软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

XuanYuan：AI 原生数据库系统*

李国良，周煊赫，冯建华

清华大学计算机系

通讯作者：李国良, E-mail: liguoliang@tsinghua.edu.cn

摘　要：　大数据时代下，数据库系统主要面临着三个方面的挑战。首先，基于专家经验的传统优化技术（如代价估计，连接顺序选择，参数调优）已经不能满足异构数据、海量应用和大规模用户对性能的需求。我们可以设计基于学习的数据库优化技术，使数据库更智能。其次，AI 时代很多数据库应用需要使用人工智能算法，如数据库中的图像搜索。我们可以将人工智能算法嵌入到数据库，利用数据库技术加速人工智能算法，并在数据库中提供基于人工智能的服务。再者，传统数据库侧重于使用通用硬件（如 CPU），不能充分发挥新硬件（如 ARM、AI 芯片）的优势。此外，除了关系模型，数据库需要支持张量模型来加速人工智能操作。为了解决这些挑战，我们提出了一个原生支持人工智能（AI）的数据库系统。一方面，我们将各种人工智能技术集成到数据库中，以提供自监控、自配置、自优化、自诊断、自愈、自安全和自组装功能。另一方面，我们通过使用声明性语言让数据库提供人工智能功能，以降低人工智能使用门槛。本文介绍了实现人工智能原生数据库的五个阶段，并给出了设计人工智能原生数据库的挑战。我们还以自主数据库调优、基于深度强化学习的查询优化、基于机器学习的基数估计和自主索引/视图推荐为例，展示了人工智能原生数据库的优势。

关键词：　数据库；人工智能；计算框架

中图法分类号：TP311

XuanYuan: an AI-Native Database

Guoliang Li, Xuanhe Zhou

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

Abstract:　In big data era, database systems face three challenges. Firstly, the traditional empirical optimization techniques (e.g., cost estimation, join order selection, knob tuning) cannot meet the high-performance requirement for large-scale data, various applications and diversified users. We need to design learning-based techniques to make database more intelligent. Secondly, many database applications require to use AI algorithms, e.g., image search in database. We can embed AI algorithms into database, utilize database techniques to accelerate AI algorithms, and provide AI capability inside databases. Thirdly, traditional databases focus on using general hardware (e.g., CPU), but cannot fully utilize new hardware (e.g., ARM, GPU, AI chips). Moreover, besides relational model, we can utilize tensor model to accelerate AI operations. Thus, we need to design new techniques to make full use of new hardware. To address these challenges, we design an AI-native database. On one hand, we integrate AI techniques into databases to provide self-configuring, self-optimizing, self-monitoring, self-diagnosis, self-healing, self-assembling, and self-security capabilities. On the other hand, we enable databases to provide AI capabilities using declarative languages in order to lower the barrier of using AI. In this paper, we introduce five levels of AI-native databases and provide several open challenges of designing an AI-native database. We also take autonomous database knob tuning, deep reinforcement learning based optimizer, machine-learning based cardinality estimation, and autonomous index/view advisor as examples to showcase the superiority of AI-native databases.

* 基金项目：国家自然科学基金(61632016, 61521002, 61661166012); 973 项目（2015CB358700）
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: database; artificial intelligence; computing architecture

1 前言

随着信息时代的发展，数据库系统已经被广泛应用在银行、政府、电信等多个领域，在数据存储、数据管理、查询处理、数据分析、商业决策等方面发挥了极其重要的作用。在过去的 50 年里，数据库主要经历了三次变革：

第一代是单机数据库，解决了数据存储、数据管理和查询处理等问题[2]。代表系统包括 PostgreSQL 和 MySQL 等。

第二代是集群数据库，旨在为企业关键业务提供高可用性和可靠性保障。代表系统包括 Oracle RAC、IBM DB2 和 Microsoft SQL Server。

第三代是分布式数据库和云原生数据库，旨在解决大数据时代的弹性计算和动态数据迁移问题[3]。代表系统包括亚马逊 Aurora，华为 GaussDB 和阿里的云数据库。

但是，大数据和 AI 时代带来了新的挑战：大规模异构数据、海量异构应用、大规模异质用户、多种异构计算资源，传统的数据库在大数据时代仍然存在着诸多局限性：

（1）传统的数据库设计仍然基于经验方法和人工规则，需要大量的人员参与（如 DBA）来调整和维护数据库。人工智能技术可以从三个方面提升数据库可用性。首先，数据库中有数百个参数，需要 DBA 手动调整以适应不同的场景。最近，数据库社区试图利用机器学习技术[1,9,18]自动调整参数，可以获得比 DBA 更好的结果。第二，数据库优化器依赖于代价和基数估计，但是传统的技术不能提供准确的估计。最近人们提出了基于深度学习[6,14]来估计查询代价和基数，也获得了更好的结果。此外，基于学习模型的优化器[8,12]、索引推荐[15]、自动视图生成[10]，也为数据库设计提供了可选的优化机会。第三，传统的数据库是由数据库架构师根据经验设计。最近一些基于学习的自设计技术，如学习型索引[7]、学习型 NoSQL 数据库设计[5]等，可以利用人工智能技术来优化数据库，使数据库更智能。

（2）传统数据库以关系模型为核心，提供关系型数据管理和分析能力。然而，在大数据时代，数据（如图数据、时间序列数据、空间数据）和应用（如机器学习、图计算）越来越多样化，需要一个新的数据库系统统一支持多个模型（如关系模型、图模型、张量模型），以支持多种应用（如关系型数据分析、图形计算和机器学习）。我们可以将人工智能算法嵌入数据库，设计数据库内机器学习框架，并利用数据库技术加速和优化人工智能算法，在数据库中提供人工智能能力。

（3）传统数据库多只支持通用硬件，如 CPU、RAM、磁盘等，不能充分利用 ARM、AI 芯片、GPU、FPGA、NVM、RDMA 等新硬件。数据库需要一个异构的计算框架，能够有效地利用各种计算能力来支持数据管理、数据分析和数据库内置的机器学习方法。

为了解决这些问题，我们提出了一个人工智能原生数据库（轩辕 XuanYuan）。它不仅将人工智能技术集成到数据库中，使数据库更加智能化，而且提供了数据库中的人工智能服务。特别的是，一方面，轩辕数据库将人工智能技术集成到数据库中，为数据库提供自配置、自优化、自监控、、自诊断、自愈、自安全、自组装等功能，提高数据库的可用性和稳定性，降低人力密集型应用的开销。另一方面，轩辕数据库通过使用声明性语言提供人工智能服务，以降低使用人工智能的门槛。此外，轩辕数据库还充分利用多种异构硬件的计算能力，支持数据分析和机器学习。

轩辕数据库可以分为五个阶段。第一阶段是 AI 建议型数据库（AI-advised），人工智能引擎以外挂插件的形式为数据库提供离线建议，如离线索引推荐、离线参数优化等。第二阶段是 AI 辅助型数据库（AI-assisted），它将人工智能引擎内置到数据库中，提供在线监测和建议，如在线统计采集、在线数据库状态监测和在线诊断等。第三阶段是 AI 增强型数据库（AI-enhanced），它一方面提供基于人工智能的数据库组件，例如，学习

