Vextra：面向异构向量数据库系统的统一中间件抽象

Chandan Suri 纽约，美国
cs4090@columbia.edu

Gursifath Bhasin 纽约，美国
gb2760@columbia.edu

arXiv:2601.06727v1 [cs.DB] 11 Jan 2026

摘要—向量搜索迅速集成到人工智能应用中，尤其是在检索增强生成（RAG）领域，催生了由各种专业向量数据库组成的多样化生态系统。尽管这一创新提供了丰富的功能选择和性能特征，但同时也带来了一个重大挑战：严重的 API 碎片化。开发者面对的是由各不相同、专有且往往变化频繁的 API 契约构成的复杂环境，这阻碍了应用程序可移植性，增加了维护开销，并导致供应商锁定。本文介绍了 Vextra，这是一种旨在解决这一碎片化问题的新型中间件抽象层。Vextra 为核心数据库操作提供统一的高级 API，包括数据插入或更新、相似性搜索和元数据过滤。它采用可插拔适配器架构，将这些统一 API 调用转换为各种后端数据库的原生协议。我们认为，这种抽象层是推动向量数据库生态系统走向成熟、促进互操作性以及实现更高层次查询优化的关键一步，同时仅带来极小的性能开销。

一、引言

A. 现代人工智能中向量数据库的寒武纪大爆发

大语言模型（LLM）和生成式人工智能的激增，从根本上重塑了数据管理的格局。在这一全新架构范式中，向量数据库是一个关键组成部分，它是一种专门用于存储、索引和查询高维向量嵌入的系统[1]。这些嵌入是对文本、图像或音频等非结构化数据的数字化表示，能够捕捉语义含义，使应用程序得以执行超越简单关键词匹配的相似性搜索 [2]。这项能力是现代人工智能驱动任务的基石。

这项技术的重要性引发了一段创新与市场增长的高速发展期，常被比作数据系统领域的寒武纪大爆发。2023年，全球向量数据库市场估值为16亿美元；预测显示，到2032年将持续飙升至100亿美元，复合年增长率（CAGR）超过22% [3]。这一爆发式增长催生了一个充满活力且多元化的解决方案生态系统。这些解决方案大致可分为：

- **全托管云原生服务**：Pinecone 等平台提供简化、无服务器的开发者体验，并支持自动扩缩容 [4]。
- **开源独立数据库**：功能强大、可自行托管的系统，如 Milvus、Weaviate、Qdrant 和 Chroma，提供丰富的功能和灵活的部署方式 [1]。
- **库和扩展**：Facebook AI 相似性搜索（FAISS）等高度优化的库 [43] ，以及适用于传统数据库的扩展，例如 Postgres 的 pgvector [44] ，可为现有系统添加向量搜索功能。

这一丰富的生态为开发者提供了众多选择，每种方案都针对性能、可扩展性、成本和功能集之间的不同权衡进行了优化。然而，这种快速且缺乏协调的创新，也给应用开发者带来了沉重代价：API 碎片化问题严重，令人难以应对。

B. API 碎片化问题：可移植性与可维护性的障碍

在向量数据库生态中，对于 API 设计、数据模型和通信协议完全缺乏共识，导致生态四分五裂，给开发者带来沉重负担 [5]。这种碎片化主要体现在以下几个方面：

- **命名约定和操作各不相同**：基本操作的名称并不一致。例如，插入或更新记录这一原子操作，在 Pinecone 中称为 upsert [6], ，在 Milvus 中称为 insert [7]（更新时则另有 upsert 操作），而在 Chroma 中则称为 add 或 upsert[8]。
- **不兼容的通信协议：各协议差异很大**：Pinecone 和 Milvus 使用 gRPC；Qdrant 和 Chroma 提供 RESTful API；Weaviate 则采用 GraphQL。
- **各异的查询与过滤语言**：元数据过滤的语法差异很大。Pinecone 使用 MongoDB 风格的 JSON，Qdrant 采用不同的 JSON 结构，Weaviate 使用 GraphQL where 子句，而 Milvus 使用专有的基于字符串的语言。

