库平台上一起运行。用户定义的函数可以用 C++、Java 或 Python 编写。Kinetica 还提供捆绑式 TensorFlow，用于机器学习和深度学习用例。而微软的 Azure 还提供基于云架构的机器学习训练服务，用户可以根据不同的需求快速订制 GPU 集群，批量执行 AI 作业。

- **FPGA+数据库技术：**随着微软利用 FPGA 芯片提高模型训练的灵活性和效率，FPGA 数据库也逐渐兴起[30]。如 MLWeaving 提供一套基于数据库（如 DoppioDB[34,35]）和 FPGA 的硬件加速技术，主要用于提高低精度输入数据的处理效率。FPGA 可以简单地被认为是包含低级基础芯片的电路板，例如 AND 和 OR 门，通常使用硬件描述语言（HDL）指定 FPGA 配置，可以配置成与特定任务或应用程序的需求相匹配的方式。此外，FPGA 能够最大限度地提高并行性和资源利用效率，已经被 Intel 用于加快深度学习网络的实施。MLWeaving 支持 FPGA 主要因为其在探索可能的算法和设计方面提供了更高的通用性，面对快速演进的机器学习算法有很大的意义。MLWeaving 主要提供两个方面的优化：1）支持在任何精度级别上高效的检索输入数据的内存布局；2）基于 FPGA 的设计，提供硬件加速，以加速 SGD 优化算法。而不管使用的精度如何。SGD 是一次一个样本的评估，即在计算梯度之前读取样本的所有特征。在完全精确的情况下，读取与数据点对应的行可以访问该样本的所有特性。而 MLWeaving 在位级别上垂直地划分每个数据点（表中的一行），以便连续存储数据点所有特性的第一位，然后存储第二位等。这提供了两个好处。首先，读取值所需的内存访问数与所用的精度成正比。低精度会导致更少的内存访问。第二，该格式允许以位流的形式将数据序列化到硬件加速器中，从而提高内存带宽利用率。而 Swarm64 系统[1]还提供 CPU+FPGA 的异构芯片支持服务，利用 CPU 阵列处理传统的关系型事务，利用 FPGA 模型训练机器学习模型，更好的提高事务处理的效率和资源的利用率。
- **端到端的执行优化系统：**传统的机器学习使用方式是一种流水线式的：将输入数据处理成数值型的向量，然后在特征向量上执行 ML 模型。计算特征的代价比较高，是一个重要的瓶颈。

  一方面，一些工作主要从最终训练目的的角度预估执行代价，自动生成 ML 执行流水线，如 Kraft 等人提出了一套为人工智能模型提供端到端优化的系统 Willump[36]。机器学习不同于传统负载。首先，很多情况下系统不必计算所有的输入数据，可以选择更加“廉价”的模型来预估原有模型，如不计算所有的输入特征，从而加速训练。其次，不同于传统 SQL 查询封装机制，机器学习算法多直接被高层应用调用。针对以上特点，1）Willump 使用一个完整的程序数据流分析算法和一个成本模型来识别重要但计算成本低的特性，从而有效和准确地加速模型的执行。利用这些特性，Willump 自动训练一个近似模型，该模型可以识别和分类“简单”的数据输入。例如，毒性评论分类的近似模型可能会将带有诅咒词的评论分类为毒性，并将其他评论级联到更强大的模型中；2）Willump 提供查询感知优化。单个数据输入的并行计算是困难的，因为 ML 流水线可能不会自然地并行化，并且通常用缺乏低延迟多

1 https://www.swarm64.com

线程的语言编写。Willump 使用一种数据流分析算法来解决这些挑战，该算法识别 ML 流水线的计算独立组件，并编译为低延迟多线程框架，将延迟提高 4 倍。此外，Willump 将这两方面的优化方法与标准编译器优化方法和传统计算图构建方法相结合，提供自动的 ML 流水线快速推理代码的特征缓存方案，即迭代式的增加参与计算的特征直到表现足够满意。

另一方面，一些工作设计集成式的执行管理系统，帮助执行引擎更好的完成训练任务。如 Amazon SageMaker[1]允许用户方便的创建、训练、部署任意规模的机器学习模型，优势包括：1）免基础设施管理。SageMaker 运行用户定制训练集群（EC2 类型，节点数目），由系统自动进行配置，并在训练任务结束时自动终止进程；2）降低训练开销。EC2 Spot Instance 利用全管理 Spot 训练技术（Managed Spot Training），包括内置 AI 算法、自定义 AI 模型、模型调优等，进一步降低近 90%的训练开销；3）快速异常恢复。SageMaker 利用 Checkpoint 机制，保证程序在异常终止时迅速从最近镜像恢复。

4.4. 智能数据分析技术

数据分析技术是一个非常广泛的领域，包括数据处理、分析、预测等诸多方面[37]。传统数据分析技术有很大的局限性，如算法适应性差、局限性大等。人工智能为数据分析技术提供了新的算法和思路，在数据分析的优化中做出了很大的贡献[38,39]。然而，基于 AI 的数据分析技术仍然有很大的优化空间[40]。根据前三节的工作，我们进一步调研基于智能数据库的数据分析技术。