型索引、学习型优化器、学习型成本估算、学习型存储布局，另一方面，提供了基于数据库技术的人工智能算法和加速器。第四阶段是 AI 组织型数据库（AI-assembled），它提供多种数据模型（如关系模型、图模型、张量模型），并充分利用新的硬件支持异构计算（如支持 AI 芯片、支持 ARM 芯片）。一方面，它可以为数据库每一层处理提供多个选项，例如，基于代价的优化器、基于规则的优化器、学习型优化器等，可以自动组装数据库组件以基于负载自动组装数据库，保证在不同的场景下获得最佳性能。其思想类似于乐高积木，我们提供不同类型的积木块（数据库组件），不同积木有着标准的接口，可以根据用户需求自动组装这些组件来搭建用户满意的积木形态（数据库）。而拼装的过程类似于 AlphaGo，它可以探索比人类更多的优化空间，找到更好的组合方案。第五阶段是 AI 设计型数据库（AI-designed），它将人工智能集成到数据库设计、开发、评估和维护的整个生命周期中，为各个场景提供最佳的性能。

本文首先给出人工智能原生数据库设计思想和理念，并介绍各个发展阶段的详细情况。

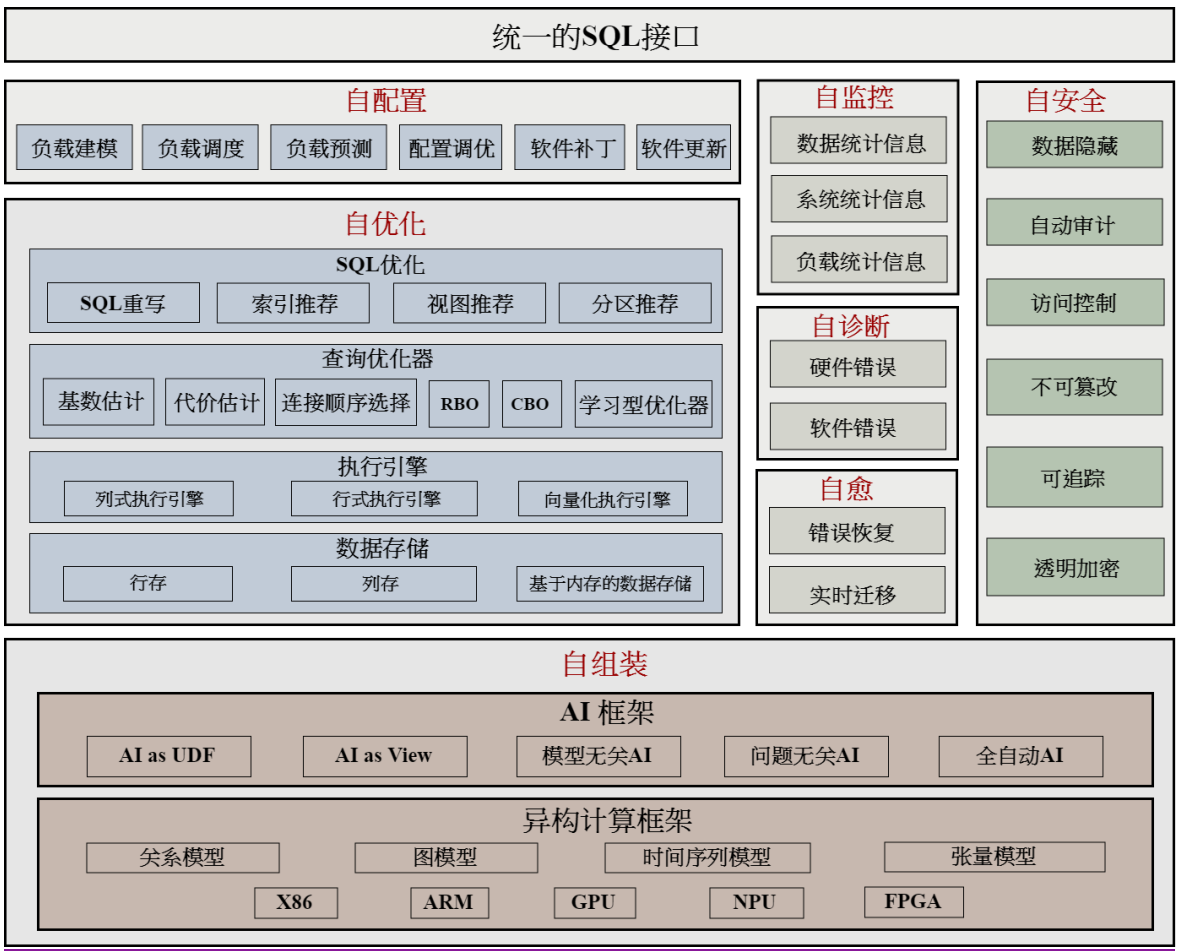

Fig.1 Architecture of AI-Native database

图 1 AI 原生数据库的架构

2 动机

本章讨论如何将人工智能和数据库技术融合，一方面提升数据库的智能性，另一方面降低 AI 使用门槛。首先，我们讨论如何利用人工智能技术优化数据库（AI4DB）。其次，我们介绍如何利用数据库技术帮助人工智能技术落地（DB4AI）。最后，我们讨论如何充分利用新的硬件来同时支持人工智能和数据库技术。

2.1 AI4DB

人工智能技术可以在多个维度为数据库赋能。其一，人工智能模型可以作为外挂工具为数据库提供优化服务。比如外挂的 SQL 重写工具，先给出 SQL 语句的优化建议，再传递给数据库内核执行。其二，人工智能模型可以融合到数据库内部，提供运行时优化。比如我们可以把配置优化模块植入查询优化器，提供查询级别的参数调优。其三，我们可以用人工智能算法代替数据库传统组件。比如提供学习型优化器、学习型索引、学习型视图构建等等。我们甚至可以利用人工智能算法对数据库重组织，根据负载特性自动生成合适的执行路径。其四，我们可以把人工智能技术融入到数据库的整个生命周期，在数据库部署的各个阶段提供智能优化。其五，用人工智能技术改造查询优化模块。由于合理的执行计划对于查询处理非常重要，我们可以通过在查询解析、处理、优化的各个层次结合合适的人工智能技术，提高执行计划的生成效率和表现。其六，用人工智能技术优化数据库运维。数据库运维对于保证数据库可用性、可靠性等诸多方面有着重要意义。而数据库运维是一个非常复杂的工作，包括状态监控、问题诊断、性能评估、意外处理等多个方面。通过结合人工智能技术，数据库可以实现自诊断、自愈、自监控，进而解放人力实现自动运维。其七，用人工智能技术保证数据库安全。数据库安全对于确保用户隐私信任非常重要。传统的数据隐藏、加密技术多有被破解的隐患，不能保证足够高的安全性。通过结合深度学习，利用复杂的神经元连接方式进行数据隐藏、加密和访问控制，我们可以极大提升数据库的安全性。

2.2 DB4AI

人工智能技术严重依赖于数据。通过有效结合数据库在存储、管理和操作数据上的优势，人工智能的训练和学习过程可以更加高效。此外，数据库不仅能为人工智能提供数据，而且能更好地支持人工智能服务。从建模、训练、重用的角度来看：1）利用统一的 SQL 接口，用户可以使用用户定义的函数或存储过程轻松构建人工智能模型；2）训练人工智能模型需要进行大量的张量计算。通过扩展关系代数，数据库可以更好地支持张量计算，并统一在执行器中执行，有助于模型训练；3）收敛的人工智能模型可以以物化视图、查询表等方式持久化，方便用户重用。但要在数据库上支持人工智能技术，还有一些亟待解决的问题。首先，人工智能和数据库有不同的使用方式。通常，人们用 python 或 R 语言编写人工智能模型，但使用 SQL 语句访问关系数据库中的数据。因此调用人工智能相关的服务有两个问题。1）在编写人工智能的应用程序时，人们需要经常在不同的系统之间切换。2）传统的数据分析师只知道一些 SQL 知识，编写人工智能代码并不轻松。因此，如果我们能够扩展解析规则，使 SQL 同时支持 DB 和 AI，那么提供与 AI 相关的服务将更加方便。其次，人工智能技术多基于张量数据模型，现有数据库还不能很好支持这类数据模型。因此我们需要扩展的关系代数和统一的数据模型来在数据库上支持人工智能服务。