这种碎片化导致严重的供应商锁定，因为应用程序逻辑会与单一数据库紧密耦合。迁移到其他提供商会带来高昂的工程工作成本，通常需要彻底重构数据访问层。这是构建可移植且易于维护的人工智能应用的系统性障碍。表 1 展示了基础操作所用 API 的差异。

这种分歧导致了传统数据库技术栈的倒置：工具不再构建于标准

表 I：原生向量数据库 API 的比较分析

Operation	Pinecone (Python SDK)	Weaviate (Python Client v4)	Qdrant (Python Client)
Connect	pc = Pinecone(api_key="...")	client = weaviate.connect_to_local()	client = QdrantClient(host="...")
Create Collection	pc.create_index(name, dimension, metric)	client.collections.create(name, properties,...)	client.recreate_collection (collection_name, vectors_config)
Upsert Vectors	index.upsert(vectors=[{'id': 'a', 'values': [...], 'metadata': {...}}])	collection.data.insert_many([{'properties': {...}}])	client.upsert(collection_name, points=)
Query with Filter	index.query(vector=[...], top_k=5, filter={"genre": {"$eq": "drama"}})	collection.query.near_vector(..., filters=Filter.by_property("genre").equal("drama"))	client.search(collection_name, query_vector=[...], query_filter=Filter(...))

接口（如 SQL）之上，而是将抽象层推入 LangChain 和 LlamaIndex 等框架中。这些框架可能会使数据库沦为单纯的“后端存储系统”，无法参与更高层次的查询优化 [9]。因此，由于缺乏通用的编程基础，价值被外部编排层所攫取。

C. 我们的贡献：*Vextra* 抽象层 *ayer*

本文介绍了 Vextra，这是一种中间件抽象层，为向量数据库操作提供稳定、统一且与数据库无关的 API。Vextra 位于应用程序与数据库之间，恢复了逻辑数据独立性并提升了开发者生产力。本工作的主要贡献有三点：

- **统一的数据和 API 模型**：我们提出了一种高级 API，涵盖集合管理、数据操作和过滤搜索等常见操作，同时屏蔽供应商特定语法和协议。

- **可插拔适配器架构**：我们详细介绍了一种基于适配器设计模式 [10]的系统。该架构将特定于数据库的逻辑隔离到可插拔模块中，使 Vextra 无需更改面向应用程序的 API 即可扩展支持新的向量数据库。

- **编译器启发式查询转换引擎（QTE）**：我们介绍了一种新颖的 QTE，用于解决供应商特定查询语言碎片化这一关键问题。该引擎通过以下方式规范化查询翻译：– **解析** 规范查询，将其转换为通用的 **抽象语法树（AST）** [11],，从而保留查询精确的逻辑结构。– **转译** 该抽象语法树，使用特定于后端的转译器将其转换为原生查询。

  这种基于抽象语法树的方法比直接进行字典到字典的映射更加稳健、易于维护。它确保应用程序逻辑与数据库特定语法完全解耦，从而真正抵御供应商 API 变更，并简化未来将后端集成为适配器的过程。

- **实证验证与可移植性演示**：我们提供了性能评估，量化抽象层带来的开销，以证明其在实践中的可行性。我们还展示了一个检索增强生成应用，说明只需更改一行配置，即可在两个数据库后端之间完成迁移。

Vextra 旨在将向量数据库从单体且紧密耦合的依赖项转变为可互换的组件。这一方法与 JDBC 和 ODBC 对关系型数据库领域的影响如出一辙 [10]。Vextra 是迈向互操作性的重要一步，也为未来的查询优化和联邦搜索铺平了道路。

II. 相关工作

为阐明我们的贡献，我们考察了三个关键的既有研究领域：数据库抽象的历史、NoSQL 中间件的经验，以及当代向量数据管理工具。我们发现了一种反复出现的模式，这也凸显了 Vextra 作为一项适时进展的意义。

A. 数据库抽象层的历史视角

数据库研究中的一个核心主题是数据独立性。关系型数据库实现了物理数据独立性（将应用程序与存储变更隔离开来）。然而，许多人认为它们的逻辑数据独立性较差，因为关系模型往往会暴露实现层面的细节 [12]。