- **可扩展的数据分析框架：**预测性分析日益成为企业预测需求的重要方式，如估计客户需求或安排日常维护。数据仓库通常为企业存储最有价值的数据集，但这些分析能力传统上是机器学习专家的领域，多是大多数数据分析师或业务用户无法共享的技能。BigQuery ML[41]推出具有扩展的机器学习模型，使数据分析师能够使用熟悉的 SQL 直接在 BigQuery 内部的大型数据集上构建和部署机器学习模型。此外，BigQuery 集成了经典的机器学习模板，如 k-均值聚类和矩阵分解，以进行客户身份细分和产品建议。客户也可以通过 BigQuery ML 建立和直接导入基于 TensorFlow 框架的深度神经网络模型。Geotab 利用 100 多万辆联网车辆的汇总数据提供新的智能城市解决方案。我们能够使用 BigQuery 的地理信息系统来了解交通流模式，而 BigQuery ML 帮助我们深入了解根据恶劣天气预测城市危险驾驶区域。再有，将机器学习应用于表格数据而不编写单行代码。并非所有能从机器学习分析中获益的人都是 SQL 专家。为了使在存储在 BigQuery 和云存储中的结构化数据上更容易应用 ML，我们提供 AutoML 表，让数据科学家、分析师和开发人员等任何用户只需点击几下，就可以在结构化数据上自动构建和部署最先进的机器学习模型，从而将所需的总时间从几周缩短到几天，而无需编写一行代码。
- **智能的近似查询：**对于很多用户来说，非精确但快速响应的数据库查询是传统查询的替代方法。形式概念分析（FCA）的使用使人们能够对经典数据库管理系统（DBMS）提供的答案作出近似的回答[42]。然而，灵活的查询在响应质量上可能代价高昂（尤其是需要计算聚合函数（“sum”、“avg”、“count”、“var”等），因此 Tlili 等人提出了一种机器学习和近似查询处理技术（AQP）相结合的查询处理方法[43]：1）对于聚合查询，他们仍然使用 FCA 生成层次结构的数据路由方法，允许用户将这些响应个性化为多个级别；2）为了降低响应时间，他们使用近似查询处理（AQP），牺牲准确性以降低响应时延；

1 https://aws.amazon.com/cn/sagemaker/

3）为了提高回答准确性，他们同时采用在线聚合技术[41]，在运行应用程序时逐渐接近答案。此外，在线聚合技术包括对数据库的初始数据应用已知的样本信息，以最小化磁盘访问，平衡响应时间和回答准确性。

5 面向 AI 的数据治理引擎

AI 算法和模型依赖高质量的数据，但是现实中数据存在大量的错误和不一致信息。传统数据管理技术包括数据发现、清洗、融合等工作，涉及多模数据管理、多源数据融合等诸多复杂的问题，但是还存在不足，例如质量不高、仍然需要大量的人力投入。因此，在数据管理系统支持 AI 算法时，面对海量异构的训练数据和模型数据，我们需要提供更加高效的数据存储和分析技术。智能数据管理引擎[45]通过结合 AI 技术优化数据库组件，可以为大规模的数据发现、清洗、融合提供更加智能的服务。

5.1 面向AI的数据发现技术

多源异构数据中可能存在大量的信息丢失。此外，很多数据都是稀疏的，不容易发现真正有意义或敏感的数据。这些问题由于缺少统一的评价规则、数据结构复杂、信息不对称等挑战变得非常难以解决。所以下面我们分别介绍如何结合数据管理系统和分布式计算平台，更好的实现数据发现：

- **统一的数据处理架构**：Teradata-for-DL 平台提供了统一的数据处理架构[46]，重新记录不同的数据是如何被存储、管理、分发和持久化的。虽然 Teradata 数据库是一个传统的关系数据库，它使用 ASTER 分析系统进行数据发现，这包括对非结构化或半结构化数据的“极端 SQL”分析和高级分析（非 SQL）。此外，Aster 引入了不同类型数据需要的计算平台，包括引入 MapReduce 提供分布式计算能力；引入图形引擎和 NPath 分析等。 这样，Teradata-SQL-for-DL 能够更好的混合、使用不同的数据源（如传统的关系型数据存储在企业数据仓库中；实时数据流存储在 Hadoop 中等），供深度学习模型学习，如给银行客户的信用进行打分。此外, 它提供在线的 Web 服务，用户可以利用 Restful 框架方便的自定义数据处理的方式。

- **数据编排**：随着来自不同系统、业务流程的数据的容量、速度、多样性的增长，企业需要面对更加复杂的数据访问、发现、管理、安全等问题，导致查找和验证数据集变成一个复杂的手动过程，并且不断增加的法规和合规性要求使其变得更加重要。因此，BigQuery ML 提供了**数据目录技术**，用于数据发现、管理和简化。为了帮助企业快速发现、管理和理解他们的数据资产，数据目录提供完全管理和可扩展的元数据管理服务。数据目录为数据发现提供了一个简单易用的搜索界面，由支持 Gmail 和 Drive 的相同 Google 搜索技术提供支持，并为捕获技术和业务元数据提供了一个灵活而强大的编目系统。安全和数据治理方面，它与云 DLP 系统（Cloud Data Loss Prevention System）集成，用户可以发现、编目敏感的数据资产。此外，它基于谷歌云支持源访问控制列表（ACLs），简化用户访问和管理数据，使用户在企业级数据平台上以可信方式管理数据。