2.3 异构计算框架

随着摩尔定律濒临失败，数据库不能再依靠单处理器来提高处理能力。当今异构计算框架带来了新的潜力。首先，它具备了异构的计算能力（如 GPU、TPU、FPGA）。其次，不同的协处理器能够处理不同的任务，可以大大提高数据库的处理能力。但要支持异构计算框架，数据库需要解决三个主要问题。

- 单一的系统级资源调度。AI 芯片（计算密集型）擅长处理并行的张量计算，而并不适合传统数据库的理论架构和操作（数据密集型操作）。也就是说，结构化数据分析是否可以利用 AI 芯片的高并

行计算能力？此外，为了支持不同的作业（AI+DB）并合理利用计算和存储资源，数据库需要融合异构计算单元。

- 单一的加速器架构。传统的加速器架构只支持 OLAP 类型，对于 OLTP 和 HTAP 工作负载，为了保证系统内存和加速器的本地内存之间的数据一致性，现有加速技术实际效率很低。数据库需要整合新的技术，如 Intel 发布的 CXL 高效的内存访问能力、RDMA 的通讯能力、可编程硬件能力，提升数据库查询处理能力。

- 单一的数据模型。首先，为了提供异构计算，我们需要为不同的操作定义数据模型。例如，关系数据模型依赖关系代数来实现数据管理，但不适合 TPU/NPU 处理模型；张量模型适合于矩阵和迭代计算，但不适合于关系数据处理。此外，我们希望不同的计算能力协同工作。例如，我们需要扩展关系代数来支持张量计算，从而加速多种关系型运算（如联接，聚合等）。因此需要解决，关系数据模型和张量模型是否可以互相转换，是否可以互相受益？是否有统一高效的调度算法来充分发挥异构硬件的能力？

3 AI 原生数据库

本章介绍 AI 原生数据库（轩辕）的设计理念。图 1 中展示了数据库的整体架构。通过将 AI 结合到数据库的处理、运维和组装中，数据库实现了自监控、自配置、自优化、自诊断、自愈、自安全和自组装。并为人工智能和数据库服务提供了统一的调用接口。接下来，我们从五个阶段对该数据库做进一步介绍☞。

3.1 第一阶段：AI建议型数据库

第一阶段是 AI 建议型数据库。它包括一个人工智能引擎，通过自动化建议，提供数据库的离线优化[1,9,13]。这种外挂式的人工智能引擎与数据库松耦合。受到可用资源的限制，AI 建议型数据库主要从四个方面提供辅助优化：

- 负载管理：我们从以下三个方面实现基于人工智能的负载管理。首先，基于人工智能的模型可以用于对工作负载建模。使用独立的特性（例如表、列、谓词等）直接对工作负载建模可能会导致巨大的信息损失，例如不同表之间的引用关系。因此，我们使用编码器-解码器模型来学习用户工作负载的抽象表示，它可以反映基本特性之间的相关性。其次，基于人工智能的模型可以用于工作负载调度。对于混合型工作负载（HTAP），基于 AI 的模型可以估计每个查询所需的资源（如 CPU、RAM、磁盘 IO 等）和运行时间。然后，基于人工智能的模型可以合理调度工作负载，为关键任务分配高优先级，并推迟执行资源开销大的查询。此外，资源调度严重依赖于与资源控制相关的一些参数，如最大并发量等。但这些参数是静态的，需要手动配置。强化学习可以用来学习数据库状态（物理/逻辑）、工作负荷和数据库资源之间的关系，提供一种合理而健壮的调度机制。最后，基于人工智能的模型可以用于工作负载预测。通过负载预测，以更好适应未来工作负载的需求。传统的工作量预测方法多依靠数据库专家根据统计数据进行判断，不能保证很高准确度。相反，用于负载预测的机器学习方法[11]对不同的工作量能够有更好的适应性。
- SQL 优化：我们从以下四个方面优化 SQL 查询。首先，SQL 重写器可以通过重写 SQL 语句提高性能。传统 SQL 重写严重依赖于 DBA 对造成性能瓶颈的查询逐条优化，这在繁重的业务场景下是不现实的。基于人工智能方法，我们可以提供一个 SQL 重写工具，来学习 SQL 书写的原理（例如，避免全表扫描，选择索引列作为连接列，使用表变量等），并进一步优化 SQL 结构。其次，利用人工智能方法对索引推荐进行优化。数据库索引对于提高复杂数据集上的 SQL 查询效率非常重要[4]。

☞ 本文是英文版论文[31]的扩充

然而，传统方法多基于 DBA 的经验构建索引，不能扩展到过大规模，如数千个关系表。而基于人工智能的模型可以学习在给定查询工作量的情况下建立索引的优势和成本，然后根据学习到的信息自动推荐索引。第三，视图推荐可以基于人工智能模型进行优化。给定一组查询，我们首先可以提取等价的子查询，选择高频率的子查询建立物化视图，了解在不同子查询上构建视图的优势和成本。然后，基于人工智能模型可以实现自动物化视图推荐。第四，数据分区推荐也可以基于人工智能模型进行优化。传统分区是由 DBA 基于一些主键或手动指定的键生成。然而，这些方法可能无法找到最佳的分区策略。相反，基于人工智能的模型可以动态的学习分区的可能分布情况，估计在不同工作负载的好处，并推荐高质量的分区策略。

- 数据库监控：我们通过监控数据库状态，调整数据库配置，避免数据库故障或及时进行故障转移。首先，对于数据统计信息收集，它自动监视数据属性（例如，访问频率、修改情况等）以及数据如何在数据库集群中分布。其次，对于系统统计，它自动监视数据库系统的状态（例如，每秒批处理请求的数量、用户连接的数量、网络收发器的效率）。然后利用机器学习算法对这些指标进行分析，调整系统参数，并对异常事件进行预警。其次，对于工作负载统计，它监视用户工作负载的性能，并分析工作负载的变化情况。利用工作负载特性，数据库预测未来的状态并相应地调整自身状态。
- 数据库安全：我们将安全和加密技术与人工智能相结合，提高数据库的安全性。首先，自主隐藏（autonomous masking）旨在隐藏身份证号等隐私数据。自主隐藏技术根据历史数据和用户访问模式智能地选择要屏蔽的数据列。其次，自主审计（autonomous auditing）从数据预处理和动态分析两个方面对审计进行优化。传统审计技术往往要求审计人员提前了解大量的业务数据，这是一种非常浪费人力精力的行为。自主审计通过对数据预处理，节省了人力成本，并通过从海量数据中总结有用的信息，帮助审计人员做出更好的决策。第三，自主访问控制（autonomous access control）可以自动检测系统漏洞。现有的自主检测方法主要是通过安全扫描进行检索，而未知的安全漏洞需要做大量的未知检索和测试工作。基于人工智能的模型可以快速定位安全漏洞[20]。我们不仅可以检测漏洞数据库中已知的漏洞，还可以预测和评估潜在的漏洞。