Deshpande 等人认为，与逻辑布局的紧密耦合会使数据库沦为后端存储系统 [9]。他们提出采用更抽象的数据模型，赋予系统进行优化的自由。这一原则直接促成了 Vextra。当前的向量数据库格局体现出极低的逻辑数据独立性。Vextra 提供了面向“记录”和“集合”的更高层 API，以引入一层逻辑独立性。

JDBC 和 ODBC 标准为 SQL 查询提供了统一的 API，从而实现了可移植性 [9]。在更高层次上，对象关系映射（ORM）框架应运而生，用于弥合面向对象模型与关系数据模型之间的“阻抗失配” [9]。Vextra 综合了这两种理念，并将其应用于向量数据库：它提供了类似 JDBC 的标准化接口，但面向以向量嵌入为核心的数据模型。

B. NoSQL 和异构系统中间件的经验教训

NoSQL 生态系统提供了一个高度相关的参照。NoSQL 运动催生了大量数据库类型（如键值型、文档型、列式和图数据库），每种类型都有自己的查询语言 [13]。这种碎片化与向量数据库的当前状况如出一辙，并推动了对抽象层的研究。

研究凸显了中间件固有的权衡：为了实现可移植性并降低迁移成本，人们愿意接受可测量的性能开销 [9]。研究表明，尽管直接 API 调用速度更快，中间件仍能提供至关重要的

解耦能力，尤其是在快速演变的生态系统中。这一发现直接指导了我们对 Vextra 这项权衡进行量化的评估。

此外，对统一查询框架的研究概括了常见的架构模式 [14]。此类系统通常采用解析层、查询规划器 / 转换器和可插拔驱动程序 [15]。这种分层架构直接影响了 Vextra 适配器组件的设计。数据库系统的发展历史呈现出一种周期性模式：范式出现，碎片化发生，最终由抽象层加以解决。Vextra 将这一模式应用于向量数据库范式。

C. 当代向量数据管理框架

为确立创新性，我们将 Vextra 与现有工具进行区分。它们通常针对不同的问题。

VectorAdmin：一种开源数据库管理工具，提供图形用户界面（GUI），可连接多个数据库、检查数据并迁移数据 [16]。其主要界面是可视化的，而非程序化的。它不提供用于应用程序数据访问的 API。VectorAdmin 解决的是统一管理问题，而 Vextra 解决的是统一应用程序开发问题。

Towhee：该框架是一种 ETL 和机器学习流水线工具。Towhee 的核心目标是简化向量嵌入生成 [17]。Towhee 对嵌入生成过程进行了抽象，但向量数据库仍然是终端端点。它不提供针对查询的抽象。

Jina AI：Jina 是一个云原生机器学习运维和微服务框架 [18]。其主要目标是编排完整的分布式人工智能服务。相比之下，Vextra 是一个专注且轻量的库，用于解决数据库 API 抽象问题。Vextra 可以作为 Jina AI 的组件，而不是该框架的竞争对手。

总而言之，现有工具相互补充，而非彼此竞争。VectorAdmin 提供管理功能，Towhee 提供 ETL，Jina AI 提供分布式人工智能系统。这些工具都没有提供专用的编程式抽象层来实现应用程序可移植性。Vextra 是首个旨在填补这一特定且关键空白的解决方案。

III. VEXTRA 统一数据和 API 模型

其基础是一个定义明确的数据模型，在隐藏实现细节的同时捕获核心概念。Vextra 模型旨在保持简单且易于理解。它由通用数据模型和标准化 API 组成。

A. 定义通用向量数据模型

为了保护应用程序免受底层数据库多样化数据结构的影响，Vextra 为向量数据建立了规范表示。该模型基于各种流行向量数据库所具备功能的最大公约数。

Vextra 中数据的基本单位是 VextraRecord。它是一种简单结构，由三个关键元素组成：

- **标识符**：记录必须具有的唯一标识符，可以是字符串或整数。这既适用于使用 UUID 的数据库，也适用于使用数字 ID 的数据库。
- **向量**：必需的浮点数列表，用于表示稠密向量嵌入。为支持日益重要的混合搜索（结合语义搜索和关键词匹配） [2]，数据模型可以扩展为包含一个可选的“稀疏_向量”字段。该字段将包含一个表示稀疏向量的字典，通常由“索引”和“值”键组成。
- **载荷**：一个可选的字典（键值映射），用于存储与向量相关的任意元数据。键必须是字符串，值可以是字符串、数字、布尔值或字符串/数字列表等基本数据类型。这与 Pinecone 和 Chroma 中的元数据概念，以及 Qdrant 中的载荷概念直接对应 [6]。