5.2 面向AI的数据清洗技术

传统数据清洗非常繁杂，包括噪声过滤、异常值检测、去重处理等很多方面的工作（数据工作流），用传统

数据库系统需要结合大量的人力投入。而人工智能算法中对数据清洗质量和效率的要求较高。所以，我们分析利用 AI 优化的数据库技术[42]提供数据清洗的工作：

- **自动生成数据流水线**：Shang 等人提出了一款迭代式生成数据工作流的工具 Alpine Meadow[47]。他们的工作主要是将数据工程平民化（democratization），即帮助没有足够专业知识的用户高效的进行数据清洗工作。它需要显著减少构建机器学习流水线式架构所需的专业知识。理想情况下，用户应该能够指定一个高级任务（例如，根据数据预测标签 X），然后系统自动组成一个机器学习流水线来完成该任务，包括所有必要的数据清理、功能工程和超参数调整步骤。Meadow 的核心设计思想是通过模拟经验丰富的数据科学家的决策过程来解决 ML 问题。一个有经验的数据科学家如何处理一个问题？首先，他会检查数据，并根据自己的经验，对特征缩放、嵌入、数据清理等做出高层决策。Meadow 把这个过程映射为一个“数据处理流水线”，并将所有可能抽象成一个有向无环图上的搜索空间。基于数据库中存储的大量数据处理相关的经验，它自动学习构建数据处理流水线的规则。其次，数据科学家可能会使用可靠且经常成功的模型族，例如随机森林，并检查最常见的错误（例如标签不平衡或重复标签列）。在初始结果之后，数据科学家将开始通过添加更复杂的处理步骤、更改模型族、添加/删除特性、增加样本大小等来修改流水线框架。在 Meadow 中，每个逻辑流水线的超参数空间都有一个关联的性能模型，用于查找有潜力的配置。如果从未使用过逻辑流水线，则没有与之关联的任何模型，因此我们开始使用默认或随机配置。一旦第一个结果收集，我们的系统开始选择下一个基于贝叶斯优化的超参数。通过评估不同的物理流水线，我们收集了当前数据集的一些经验，我们可以使用这些经验更新我们的成本和质量模型来选择逻辑流水线，并使用贝叶斯优化模型来选择物理流水线。这是一个迭代和增量的过程。类比数据科学家的工作方式，Meadow 结合数据库的数据管理能力和机器学习模型的学习能力共同完成复杂的数据清洗工作。
- **自动化数据管理**：智能数据库系统[42]可以为大规模共享数据提供智能管理服务。其一，智能数据库系统集成了 AI 组件，帮助实现数据搜索的各类智能操作（AI 优化数据库组件）。其二，智能数据库系统集成了大量数据优化技术，帮助更好的融合不同数据（数据库优化技术）。以**异常值检测**为例，异常值检测广泛用于特征工程，对异常数据进行过滤。它的复杂度随单元的磁盘算法中的单元数呈指数增长，性能会随着单元数和数据点数的增加而降低。一方面，我们可以利用数据库优化技术优化 AI 算法，如 Zhao 等人提出利用索引结构 CD Tree 对异常点检测进行优化[48]。CD Tree 只存储非空的单元格，并使用聚合技术将同一单元格中的数据对象存储到连续的磁盘页中，从而大大提升了异常值检测的表现。另一方面，我们可以用 AI 克服数据多样性的挑战，如目前数据库系统会综合性的存储多模数据，Xu 等人提出基于希尔伯特指数的异常检测算法[52]，将不同的数据类型抽象到度量空间中，实现更高的检测速度。

5.3 面向AI的数据融合技术

复杂的 AI 算法需要大量的数据作为训练集。一个训练集往往有多个数据来源，如社交评论、访问信息、交易信息等，存在大量不同的数据格式，如 csv、json、jpg 等。对这些异构数据直接进行建模、类型转换、连接等开销较大。所以，我们分析利用 AI 优化的数据库技术[45]提供数据融合的工作：