3.2 第二阶段：AI辅助型数据库（AI-Assisted）

第二阶段是 AI 辅助型数据库。它将 AI 引擎集成到数据库内核中，提供运行时优化。AI 工具(如调优模型、工作负载调度、视图推荐)可以合并到相应的数据库组件[10]中。通过这种方式，人工智能被集成到数据库的工作过程中。例如，将调优模型嵌入到查询优化器，每次生成查询计划时，我们首先进行查询调优，再生成、执行查询计划。这样，通过调优用户级参数，我们可以更好地自适应要处理查询的特性。人工智能辅助数据库的优势在于：1）提供更细粒度的优化；2）通过在内核中嵌入人工智能引擎，可以降低如通信成本等多方面的开销。

此外，通过内置 AI 引擎，AI 辅助型数据库可以提供自配置、自优化、自监控、自诊断、自愈和自安全服务。

- 自配置（Self-configuring）：数据库可以自动调整自己的配置，以适应工作负载和环境的变化。首先，自动配置工作负载。数据库可以并行地执行不同粒度的查询，不同粒度对系统资源和性能都有不同的要求。数据库可以根据工作负载特性为工作负载配置系统参数，而这些特性是从第一级中工作负载的建模、调度和预测中得到的。其次，数据库通过配置调优、软件打补丁、软件升级等机制进行数据库管理。例如，所有数据库都有数百个参数，对数据库维护的各个方面都至关重要。然而，这些配置需要手动调优，这不仅费时，而且无法找到最佳配置。基于人工智能的方法，如深度强化学习，可以自动进行数据库调优[9]。
- 自优化（Self-optimizing）：首先，我们设计了一个基于学习的查询优化器[28]，主要针对代价/基数估计、连接顺序选择和数据结构设计三个方面：1）代价/基数估计对于查询计划选择至关重要。然

而，数据库主要根据原始统计数据(例如读/写块、后端、死锁)估计基数，而使用直方图估计每个查询操作符的行数(例如散列连接、聚合、过滤器)则比较差。基于人工智能的方法，如 Tree-LSTM[32]，可以深入学习数据分布，提供更准确的成本/基数估计。2）AI 技术可以优化连接顺序的选择。不同的连接方案对查询性能有很大的影响，而寻找最佳计划是 NP-hard 问题。在静态算法(如动态规划、启发式算法)中，数据库中连接顺序选择的性能受查询估计的影响。基于人工智能的方法[33]以一步连接作为短期奖励，以执行时间作为长期奖励，可以更好地选择不同的连接顺序计划。3）内置的 AI 模型也可以在线推荐数据分区、索引、视图。其次，我们可以利用基于学习的模型来优化执行器引擎和数据存储。对于执行器引擎，我们考虑两个方面。1）提供基于行和基于列的混合查询执行方法。在这里，AI 可以服务于不同的数据应用程序(例如，基于行的 OLTP，基于列的 OLAP)。2）我们提供张量处理引擎来执行 AI 模型。为了增强人工智能技术来优化数据库组件，数据库可以使用矢量化引擎来原生执行人工智能模型。

- 自监控（Self-monitoring）：数据库可以自动监控数据库状态（如读/写块、并发状态、工作事务）。检测操作规则，如根本原因分析规则。它可以监视整个生命周期中的数据库状态（例如，数据一致性、数据库运行状况）。监控得到的信息（如数据统计信息，系统统计信息，负载统计信息等）可用在自配置、自优化、自诊断和自修复当中。其中，我们需要通过增量式的更新监控信息来最小化自监控的开销。此外，通过总结故障转移条件和采用的解决方案，数据库可以将经验以运维说明书的形式发布出来，帮助人们更好的理解和监督。数据库一般有数百个状态信息（KPI），如果没有很好的采集机制，可能会影响数据库性能，因此如何监控这些信息而不影响数据库性能就是研究的难点。
- 自诊断（Self-dignosis）：高可用性是数据库系统的另一个重要需求。数据库需要保护每个事务的健康状态，比如等待时间和分配的资源。如果没有动态保护机制，可能会影响数据库性能。自诊断包括一套诊断和纠正数据库中异常情况的策略，这些异常情况主要由硬件错误（如 I/O 错误、CPU 错误）和软件错误（如错误、异常）引起。即使某些数据库节点意外工作，自诊断也可以保证服务。例如，在数据访问错误、内存溢出或违反某些完整性限制的情况下，数据库可以自动检测根本原因，及时取消相应的事务。
- 自愈（Self-healing）：高可靠性是数据库系统的一个关键需求。但是导致数据库崩溃的原因有很多(例如，性能退化，硬件/软件故障转移)，保持数据库运行是一件很困难的事情。传统数据库的尝试包括两个方面。第一，设置基于规则的监视器，并在触发阈值时发出警报。但是这些限制是粗粒度的，不能有效地预防意外事故。第二，定期备份数据。虽然这有助于数据恢复，但这种方法并不能保证数据库的可靠性。基于人工智能的方法，数据库能够自动监控、检测运行时问题并迅速恢复。首先，它能够智能的将不同作业隔离开，防止一个作业出现错误时不会影响到其他用户。其次，它使用 AI 建议型数据库（第一阶段）的工具来进一步帮助节约时间，并将人从重复的故障恢复循环中解放出来。此外，基于 CNN 等模型，它能自动检测查询的异常（如耗时远超过优化器预估时间），并及时优化执行过程。例如，对于分布式数据库，当节点崩溃或负载倾斜时，它支持以最基于较低的开销来实现实时数据迁移。第三，我们可以自组装技术来提供多路径计算（类似于智能导航系统），避开错误路径选择可用路径，提升系统可靠性。
- 自安全（Self-security）：数据库自安全包括一下几个功能。第一，根据数据的特点，智能隐藏隐私数据，例如身份证号等。数据隐藏技术主要包括插入和替换两种方法。插入方法通过在要隐藏数据前加一段冗余数据实现数据隐藏，接收方需要根据特殊标识的位置信息找到隐藏的数据。而替换方法通过字节替换或改变字节顺序实现数据隐藏，接收方需要根据事先协商好的规则获得隐藏的数据。现在有很多数据发现工具，能够根据数据异常解析出隐藏的数据。归结原因还是隐藏算法多使用简单的数据变化。智能隐藏技术则利用神经网络内部高维的非线性数据变换机制帮助提

高数据隐藏的效果，比如在斯坦福大学与谷歌的研究项目中，就曾将卫星图片转换成不易被察觉的高频信号。第二，从数据预处理和动态分析两个方面优化审计工作。首先，传统审计工作往往需要审计人员获取大量业务数据，在对这些数据特征有了较深理解之后，才能真正开展审计工作。智能审计将提取信息的工作交给机器学习模型，不仅能节约人力开销，而且可以帮助审计人员更好的决策。其次，智能审计还要求机器学习模型能够根据用户行为来智能防范用户对数据的窃取，基于时间序列预先防范越权访问、修改等非法行为。第三，对重要数据实现透明加密，即重要数据在整个生命周期（包括存储、内存和 CPU 等）始终是加密的，对于第三方来说是不可读的。第四，数据要防篡改，防止恶意修改数据。第五，为了获取数据的访问历史，数据访问记录应该是可跟踪的。此外，基于人工智能的模型可以自动学习攻击规则，及时阻止未经授权的访问。