这些记录被组织到一个**集合**中，该集合是由具有相同向量维度和距离度量的 VextraRecords 组成的命名分组。集合类似于关系型数据库中的表、Pinecone 中的索引，或 Chroma、Milvus 中的集合。集合由名称、维度（所存储向量的长度）和度量（例如“余弦”“欧几里得”“点积”）定义 [19]。

B. 核心 *API* 契约：标准化接口

Vextra API 被设计为简洁的面向对象接口，通过单个 VextraClient 对象公开所有核心数据库功能。这些方法经过有意设计，命名直观，并将不同供应商所使用的不同术语统一起来。

连接与集合管理

- connect(config: dict) →VextraClient：一种静态工厂方法，用于初始化并返回连接到特定后端的客户端。config 字典包含所有必要的连接参数，例如提供商（‘pinecone’、‘qdrant’）、api_key、主机和端口。
- create_collection(name: str,dimension: int, metric: str =‘cosine’，**kwargs)：使用指定参数创建新集合。
- delete_collection(name: str)：删除整个集合。
- list_collections() → 列表[str]：返回所有现有集合的名称列表。

数据操作（增删改查）操作

- upsert(collection_name: str, records: List)：在指定集合中插入或更新一批记录。这个单一的幂等方法整合了不同数据库中的各种添加、插入和插入或更新函数，简化了最常见的写入操作 [6]。
- fetch(collection_name: str, ids: List[Union[str, int]]) →列表：根据唯一标识符检索记录列表。
- delete(collection_name: str, ids: List[Union[str, int]]): 根据标识符从集合中删除一条或多条记录 [33]。

查询与搜索API 中最关键的部分是统一查询方法，它抽象了相似性搜索和过滤的复杂性。

- query(collection_名称: str, vector: List[float], top_k: int, 过滤器: 字典 =None) → 列表：执行相似性搜索。– 向量：用于最近邻搜索的查询向量。– top_k：要返回的最相似结果数量。– 过滤器：可选字典，用于指定元数据过滤条件。

API 模型的一个关键要素是定义**统一过滤语法**。该语法设计为一种简单、可序列化为 JSON 的领域特定语言（DSL），QTE 可以将其转换为各后端原生的过滤语言。其设计深受 MongoDB 广泛采用的查询语法 [20]和 Pinecone [6],过滤机制的影响，采用开发者熟悉的范式。这一选择优先考虑开发者体验和易于采用，而非发明一种新语法。过滤器是一个字典，其中键为元数据字段，值用于指定条件。复杂条件通过以 $ 为前缀的逻辑运算符构建。

用于查找类型为“剧情片”且于 2020 年或之后上映的电影的示例过滤器可表示为：

```
{"$and": [{"类型" : {"$eq" : "drama"}},
{"year": {"$gte": 2020}}]}
```

这种结构具有足够的表达能力，可以表示复杂的逻辑条件，同时仍易于构建和解析。

C. 处理高级且异构的功能特性

任何抽象都无法在不过度复杂或受限的情况下，完美封装每个底层系统的所有功能。要处理并非所有系统都普遍支持的功能，需要采取务实的方法。Vextra 采用双管齐下的策略，在标准化与访问功能强大的后端特定功能之间取得平衡。

首先，对于混合搜索等日益广泛采用的高级功能，Vextra 会将其纳入核心 API。查询方法可扩展为接受一个可选的稀疏向量参数。当同时提供向量（稠密）和稀疏向量时，中间件会将其转换为后端原生的混合搜索调用，适用于支持该功能的后端，例如 Weaviate、Pinecone 和 Qdrant。