- **降低数据生成开销：**在大规模训练场景下，训练集往往来自多个数据源，因此需要 SQL 提供多源数据融合服务。对于任何机器学习算法，我们通常假设数据集是一个单一的表。但现实生活中数据集往往是由多个存在主外键依赖的表组成的。将它们直接连接再进行特征选择，非常浪费时间。而且特征数量的增加可能会使分析人员更难探索数据，而且还会增加机器学习算法和特性选择方法的执行时间。因此，Kumar 等人[50]提出了一种分析主外键（Key-Foreign Key）对机器学习表现和准确度影响的方法。首先，他们基于信息论的方法增加价值评估功能，评估每个原始特征对于模型的“有用程度”。然而，被评定为“冗余”的特征（redundancy）可能在预测有用特征方面有价值。因此，然后他们之后评估特征之间的相关性（relevancy），基于模型特点在冗余-相关性之间折中。最后，面对核心问题：如何预测一个先验，即判断一个与 r 表的连接是否是可以安全避免的，他们从使用“受控”数据集的模拟研究开始，以验证理论分析，并精确测量当改变标准化数据的不同属性时误差的变化情况。最后，他们解释得出的决策规则以及如何使用模拟测量来调整特征 join 方式。这种方法可以在提高机器学习模型表现的同时不丧失准确性，并帮助将机器学习算法应用在数据库的数据处理流程中。此外，Kumar 等人还提出基于线性回归模型优化数据源的连接操作。对于一个称为广义线性模型（GLMs）的大型通用 ML 技术类，可以在不牺牲质量和可伸缩性的情况下学习连接并避免冗余。他们主要研究用于连接操作的混合哈希算法，用于快速估计所有方法的 I/O 和 CPU 成本。此外，他们提出了三种在单节点 RDBMS 中的连接上运行批量梯度下降（Batch Gradient Descent，BGD）的替代方法：流、流重用和分解学习。每种方法都避免了某种形式的冗余。流避免写入关系表并可以保存在 I/O 上；流重用利用了 BGD 的迭代特性，并避免了在第一次迭代后重新划分基关系。但是，这两种方法都不能避免 BGD 计算中的冗余。因此，他们设计了避免计算冗余的因式分解学习方法。分解学习法通过交错连接操作和 BGD 的计算和 I/O 来实现这一点。我们的方法都不影响模型的质量。此外，使用用户定义的聚合函数（UDAFs）的抽象，很容易的在 RDBMS 中实现，这提供了可伸缩性和易于部署的能力[45,51]。
- **简化数据建模：**数据库研究的一个主要目标是将附加语义合并到数据模型中。经典的数据模型由于无法表示和处理许多实际应用中可能出现的不精确和不确定的信息而受到影响。因此，模糊集理论已广泛应用于各种数据模型的扩展，满足复杂对象不精确和不确定性建模的需要。为了更恰当地描述这种关系并更好地利用现有值，Lai 等人提出了一种不完全数据建模方法来输入缺失值[52]。该方法利用不完全记录和完整记录建立 Takagi-Sugeno（TS）模型。在这个过程中，不完整的数据集被分成几个子集，并且只包含重要变量的线性函数被建立来描述每个子集中属性之间的关系。这套方法能为存在一定量缺失值的数据集很好的建立数据模型并提高数据准确性。

- **智能数据融合：**训练集数据的结构和价值可能有很大差异。企业数据池多基于 Hadoop 等大数据引擎，需要将大量业务相关的数据源聚合在一起，然而这种数据聚合的速度对于机器学习来说太慢了。基于如 Kinetica[29]、IBM Watson Analytics[1]等结合人工智能数据库的平台，首先，我们可以利用大规模 GPU 芯片组快速进行数据编解码和类型转换；其次，IBM Watson Analytics 还允许直接查询各种数据库，包括 Cloudera Impala、MySQL、Oracle、PostgreSQL 等。它有 32 个连接器可以方便地使用来自这些源的

1 https://www.ibm.com/watson-analytics

数据，近乎实时的进行数据同步和融合。该工具使那些具有深厚数据科学技能的人能够跳过数据融合的复杂工作，直接进入模型设计阶段。

6 研究展望与未来趋势

6.1 面向AI+DB的统一的数据模型

目前数据库系统支持 AI 操作存在数据模型不一致的问题。数据库中的数据多是关系型数据，而 AI 操作存在大量张量型数据。这导致在用数据库管理 AI 时存在大量的数据存储开销，如存在格式转换、数据丢失等问题，而且会降低执行效率，如在用机器学习做图片分类问题时，图片都是张量格式，而在进行类别判断（如“删掉标签‘猫’”等），需要进行一些标量计算，导致执行引擎需要在不同的数据模型上分别操作，增加了执行开销。所以，未来我们需要研究用统一的数据模型同时支持标量、向量、大规模张量等不同类型的数据操作，更好的支持 AI 计算和数据库查询。但是现有的数据模型（如关系表、键值对、时间序列等），都有着各自的局限性，所以 Idreos 等人提出了一种学习型数据结构设计引擎 **Data Alchemist**[30]。它混合不同粒度的数据结构的设计原则，构成一般化的设计空间。通过在设计空间中对不同的组合、调优和性能进行分析，自动生成合适的数据结构。