3.3 第三阶段： AI增强型数据库（AI-Enhanced）

第三阶段是 AI 增强型数据库，不仅用人工智能技术优化数据库设计，而且提供基于数据库内置的 AI 原生服务。

3.3.1 基于学习的数据库组件

大多数数据库核心组件都是由人根据经验或经典规则、算法设计的，如配置优化、查询优化、成本估计、索引等。然而，我们发现许多数据库组件其实可以利用人工智能算法进行增强，提供传统经验技术之外的替代策略。

- 配置优化：数据库中一般都有上百个可调参数。参数之间、参数与数据库性能之间存在着复杂的关系。数据库配置优化已经被证明是一个 NP-hard 问题[30]。传统数据库调优方法存在三个主要痛点：1）基于 DBA 手动调优，耗时耗力，很难在有限时间内达到较好效果；2）启发式的调优方法很难保证稳定的调优表现，对于数据库这种对可靠性要求很高的系统是不能忍受的[30]；3）传统机器学习模型较为简单，需要基于管道式（pipeline）架构先进性特征过滤，会造成大量信息丢失[1]。因此我们采用深度强化学习（DRL）的方法进行配置优化。第一步，我们对用户负载进行编码，提取负载特征。第二步，我们利用 DRL 模型，根据负载特征、数据库状态生成观察值，模型根据观察值给出推荐配置。第三步，模型根据反馈优化推荐策略。利用深度学习强大的映射能力和强化学习高效的优化策略，这种方法可以更加有效地解决数据库配置优化问题，并适应不同的负载实现动态调优。
- 基数估计：准确的基数估计对于查询计划的选择非常重要。传统数据库主要基于直方图等进行基数估计，误差会达到几个数量级[29]。所以我们可以进一步通过机器学习技术对查询计划建模，基于 Tree-LSTM 模型提供更加准确高效的基数估计模型。整个基数估计模型可以分成三个部分。首先，编码部分将一个查询计划树上的所有节点进行编码，包括算子、谓词、关系表等等。然后，我们基于深度优先等原则将经过编码的查询计划树转化成一个特征序列。最后，基于 Tree-LSTM 模型，我们评估每个经过序列化的查询计划的基数。这种方式能帮助我们更加高效准确的进行查询基数估计。首先，它在查询编码中同时考虑了查询和物理执行计划的特征，能够提高模型对不同负载的适应能力。其次，它提供了一种端到端的基数估计方法，避免了抽样等方法中样本退化的问题。
- 索引推荐：索引推荐可以从两个方面考虑。其一，学习型索引。索引对于提升复杂数据集上检索任务的效率有着非常重要的意义。传统或非人工智能索引方法，如位图索引，基于树的索引等，在数量，速度，变化，变异性，价值和复杂性方面能够满足需求，但它们无法对“未知”用户行为和大数据有效检测。此外，数据量的增长也会导致索引大小激增。我们可以从范围索引、点索引、布隆索引等多个方面设计学习型索引，通过预测给定排序数组内键的位置高效的学习累积分布函数（CDF），使用学习型索引代替传统的 B+树等结构。其二，智能推荐索引。在数据库系统中，在一个数据库的多张表上建立索引存在很多种索引方案，每种方案带来的收益以及对其进行维护

的代价差别很大，因此，选在一个最优的索引方案成为一个重要问题。我们可以采用一种使用学习分类器与强化学习和基因算法相结合的索引调整方法。它使用学习分类器系统创建和更新由规则表示的知识，并生成覆盖数据库中绝大多数情况的调整索引来调整性能。在典型的学习分类器系统中，将每个分类器视为一个个体，并将其中适应度较低的分类器替换为适应度较高的分类器。每个分类器又一组推理规则表示，包括先行词和结果两部分，遵循"*如果<条件>-那么<动作>*"的格式。每个个体（分类器）的染色体上有 16 个基因编码 16 个先行词（条件），1 个基因编码结果（动作）。条件表示环境的一些特征，如数据表中是否存在索引等；动作表示对索引执行的操作，如创建，删除等。每个分类器的强度用来表示在环境中的适应度，强度越高的越有可能被使用并且将其优势保持下去。这种方法能够高效的为不同的场景推荐合适的索引，并有效的提高了模型适应不同环境的能力。

- 连接顺序选择：不同的连接顺序对于查询的执行效率影响很大。而一个查询计划的状态空间相对于表的连接数目是呈指数级的，基于动态规划或启发式算法的传统连接顺序选择方法很难在有限时间内有足够好的表现。因此我们利用强化学习算法，把连接顺序选择问题抽象成一种强化学习过程。比如，对于一棵左深的连接树（（A ⋈B）⋈C）⋈D，我们可以用一个序列 A-B-C-D 来表示每一位代表了最小单元的连接选择。整个强化学习过程就是学习基于基础表状态（{A, B, C, D}），迭代生成完整的优化连接树（Join Tree）序列。每次模型获得当前的序列信息，然后选择下次的两表连接行为。每一步行为的代价作为奖励（Reward），我们的目标就是求解行为选择的策略函数，即针对每个状态下如何采取行为。由于 Reward 只是当前行为的短期回报，我们再通过 Bellman 函数计算长时奖励来求解：一个长时奖励表示从一个状态到终止状态每一步 Reward 的总和，它表明了一个状态的全局的一个估计值。策略函数就通过最优化每一步的长时奖励得到。一旦我们的策略函数能够最小化长时奖励，根据策略函数我们就能得到最小代价的连接计划。基于强化学习的连接顺序选择方法有三点好处。首先，强化学习不需要非常多原始样本。对每个样本它能迭代的生成多个经验样本<当前状态，行为，下一状态，反馈值>，用于优化策略函数。其次，它不过度依赖代价估计器的准确性。传统方法都要基于代价估计器给出代价估计值评估一个计划的好坏。而这种方法基于计划实际执行的时延和长期回报进行策略更新，与代价估计器更好的解耦。此外，得益于神经网络强大的学习能力和泛化能力，结合了神经网络的深度强化学习模型可以帮助实现更好更稳定的表现。
- 端到端的查询优化器：普通查询优化器依赖于以上基数估计、索引推荐、连接顺序选择等多方面的综合性能。其实我们可以设计一个端到端的查询优化器，首先，基数估计、索引推荐、连接顺序选择等各个部件都基于不同的人工智能模型，可以动态高效的解决各自微调。其次，基数估计、索引推荐、连接顺序选择等各个部件都共享一个相同的优化目标，每次生成完整的查询计划后，根据反馈值统一更新。这种方式可以有效避免反馈在管道式传递过程中可能的衰减很损耗。

3.3.2 数据库内置的 AI 服务

虽然人工智能可以解决许多现实问题，但由于人工智能很难被普通用户使用，还没有广泛部署的人工智能系统可以应用于各个领域。因此，我们可以借助数据库技术，来降低使用人工智能的门槛。首先，SQL 易于使用并已经被广泛接受。我们可以通过扩展 SQL 来支持 AI。其次，我们可以利用数据库优化技术来加速 AI 算法，例如索引、增量计算和共享计算。

我们将数据库中支持人工智能功能的技术分为五个层次。

- AI 模型作为用户定义函数(UDF)：我们在数据库中嵌入 AI 框架(如 MADlib、TensorFlow、Scikit-learn)，并为每种算法提供用户定义函数(UDF)。然后用户可以按照 SQL 原有的语法或内嵌其他语言自定义 AI 模型，从数据库调用这些实例来使用 AI 模型。
- AI 模型作为视图：如果用户想在第一级（AI as UDF）中使用 AI 算法(例如随机森林)，用户需要先