其次，对于真正独特的提供商特定功能，Vextra 提供了一种**逃生舱**机制。create_collection 和 query 等关键方法包含可选参数 **kwargs 或 provider specific params: dict_。这使开发者能够传入一个字典，其中包含仅特定后端的适配器能够识别的参数。例如，目标为 Weaviate 的用户可以传入生成式搜索参数，而目标为 Qdrant 的用户 [23]则可以传入参数，以使用其专用的推荐 API [36]。虽然使用这种机制会牺牲该次调用的可移植性，但它能确保抽象层不会成为阻碍访问有价值功能的瓶颈。这一设计选择承认异构生态系统的现实，并为开发者提供了一条实用途径，让他们在需要时使用高级功能，同时不放弃抽象层为其核心优势带来的益处。

IV. 系统架构与实现

Vextra 中间件被设计为一个模块化、可扩展的系统，能够清晰地将统一 API 逻辑与特定数据库的实现细节分离。该架构以适配器设计模式为核心，这是一项成熟的软件工程原则，旨在让彼此不同的接口能够协同工作。这一选择对于实现 Vextra 的首要目标至关重要：将应用程序与底层数据库解耦，从而建立一道抵御供应商驱动的 API 变更的屏障，并确保长期可维护性。

A. 架构蓝图：适配器模式

Vextra 系统由如图 1 所示的多个关键组件组成，这些组件协同工作以处理应用程序的请求。该架构的高层视图包括：

- **VextraClient（API 网关）**：这是应用程序与之交互的面向公众的类。它实现了第 3 节中定义的统一 API 契约。其作用是验证传入的请求，管理与特定后端的连接，并将操作的执行委派给适当的适配器。

- **连接管理器**：一个负责实例化和管理与不同数据库后端连接的工厂组件。当应用程序调用 VextraClient.connect(config) 时，连接管理器会解析配置字典中的 provider 字段，并使用适配器注册表查找和实例化正确的适配器（例如 PineconeAdapter、WeaviateAdapter）。然后，它会将该适配器注入 VextraClient 实例。
- **查询翻译引擎（QTE）——转换为抽象语法树**：查询经过 VextraClient 验证并建立连接后，会被转换为抽象语法树（AST），供适配器后续将其转换为数据库专用查询。

- **适配器注册表**：一个简单的注册表，用于将提供商名称（如“pinecone”这样的字符串）映射到相应的适配器类。只需注册一个新的适配器类，即可轻松扩展系统，使其支持新的数据库。
- **IVextraAdapter 接口**：一个抽象基类或接口，用于定义所有特定于数据库的适配器必须遵循的契约。该接口与 VextraClient 的方法相对应，但仅供内部使用（例如 • doupsert、doquery）。

VextraClient 的方法，但仅供内部使用（例如 _do_插入或更新、_do_查询）。

- **具体适配器（例如 PineconeAdapter、WeaviateAdapter）**：它们是整个系统的主力。每个适配器都是一个实现 IVextraAdapter 接口的类。它包含与特定向量数据库通信所需的全部逻辑，包括初始化其原生 SDK 、转换 Vextra 的统一数据模型，以及将结果规范化为 Vextra 模型。

这种架构确保所有特定于数据库的代码都封装在相应的适配器中。如果某个供应商发布了一个有重大变更的新 API 或 SDK 版本 —— 在这个快速发展的领域中，这种情况很常见，Chroma 的 v1 和 v2 API 不兼容就是一例 [24] ——那么只需更新相关适配器即可。VextraClient，尤其是依赖它的应用程序代码，都完全无需改动。这使系统能够真正抵御 API 的频繁变动。