6.2 面向AI的优化器

尽管已经有工作利用声明性的语言模型支持 AI 算法，但是在优化器生成实际 AI 操作阶段存在很多问题。一种方法是直接把类 SQL 程序转化成 Python、R 等语言执行（如 SQLFlow、Rheem 等），不仅增加了语言转化开销，也没有对算法流程进行优化。而另一种方法预先针对 AI 算法定义了新的算子[7]，借助代码生成器将类 SQL 语句翻译成 AI 算子序列，但是不同于传统数据库查询生成执行计划，没有对算子类型选择以及算子之间的连接关系做有效的优化。因此，未来我们将研究面向 AI 的优化器。首先，统计 AI 执行相关的物理信息（如芯片频率、磁盘读写速度等），建立 AI 算子的代价估计机制。其次，设计针对 AI 的执行计划选择机制。因为不同于数据库查询，除了数据建立的联系，AI 算子之间还存在环境、计算等依赖关系，或没有显式关系（泛化为“执行森林”）。所以计划选择算法（如泛化的动态规划、强化学习算法等）需要基于代价估计的结果，智能的选择合适的运行环境、算子类型、组织逻辑等，在算法的优化程度和选择效率之间实现平衡。

6.3 AI+DB联合优化

AI 和数据库融合技术有非常大的发展潜力。首先，通过结合 AI 技术（AI for DB），智能数据库系统可以实现组件学习化，包括智能参数调优[56,58]、优化器[60,61]、物理设计[62,63]、表现预测[64]等等。比如，学习型参数调优[56,58]通过理解不同负载特点，动态的调整参数来实现整体吞吐量和单条查询执行效率之间的平衡；智能计划选择[61]利用机器学习模型，从历史数据中学习对不同连接方式的代价估计方法，进而选择合适的执行计划；动态索引选择[62]基于强化学习等算法，针对不同的查询和数据特点选择合适的索引，平衡索引建立开销和查询的执行效率。因此 AI 技术可以全方位的提高数据库处理海量查询、异构场景的能力。其次，前面我们已经介绍了基于传统数据库对 AI 易用性、执行效率、数据质量进行优化的方法，而基于智能数据库系统，我们可以进一步提供智能灵活的 AI 解决方案，包括：1）智能的 AI 语言模型：结合 NLP 等方面的研究成果，数据库可以支持自然语言级别的 SQL 翻译，允许基于用户对问题的模糊描述给出较为精确的分析结果，进一步实现 AI 平民化；2）智能的 AI 算法优化引擎：目前生成类数据库的 AI 执行计划仍然是很有挑战性的，包括对大量不同的 AI 算法统一建模、在大规模算子空间中进行算法匹配等等。利用深度学习、强化学习等算法，

可以更快更好的学习算法组织策略，并根据需求变化智能的调整优化目标；3）智能的 AI 执行引擎：目前数据库执行引擎需要调度不同类型的 AI 算子和传统数据库算子，负担较大。结合 AI 技术，智能 AI 执行引擎可以预先估计 AI 算法可能的计算、存储需求，节省资源轮询的开销。此外，通过智能的调配空闲资源，允许等待的任务提前执行，更加充分的使用不同的硬件资源，提高整体训练效率。再有，AI 和数据库融合技术为下一代计算方法指明了方向。面对海量异构数据，传统的计算方法存在用时长（普通数据库处理查询）、结果不够准确（AQP）等问题，通过结合深度学习等技术，我们可以在一个 KB 级神经网络中压缩 TB 级数据信息，根据问题在毫秒级别给出计算结果（前向传播），为传统计算方法带来革命性的创新和优化。AI 原生数据库通过结合 AI 较强的学习、计算能力和数据库系统的数据处理、管理等经验，降低 AI 使用门槛的同时，可以大幅度提升数据库各方面的性能，为下一代数据库和 AI 技术的发展指明了一个方向。

6.4 大规模参数调优

在今天的机器学习任务中，随着训练数据和模型的增长，很难在单个服务器上执行训练过程，这就需要分布式机器学习方法（Distributed Machine Learning）在多个服务器之间划分工作负载[29]。比如，训练数据的实际数量如果在 1TB 到 1PB 之间，就可以创建具有 10^9 到 10^{12} 个参数的强大和复杂的模型，这些模型通常由所有工作节点全局共享，在执行计算以优化共享参数时，这些工作节点必须经常访问共享参数[53]。为了更好地管理、优化、分析这些共享参数，目前有很多参数服务器方面的研究[54,55,56]，支持大规模参数的分布式存储和协同。但是这些工作多需要基于独立的服务器，或搭载在主流的分布式机器学习系统上（如 TensorFlow）。而目前已经有很多比较成熟的分布式数据库系统（Distributed DBMS），通过在事务工作负载中利用并行性来实现更高的性能[51]。未来如果用分布式数据库支持参数服务器，我们可以将机器学习训练任务作为负载下发给分布式数据库系统，一方面，由数据库自动的进行任务调度和执行，简化分布式机器学习算法的实现；另一方面，将全局共享参数表示为关系型数据，帮助算法在更细粒度上实现并行操作，提高训练任务的执行效率。此外，利用数据库的可靠性控制机制（如自动诊断、错误容忍、自动恢复等），我们可以有效解决参数更新同步等问题，避免数据不一致带来的资源开销和结果误差[57]。