训练模型，然后使用模型。在第二级中，我们把 AI 算法看做一个视图。如果一个用户首先调用使用 AI 算法，数据库可以将训练出来的模型物化下来（materialized view），之后其他用户就可以直接使用这个模型。AI 模型也可以通过增量训练进行更新（fine-tuning）。

- 模型无关 AI：在第一、二级 AI 服务中，用户必须指定调用的 AI 算法（例如，用于聚类的 k-means）。实际上，用户可能只知道应该解决哪些问题，例如回归或分类，但不知道应该具体使用哪些算法。通过自动根据问题选择算法，数据库可以自动推荐最适合用户场景的算法。
- 问题无关 AI：很多时候用户甚至不能解释清楚需要解决的问题，例如具体的分类标准。在给定数据库的情况下，全自动 AI 可以自动发现哪些问题可以由人工智能算法来解决，并推荐合适的人工智能算法。
- 全自动 AI：该系统最终可以自动发现人工智能的应用机会。包括自动发现问题，选择合适的 AI 模型、算法、数据和训练方法等。

3.3.3 AI 与 DB 的混合引擎

上述方法仍然使用两个引擎：AI 引擎和 DB 引擎。我们需要设计一个混合型引擎同时支持人工智能和数据库服务。给定一个查询，SQL 解析器解析 SQL 查询，生成通用的查询计划。基于计划中的算子类型，优化器决定是处理数据还是使用 AI 模型。如果是为了处理数据，查询计划发送给相应的数据库执行器；否则，发送给相应的 AI 执行器。此外，它也要求设计新的模型既支持关系代数模型，又支持张量模型，这样，我们就可以利用统一的数据模型来同时支持人工智能和数据库服务。

3.4 第四阶段：AI自组装型数据库（AI-Assembled）

第四个阶段是 AI 自组装型数据库，它不仅自动地组装数据库组件来生成最适合给定场景的数据库，而且还将不同任务调度到合适的硬件上。

3.4.1 数据库自组装

首先，每个数据库组件都有多个选项。例如，优化器包括基于代价的模型（cost-based）、基于规则的模型（rule-based）和基于学习的模型（learning-based）。我们可以根据用户的需求来选择最好的组件。其中，相同组件的不同变体应该采用相同的标准接口，以便可以组装组件。因此，我们提出了一个自组装模块。对于不同的场景，我们可以在每个服务层中动态地选择适当的组件，并组合适当的执行路径。执行路径可以看作是自然语言序列(NLS)，比如<pg_parser、optimizer_RBO、row-based executor、accelerator>，其中每个位置只有离散的选项。因此，问题是如何在查询级别生成离散的序列。一种可能的方法是使用增强学习算法（RL）。它以整个路径序列为一个 epoch，以一个动作为一个 episode。在每个 eposide，强化学习选择执行查询的下一个组件（action）。在这个问题中，动作是离散的，所以可以使用 DDQN 算法[16]。与其他离散的 RL 算法相比，DDQN 通过选择解耦动作并计算目标 Q 值，消除了估计与最优解存在较大偏差的问题。

然而，基于 RL 的选路算法存在两个问题。首先，直到最后一个节点生成，Q 网络才会对整个路径打分。它对中间节点的选择不敏感。然而，就整个道路而言，有必要对当前和未来的影响作出全面的评价。其次，以 eposide 为单位进行训练时，将路径的每个节点分散在一个训练样本中，而不是作为一个整体来计算梯度。生成式对抗网络(GAN)[19]较好地解决了上面端到端的路径选择问题。我们使用 G 网络根据工作负载、数据库状态和组件特性生成路径向量，并使用 D 网络作为性能评估模型。但传统的 GAN 网络模型并不能完全解决这一问题。由于它主要用于生成取值范围连续的数据，而很难生成有离散特征的路径向量。这是因为 G 网络需要基于梯度下降和回归期望进行微调。但是当数据是离散的特征时，微调通常是没有意义的。因此我们选择将 RL 算法与 GAN[17]相结合。首先，在 RL 算法中使用 G 网络作为代理（agent）；action 是下一个服务节点；状态不仅包括查询状态、数据库状态，还包括生成的节点信息。每个迭代生成一个完整的路径。其次，与传统的 RL 算法不同，每个动作都由 D 网络进行评分，指导整个路径序列的生成。

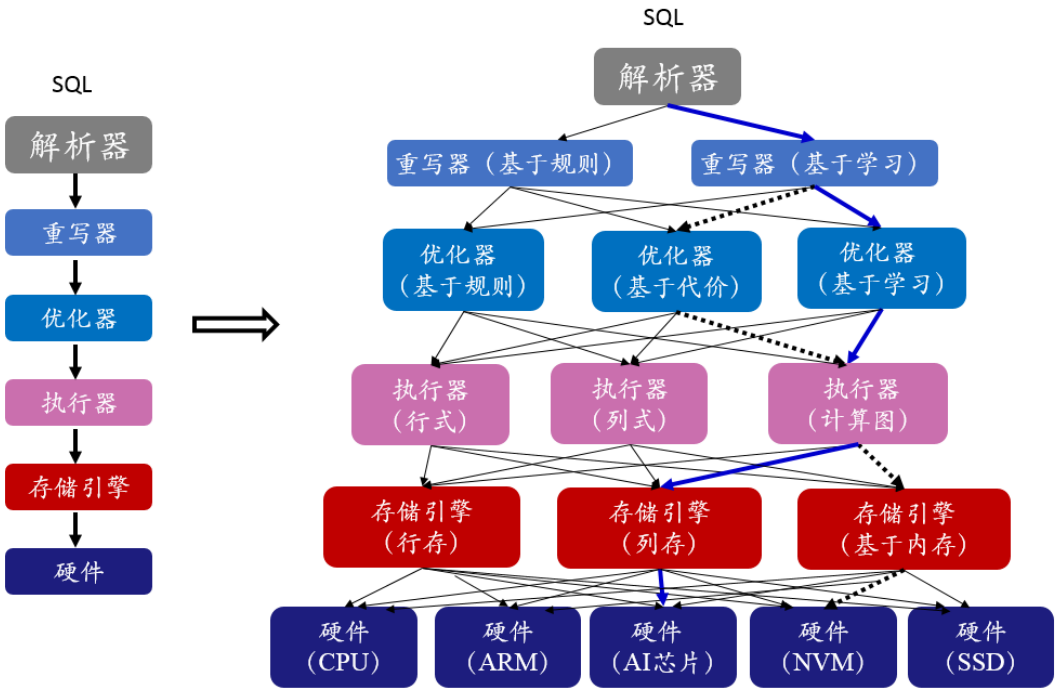

Fig.2 AI-Assembled Database

图 2 AI 组织型数据库

3.4.2 异构计算架构

我们需要充分利用多样化的计算能力。数据库和人工智能技术通常需要不同的计算能力和硬件。对于数据库，传统的优化器使用 CPU 处理查询。而人工智能技术则需要新的 AI 芯片来支持并行处理(如 GPU，NPU)和自调度。现在许多应用程序都需要同时使用 DB 和 AI 技术，尤其是在大型数据分析场景中。因此，我们需要支持多个模型，如关系模型、图模型、流模型、矩阵模型、向量模型和张量模型。我们可以自动选择应该使用哪些模型。我们还需要能够根据数据模型切换计算能力。例如，对于优化器中的调优模块，在训练调优模型时，我们使用 AI 芯片将训练数据提取到内存中，然后使用 NPU 进行反向传播(训练神经网络)。对于连接和过滤操作，我们仍然可以使用传统的硬件。我们还需要研究一个模型(如关系模型)是否可以转化为其他模型(如张量模型)。最终目标是充分利用 x86、ARM、GPU、NPU、加速器等多种计算能力。此外，除了新型计算算力还有存储算力（NVM）和网络算力（RDMA）。我们可以通过存储算力和网络算力来重构数据库架构。例如通过 NVM 来设计 Record-level 的存储模型来取代 Page-level 的引擎，发挥 NVM 读写不均衡的特性；通过 RDMA 来设计共享内存架构，通过可编程 RDMA 来进行网卡上数据过滤，提升计算效率，从而改变数据库计算机制。