6.5 错误容忍的深度学习系统

深度学习模型训练不具有错误容忍能力，进行分布式训练时，一个进程崩溃，整个任务就会失败。而且放在内存中的数据更容易出现这类问题。因此，未来我们可以结合数据库系统的错误容忍技术，提高深度学习的鲁棒性。为了确保在可预见和不可预见的人为或自然灾害下的业务连续性，数据库系统都必须从以下两个方面保证容错和灾难恢复能力。首先，由相同或等效系统备份的硬件系统。例如，可以通过使用并行运行的相同服务器，将所有操作镜像到备份服务器，使服务器具有硬件上的容错性。其次，由其他软件实例备份的软件系统。例如，具有客户信息的数据库可以连续地复制到另一台机器上（复制集）。如果主数据库关闭，操作可以自动重定向到第二个数据库。此外，目前还有在流式处理系统上提供的近似错误容忍技术[33]，通过适应性的发布备份（如仅当错误超过用户定义的可接受级别时才发出备份操作），确保故障时的误差受到理论保证的限制。

6.6 大规模分布式训练

分布式机器学习包括多节点的机器学习算法和系统，用于提升执行性能、提高计算精度、扩展到更大的输入数据（帮助显著减少学习的错误率）[51,58]，因此已经被广泛使用在真实生产环境中，帮助公司、研究人员和

个人从大量数据中得出有意义的结论。然而分布式机器学习仍然面临很多亟待解决的问题，如它不支持弹性调度。比如一个有 N 个 GPU 的集群上在运行一个作业，使用了一个 GPU。此时一个新提交的作业要求使用 N 个 GPU，因为空闲 GPU 个数是 N-1，所以这个新的作业不能开始执行，而是得一直等数小时甚至数天，直到前一个作业结束、释放那个被占用的 GPU。这么长时间里，集群利用率 ＜1/N，导致集群利用率很低。所以，我们可以结合数据库系统，提供多方面的系统性能优化：1）一致性：数据库系统能够有效的保证训练一致性，如在多个节点为同一个任务工作时，确保在不同分区的同一套全局数据的一致性；2）容错性：如上节讨论的，当我们把一个负载任务分发到 10000 个计算节点上，在一个节点宕机时仍然能够保持训练（如根据最近的历史镜像回退），避免重新训练；2）通信：分布式机器学习涉及大量的 I/O（如磁盘读写）和跨节点数据传递过程。基于数据库的存储引擎，我们能高效的调度不同类型环境（如单节点磁盘系统、分布式文件系统等）上的 I/O 操作，提供无阻塞的数据处理过程；3）资源管理：构建和维护一个计算集群的成本较高，因此一个集群通常由许多用户共享。基于数据库的负载调度机制，在最大化利用率的同时，帮助管理集群并适当地分配资源以满足每个人的请求（个性化定制）。

7 总结

本文综述了支持人工智能的数据管理技术。针对人工智能技术中存在的使用门槛高、数据密度高、算力要求高等问题和挑战，我们从数据管理系统的层次角度出发，分别分析不同层级数据管理方法如何优化人工智能算法。我们首先概述了支持人工智能的数据管理技术的整体架构，给出了不同层级的主要内容，然后针对每一层次分别展开综述。在声明性语言模型方面，我们分别概述声明性 SQL 语言的扩展、逻辑封装与优化技术；在算法优化引擎优化 AI 方面，我们分别概述算子代价估计与选择、算法组装与选择、模型管理等技术；在异构执行引擎方面，我们分别从新硬件和分布式架构两方面概述异构 AI 算子的计算和优化方法；在智能数据治理方面，我们分别概述多模、多源数据存储和基于智能数据库的数据查询优化、数据分析优化和数据建模简化方面的工作。最后，我们讨论了支持人工智能的数据管理技术的发展方向，并给出进一步的展望。

致　谢　本文由国家自然科学基金（61925205, 61632016）、华为、好未来教育公司支持。

References:

[1] Zhou X, Chai C, Li G, Sun J. Database Meets Artificial Intelligence: A Survey[J]. IEEE Transactions on Knowledge and Data Engineering, 2020.

[2] Li G, Zhou X, Sun J, Yu X, Yuan H, Liu J, Han Y. A Survey of Machine-Learning-based Database Optimization Techniques. A survey. Chinese Journal of Computers, 2019, :1-33 (in Chinese with English abstract).

[3] Chai M, Fan J, Du X. Learnable Database Systems: Challenges and Opportunities. Journal of Software, 2020,31(3): 806-830 (in Chinese with English abstract).

[4] Timo Schindler, Christoph Skornia. Secure Parallel Processing of Big Data Using Order-Preserving Encryption on Google BigQuery. CoRR, 2016, abs/1608.07981.

[5] Natwadee Ruedeeniraman, Makoto Ikeda, Leonard Barolli. TensorFlow: A Vegetable Classification System and Its Performance Evaluation. IMIS, 2019, 132-141.

[6] Rory Mitchell, Andrey Adinets, Thejaswi Rao, Eibe Frank. XGBoost. Scalable GPU Accelerated Learning. CoRR, 2018, abs/1806.11248.

[7] Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann and et al. The MADlib Analytics Library or MAD Skills, the SQL. PVLDB, 2012, 5(12): 1700-1711.