我们的目标是充分利用异构计算算力和特点，发挥它们各自优势，构建一个面向数据处理和分析的完整计算生态系统，通知支持数据库和 AI 的典型应用。

3.5 第五阶段：AI自设计型数据库（AI-designed）

第五阶段是 AI 自设计型数据库。在这个阶段，数据库完全由人工智能设计，包括设计、编码、评估、监控和维护等各个阶段。我们将人工智能技术集成到整个数据库生命周期中，使数据库和人工智能都能获得最佳性能。

表 1 AI 原生数据库的五个阶段

阶段	特点	简介	举例
1	AI 建议型数据库	提供插件形式的 AI 引擎	○ 负载管理（例如，负载调度） ○ SQL 优化（例如，SQL 重写器、索引推荐） ○ 数据库监视器（例如线下参数调优、系统统计） ○ 数据库安全性（例如，自动审计/屏蔽）
2	AI 辅助型数据库	提供基于数据库的 AI 引擎	○ 自配置（例如，在线参数调优） ○ 自优化（例如，SQL 优化、数据存储） ○ 自监测（例如，监控系统状态） ○ 自诊断（例如，发现硬件/软件问题） ○ 自愈（例如，故障恢复，在线迁移） ○ 自安全（例如，可回溯性、防信息泄漏）
3	AI 强化型数据库	提供统一的 AI、DB 引擎	○ 基于学习的数据库组件 ● 学习型查询重写器 ● 学习型代价估算器 ● 学习型优化器 ● 学习型执行器 ● 学习型存储引擎 ● 学习型索引 ○ 声明型人工智能（UDF；视图；模型无关；问题无关；全自动）
4	AI 自组装型数据库	提供异构处理架构	○ 自组装 ○ 充分利用异构硬件（如 ARM、GPU、NPU）
5	AI 自设计型数据库	基于 AI 的数据库生命周期	设计，编码，评估、监控和运维

4 挑战和机遇

前面一章我们介绍了如何通过五个阶段来设计一个以支持数据管理、数据分析和机器学习为目标的人工智能原生数据库。本章中我们将会进一步给出 AI 原生数据库在设计、开发、部署、运维等多个方面带来的挑战和机遇。

4.1 从One-Size-doesn’t-fit-all到one-stack-fits-all

Michael Stonebraker 认为，由于应用程序的多样性(例如 OLTP、OLAP、stream、graph)和硬件的多样性(例如 CPU、ARM、GPU、FPGA、NVM)，一种数据库并不能适合所有的情况（one-size-doesn’t-fit-all）。数据库架构师通常根据已有规则和经验，结合不同的技术变体来设计数据库架构。尽管数据库组件及其变体是有限的，但是这些组件组装成数据库的可能组合数量是巨大的。因此，传统设计的数据库很可能不是最优的，它们通常会陷入局部最优。它要求使用能够适应不同场景的 AI 技术，自动设计数据库。

但是，我们认为通过构建一个智能的数据库栈，可以适应所有的情况（one-stack-fits-all）。其基本思想是首先实现数据库每层处理的可能组件，例如索引、优化器、存储（每个组件都有多个选项）。然后使用 AI 技术智能的组装这些组件，形成一些数据库候选组件，最后动态的根据给定场景生成最适合的数据库。通过

这种方式，我们可以自动验证不同的数据库(即，探索更多可能的数据库设计，并可以设计更强大的数据库。这类似于 AlphaGO，基于学习的方法打败了人类，因为机器可以探索更多未知空间。

“One-stack-fits-all”有几个挑战。第一，每个组件应该提供标准接口，以便不同的组件可以集成在一起。第二每个组件应该有不同的变体或实现，例如，不同的索引类型，不同的优化器。第三，它需要一个基于学习的组件来组装不同的组件。第四，在部署数据库之前，需要对所装配的数据库进行评估和验证。第五，支持异构的计算框架。不同组件可能需要运行在不同的硬件上，例如，学习优化器应该运行在 AI 芯片上，传统的基于成本的优化器应该运行在通用芯片上。它需要有效的硬件调度算法来安排不同任务。第六，传统芯片设计有 EDA 等软件辅助，但是软件设计并没有一个类似的工具来评价设计效果，因此需要设计类似软件来对数据库的设计给出评估。

4.2 下一代分析型处理：OLAP 2.0

传统的 OLAP 侧重于关系型数据分析。然而，在大数据时代，出现了许多新的数据类型，如图数据、时间序列数据、空间数据、文本数据、图像数据，需要新的数据分析技术来分析这些多模型数据。此外，除了传统的聚合查询，许多应用程序还需要使用机器学习算法来增强数据分析，例如图像分析。因此，集成 AI 和 DB 技术来提供新的数据分析功能是很挑战性的。我们认为多模型数据的 DB 和 AI 混合在线分析处理应该是下一代 OLAP，即 OLAP 2.0。

支持 OLAP 2.0 有几个挑战。首先，不同的数据类型使用不同的模型，如关系模型、图模型、KV 模型、张量模型，需要一个新的模型来支持多模数据分析。其次，OLAP 2.0 查询可能涉及数据库和人工智能操作，它需要设计新的模型来优化这些跨硬件的异构操作。

4.3 下一代事务处理:OLTP 2.0

传统的 OLTP 主要使用通用硬件，如 CPU、RAM 和磁盘，但不能充分利用新硬件，如 AI 芯片、RDMA 和 NVM。实际上，我们可以利用新的硬件来改进事务处理。首先，根据 NVM 的非易失性、读写速度不对称和磨损均衡等特点，需要重新考虑数据库体系结构。例如，我们可以使用 NVM 替换 RAM，并使用 NVM 上的记录级存储替换页级存储。其次，我们可以利用 RDMA 来改进数据库中的数据传输。我们可以利用智能以太网卡的可编程特性，实现对 RDMA 的过滤，避免在 RAM 和 CPU 中进行不必要的处理。第三，现在已经有了一些专门人工智能设计的 AI 芯片，设计专门为数据库定义的硬件芯片也是很有前景。

支持 OLTP 2.0 有几个挑战。首先，充分利用新硬件设计新一代数据库需要集成多种数据模型和调度侧率。其次，评估和验证新硬件是否能使数据库体系结构受益也是一件很难的事情。

4.4 AI4DB

在数据库中嵌入人工智能功能有如下几个挑战。

- 训练样本：大多数人工智能模型都需要大规模、高质量、多样化的训练数据来实现良好的性能。然而，在数据库中获取训练数据相当困难，因为这些数据要么对安全至关重要，要么依赖于 DBA。例如，在数据库调优中，需要根据 DBA 的经验获取训练样本。因此很难得到大量的训练样本。此外，训练数据应该涵盖不同的场景、硬件环境和工作负载。
- 模型的选择：机器学习算法有很多，很难针对不同的场景自动选择合适的算法。此外，模型的选择受到许多因素的影响，如数据质量、训练时间等。例如，深度学习可能是代价估计的最佳选择，而强化学习可能是连接顺序选择的最佳选择。训练时间可能也很重要，因为有些应用程序的性能很关键，不能忍受长时间的训练。
- 模型收敛：模型能否收敛是一个非常重要的问题。如果模型不能聚合，我们需要提供其他方法来避免做出错误的决策。例如，在调优中，如果模型不收敛，我们就不能利用模型对参数进行建议。
- 适应性：模型应该适应不同的场景。例如，如果硬件环境发生变化，模型可以适应新的硬件。

- 泛化能力：模型应该适应不同的设置。例如，如果工作负载发生了更改，那么模型应该支持新的工作负载。如果更新了数据，模型需要有能力适应新的数据。

4.5 DB4AI

首先，我们可以使用索引技术加速人工智能算法。大部分的研究都集中在人工智能算法的有效性上，而忽略了效率，而这也是非常重要的。它要求利用数据库技术来提高人工智能算法的性能。例如，自动驾驶汽车需要大量的训练样本，这是相当费时的。实际上，它只需要一些重要的训练数据，比如晚上或者下雨天的训练用例，而不是很多冗余的训练用例。因此，我们可以对样本和特征建立索引，利用索引来进行高效节能的训练。

其次，AI 原生数据库要有能力理解需求、发现模型。普通用户可能只知道他们的需求，例如，使用一个分类算法来解决一个问题，但不知道应该使用哪个 AI 算法。因此，自动发现人工智能算法非常重要。此外，不同用户重用收敛的人工智能模型也非常具有挑战性。

4.6 边缘计算数据库

大多数数据库设计目标都是部署在服务器上。而随着 5G 和物联网设备的发展，我们需要在小型设备中嵌入微型数据库。设计这样一个“数据库”有几个挑战。首先，保护数据的安全性。其次，实时数据处理能力。小型设备具有较低的计算能力，在这样的小型设备上提供高性能是相当具有挑战性的。再有，不同设备之间的数据迁移。有些设备的存储空间很小，在不同设备之间迁移数据非常困难。最后，实时控制是 5G、IOT 的重要需求，需要研究实时控制技术。

5 结论

我们提出了一种人工智能原生数据库（轩辕数据库）。它不仅利用人工智能技术实现了自配置、自优化、自监控、自诊断、自愈、自安全和自组装服务，而且还提供了数据库内置的人工智能服务，为数据库和 AI 应用提供统一的访问接口，降低了使用人工智能的门槛。我们将 AI-native 数据库分为五个层次，AI 建议型数据、AI 辅助型数据库、AI 增强型数据库、AI 自组装型数据库和 AI 自设计型数据库。我们还讨论了未来研究的挑战，并给出了在人工智能原生数据库设计中的机遇。

References:

[1] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *SIGMOD*, pages 1009–1024, 2017.

[2] C. J. Date. *An introduction to database systems (7. ed.)*. Addison-Wesley-Longman, 2000.

[3] G. Figueiredo, V. Braganholo, and M. Mattoso. Processing queries over distributed XML databases. *JIDM*, 1(3):455–470, 2010.

[4] A. Gani, A. Siddiqa, S. Shamshirband, and F. H. Nasaruddin. A survey on indexing techniques for big data: taxonomy and performance evaluation. *Knowl. Inf. Syst.*, 46(2):241–284, 2016.

[5] S. Idreos, N. Dayan, W. Qin, M. Akmanalp, S. Hilgard, A. Ross, J. Lennon, V. Jain, H. Gupta, D. Li, and Z. Zhu. Design continuums and the path toward self-designing key-value stores that know and learn. In *CIDR*, 2019.

[6] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*, 2019.

[7] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, pages 489–504, 2018.

[8] S. Krishnan, Z. Yang, K. Goldberg, J. M. Hellerstein, and I. Stoica. Learning to optimize join queries with deep reinforcement learning. *CoRR*, abs/1808.03196, 2018.

[9] G. Li, X. Zhou, B. Gao, and S. Li. Qtune: A query-aware database tuning system with deep reinforcement learning. In *VLDB*, 2019.

[10] X. Liang, A. J. Elmore, and S. Krishnan. Opportunistic view materialization with deep reinforcement learning. *CoRR*, abs/1903.01363, 2019.

[11] L. Ma, D. V. Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 631–645, 2018.

[12] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, pages 3:1–3:4, 2018.

[13] M. Mitzenmacher. A model for learned bloom filters and related structures. *CoRR*, abs/1802.00884, 2018.

[14] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning state representations for query optimization with deep reinforcement learning. In *SIGMOD*, pages 4:1–4:4, 2018.

[15] W. G. Pedrozo, J. C. Nievola, and D. C. Ribeiro. An adaptive approach for index tuning with learning classifier systems on hybrid storage environments. In *HAIS*, pages 716–729, 2018.

[16] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[17] W. Wang, G. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K. Tan, and S. Wang. SINGA: putting deep learning in the hands of multimedia users. In *SIGMM*, pages 25–34, 2015.

[18] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K. Tan. Database meets deep learning: Challenges and opportunities. *SIGMOD Record*, 45(2):17–22, 2016.

[19] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[20] J. Zhang, Y. Liu, K. Zhou, and G. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In SIGMOD, 2019.

[21] Z. Zhou, J. Liang, Y. Song, L. Yu, H. Wang, W. Zhang, Y. Yu, and Z. Zhang. Lipschitz generative adversarial nets. CoRR, abs/1902.05687, 2019.

[22] S. Zong, A. Ritter, G. Mueller, and E. Wright. Analyzing the perceived severity of cybersecurity threats reported on social media. CoRR, abs/1902.10680, 2019.

[23] V.Leis, A.Gubichev, A.Mirchev, P.Boncz, A.Kemper, and T.Neumann. How good are query optimizers really? PVLDB, 9(3) 2015.

[24] V.Leis, B.Radke, A.Gubichev, A.Kemper, and T.Neumann. Cardinality estimation done right: Index-based join sampling. CIDR, 2017.

[25] F.Giroire, Order statistics and estimating cardinalities of massive data sets.Discrete Applied Mathematics, 157(2):406–427, 2009.

[26] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. ACM Transactions on Database Systems (TODS), 15(2):208–229, 1990.

[27] M. Durand and P. Flajolet. LogLog counting of large cardinalities. In European Symposium on Algorithms, pages 605–617. Springer, 2003.

[28] P. Flajolet, E´. Fusy, O. Gandouet, and F. Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. Discrete Mathematics and Theoretical Computer Science (DMTCS) Proceedings, AH(1):127–146, 2008.

[29] Xudong Lin, Xiaoning Zeng, Xiaowei Pu, Yanyan Sun: A Cardinality Estimation Approach Based on Two Level Histograms. J. Inf. Sci. Eng. 31(5): 1733-1756.

[30] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In SoCC, pages 338{350, 2017.

[31] G. Li, X. Zhou, S. Li. XuanYuan: An AI-Native Database. IEEE Data Eng. Bull. 42(2): 70-81 (2019).

[32] Ji Sun, G. Li: An End-to-End Learning-based Cost Estimator//Proceedings of the International Conference on Very Large Data Bases. Tokyo, Japan, 2020.

[33] X. Yu, G. Li, C. Chai, N. Tang. Reinforcement Learning with Tree-LSTM for Join Order Selection//Proceedings of the IEEE International Conference on Data Engineering, Dallas, USA, 2020.