[8] J. MacGregor. Predictive Analysis with SAP: The Comprehensive Guide. SAP PRESS, 2013.

[9] Xiangrui Meng, Joseph K. Bradley, Burak Yavuz and et al. MLlib: Machine Learning in Apache Spark. J. Mach. Learn. Res, 2016, 17(34):1-34.

[10] X. Li, B. Cui, Y. Chen, W. Wu, and et al. Mlog: Towards declarative in-database machine learning. PVLDB, 2017, 10(12):1933–1936.

[11] C. Ordonez. Integrating k-means clustering with a relational DBMS using SQL. TKDE, 2006, 18(2):188–201.

[12] A. Kumar, J. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. SIGMOD, 2015, pages 1969–1984.

[13] Yi Zhang, Kamesh Munagala, Jun Yang. Storing Matrices on Disk: Theory and Practice Revisited. PVLDB, 2011, 4(11): 1075-1086.

[14] M. Aref, B. ten Cate, T. J. Green and et al. Design and implementation of the LogicBlox system. SIGMOD, 2015, pages 1371–1382.

[15] C. Borraz-Sánchez, J. Ma, D. Klabjan and et al. Algebraic modeling in Datalog. http://dynresmanagement.com/uploads/3/3/2/9/3329212/datalog_modeling.pdf.

[16] Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. PVLDB, 4(6):373–384, 2011.

[17] Z. Cai, Z. Vagena, L. Perez, S. Arumugam and et al. Simulation of database-valued Markov chains using SimSQL. SIGMOD, 2013, pages 637–648.

[18] Green T J, Olteanu D, Washburn G. Live programming in the LogicBlox system: a MetaLogiQL approach. PVLDB, 2015, 8(12): 1782-1791.

[19] Manasi Vartak, Harihar Subramanyam, Wei-En Lee and et al. ModelDB: a system for machine learning model management. HILDA@SIGMOD, 2016, pages 1-3.

[20] R. Jampani, F. Xu, M. Wu and et al. The Monte Carlo Database System: Stochastic analysis close to the data. TODS, 2011, 36(3):1–41.

[21] Kaan Kara, Ken Eguro, Ce Zhang and et al. ColumnML: Column-Store Machine Learning with On-The-Fly Data Transformation. PVLDB, 2019, 12(4), 348-361.

[22] M. Kandemir, H. Zhao, X. Tang, and M. Karakoy. Memory row reuse distance and its role in optimizing application performance. Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2015, pages137-149.

[23] Zhang C, Ré C. Dimmwitted: A study of main-memory statistical analytics[J]. arXiv preprint arXiv, 2014, 1403.7550.

[24] C. Balkesen and et al. Multi-core, main-memory joins: Sort vs. hash revisited. PVLDB, 2013, 7(1):85–96.

[25] S. Shalev-Shwartz and A. Tewari. Stochastic methods for l1-regularized loss minimization. Journal of Machine Learning Research, 2011, 12(Jun):1865–1892.

[26] R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 1996, pages 267–288.

[27] M. Jaggi, V. Smith, M. Takac and et al. Communication-efficient distributed dual coordinate ascent. In Advances in Neural Information Processing Systems, 2014, pages 3068–3076.

[28] G. Guennebaud, B. Jacob and et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[29] Jeffrey M. Rzeszotarski, Aniket Kittur. Kinetica: naturalistic multi-touch data visualization. CHI, 2014, pages 897-906.

[30] S. Idreos, K. Zoumpatianos, M. Athanassoulis and et al. The Periodic Table of Data Structures. IEEE Data Engineering Bulletin, 2018, 41(3):64–75.

[31] Divy Agrawal, Mouhamadou Lamine Ba, Laure Berti-Équille and et al. Rheem: Enabling Multi-Platform Task Execution. SIGMOD, 2016, pages 2069-2072.

[32] Christopher Root, Todd Mostak. MapD: a GPU-powered big data analytics and visualization platform. SIGGRAPH Talks, 2016, 73(2):1-73.

[33] Katayoun Neshatpour, Maria Malik, Houman Homayoun. Accelerating Machine Learning Kernel in Hadoop Using FPGAs. CCGRID, 2015, pages 1151-1154.

[34] Kaan Kara, Zeke Wang, Gustavo Alonso, Ce Zhang. doppioDB 2.0: Hardware Techniques for Improved Integration of Machine Learning into Databases. PVLDB, 2019, 12(12): 1818-1821.

[35] David Sidler, Zsolt István, Muhsen Owaida and et al. doppioDB: A Hardware Accelerated Database. SIGMOD, 2017, pages 1659-1662.

[36] Peter Kraft, Daniel Kang, Deepak Narayanan and et al. Willump: A Statistically-Aware End-to-end Optimizer for Machine Learning Inference. CoRR, 2019 abs/1906.01974.

[37] Stéphane Marchand-Maillet, Birgit Hofreiter. Big Data Management and Analysis for Business Informatics - A Survey. Enterprise Modelling and Information Systems Architectures, 2014, 9(1): 90-105.

[38] Vasilios P. Androvitsaneas, Konstantinos Boulas, Georgios D. Dounias: Intelligent Data Analysis in Electric Power Engineering Applications. Machine Learning Paradigms, 2019, pages 269-313.

[39] Cynthia Rudin, David Carlson. The Secrets of Machine Learning: Ten Things You Wish You Had Known Earlier to be More Effective at Data Analysis. CoRR, 2019, abs/1906.01998.

[40] Kevin Ross, Melody Moh, Teng-Sheng Moh and et al. Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection. ACM Southeast Regional Conference, 2018, 1(1):1-8.

[41] Sérgio Fernandes, Jorge Bernardino. What is BigQuery? IDEAS 2015, pages 202-203.

[42] María José Benítez-Caballero, Jesús Medina, Eloisa Ramírez-Poussa. FCA Attribute Reduction in Information Systems. IPMU, 2018, pages 549-561.

[43] Oussama Tlili, Minyar Sassi, Habib Ounelli. Intelligent Database Flexible Querying System by Approximate Query Processing. CoRR, 2012, abs/1204.3223.

[44] Yun Li, Yanlong Wen, Xiaojie Yuan. Online Aggregation: A Review. WISA 2018., pages 103-114.

[45] Neelu Nihalani, Sanjay Silakari, Mahesh Motwani. Integration of Artificial Intelligence and Database Management System: An Inventive Approach for Intelligent Databases. CICSN, 2009, pages 35-40.

[46] Sam Madden. From Databases to Big Data. IEEE Internet Computing, 2012, 16(3): 4-6.

[47] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti and et al. Democratizing Data Science through Interactive Curation of ML Pipelines. SIGMOD, 2019, pages 1171-1188.

[48] Huanliang Sun, Yubin Bao, Faxin Zhao and et al. CD-Trees: An Efficient Index Structure for Outlier Detection. WAIM 2004, pages 600-609.

[49] Honglong Xu, Haiwu Rong, Rui Mao and et al. Hilbert Index-based Outlier Detection Algorithm in Metric Space. IJGHPC, 2016, 8(4): 34-54.

[50] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, Xiaojin Zhu. To Join or Not to Join?: Thinking Twice about Joins before Feature Selection. SIGMOD, 2016, pages 19-34.

[51] X. Feng, A. Kumar, B. Recht, and C. R´e. Towards a Unified Architecture for in-RDBMS Analytics. SIGMOD, 2012, pages 325-336.

[52] Xiaochen Lai, Xin Liu, Liyong Zhang and et al. Missing Value Imputations by Rule-Based Incomplete Data Fuzzy Modeling. ICC 2019, pages 1-6.

[53] Jinkun Geng, Dan Li, Shuai Wang. Accelerating Distributed Machine Learning by Smart Parameter Server. APNet, 2019, 92-98.

[54] Mu Li, David G. Andersen, Jun Woo Park and et al. Scaling Distributed Machine Learning with the Parameter Server. OSDI, 2014, pages 583-598.

[55] M. Abadi, P. Barham, J. Chen and et al. Tensorflow: A system for largescale machine learning. OSDI, 2016, 265-283.

[56] Guoliang Li, Xuanhe Zhou, Shifu Li, Bo Gao. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. Proc. VLDB Endow, 2019, 12(12): 2118-2130.

[57] Guoliang Li, Xuanhe Zhou, Sihao Li. XuanYuan: An AI-Native Database. IEEE Data Eng. Bull, 2019, 42(2): 70-81.

[58] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, Zekang Li. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. SIGMOD, 2019, pages 415-432.

[59] Marcel Kornacker, Alexander Behm, Victor Bittorf and et al. Impala: A modern, open-source SQL engine for Hadoop. In Proceedings of the 7th Biennial Conference on Innovative Data Systems Research, 2015.

[60] J. Sun and G. Li. An end-to-end learning-based cost estimator. PVLDB, 13(3):307–319, 2019.

[61] Xiang Yu, Guoliang Li, Chengliang Chai, Nan Tang. Reinforcement learning with tree-lstm for join order selection. ICDE, 2020, pages 196–207.

[62] Xuanhe Zhou, Ji Sun, Guoliang Li, Jianhua Feng. Query Performance Prediction for Concurrent Queries using Graph Embedding. Proc. VLDB Endow, 2020, 13(9): 1416-1428.

[63] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, Yue Han. Automatic View Generation with Deep Learning and Reinforcement Learning. ICDE, 2020, pages 1501-1512.

[64] Zahra Sadri, Le Gruenwald, Eleazar Leal. Online index selection using deep reinforcement learning for a cluster database. ICDEW, 2020.

附中文参考文献:

[2] 李国良,周煊赫,孙佶,余翔,袁海涛,刘佳斌,韩越.基于机器学习的数据库技术综述[J/OL].计算机学报, 2019:1-33. http://kns.cnki.net/kcms/detail/11.1826.TP.20191104.1009.002.html.

[3] 柴茗珂,范举,杜小勇.学习式数据库系统:挑战与机遇[J].软件学报,2020,31(03):806-830.