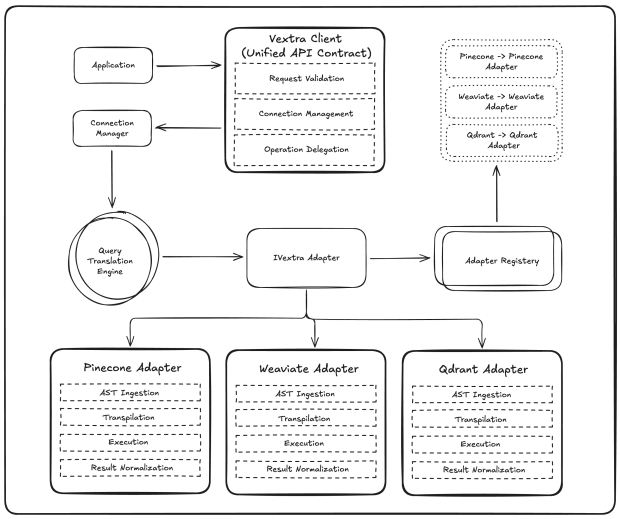

图 1：Vextra 架构：基于适配器模式

B. 查询翻译引擎与适配器接口

中间件的核心逻辑体现在 VextraClient 与已加载适配器之间的交互中。IVextraAdapter 接口要求实现所有受支持操作的方法。该接口的简化版本可用类 Python 伪代码定义如下：

当应用程序调用 $client.query(...)$ 时，Vextra 客户端只需调用 $self.adapter._query(...)$。随后，适配器，具体来说是其 QTE 组件，负责复杂的“查询翻译”过程。QTE 不依赖脆弱的参数直接映射，而是将 Vextra 的规范查询模型 [25] 视为一种形式语言，可编译为不同的目标语言，并通过几个关键步骤确保语义协调：

- **解析为抽象语法树（AST）**：QTE 在收到规范查询对象后的第一步是

```python
class IVextraAdapter(ABC):
    @abstractmethod
    def __init__(self, config: dict):
        # Initialize native client and connect
        pass

    @abstractmethod
    def _create_collection(self, name: str, dimension: int, metric: str, **kwargs):
        pass

    @abstractmethod
    def _upsert(self, collection_name: str, records: List):
        pass

    @abstractmethod
    def _query(self, collection_name: str, vector: List[float], top_k: int, filter: dict) -> List:
        pass

    #... other methods for delete, fetch, etc.
```

图 2：IVextraAdapter 的 Python 伪代码

将其解析为抽象语法树（AST） [11]。AST 是一种基于树的数据结构，用于表示查询的语法和逻辑结构，同时抽象掉 JSON 或目录语法等表层细节。树中的每个节点都代表一个逻辑组件（例如逻辑运算符“$and”、比较运算符“$eq”、字段名称或字面值）。AST 充当查询语义意图的通用且明确的中间表示。例如，过滤器“{$and: [{“year”: {$gt: 2020}}, {“genre”: “sci-fi"}]}”会被解析为一棵树，根节点为 AND 节点，并有两个子节点：一个 GT 节点和一个 EQ 节点。这种稳健的方法能够确保保留逻辑结构，包括运算符优先级，比临时进行字典操作可靠得多。

- **转译为原生查询**：对于每种受支持的数据库后端，QTE 都包含一个专用的“转译器”。转译器（或源到源编译器）将代码从一种高级语言转换为另一种高级语言。在此场景中，转译器是具体适配器中的一个组件，它会遍历抽象语法树，并递归生成适用于目标后端的等效原生查询结构。

  – **Pinecone 转译器** 会遍历抽象语法树，并构造符合 Pinecone 过滤语法的嵌套 JSON 对象。– **Weaviate 转译器** 会遍历同一棵抽象语法树，但生成包含适当“where”和“filter”子句结构的 GraphQL 查询字符串。– **Qdrant 转译器** 会生成类 SQL 布尔表达式字符串。

  如图 3 所示，这种受编译器启发的方法有效地将规范查询模型与特定后端实现解耦，使系统具有高度可扩展性。要支持新的数据库，只需在其对应的适配器中实现新的转译器，无需修改核心查询解析逻辑或规范模型本身。

- **执行**：生成原生查询后，适配器便会使用其原生 SDK 客户端执行该查询。

- **结果规范化**：最后，适配器会解析

数据库返回的原生响应，并将其转换为标准化的 QueryResult 对象列表。这一步至关重要，因为不同的数据库返回的元数据和分数表示方式各不相同，只有这样才能提供一致的体验。例如，返回欧几里得距离（数值越低越好）的后端，必须将分数反转并归一化，使其与返回余弦相似度（数值越高越好）的后端保持一致， [35]，确保应用程序始终收到介于 0 和 1 [19] 之间的一致相似度_分数。

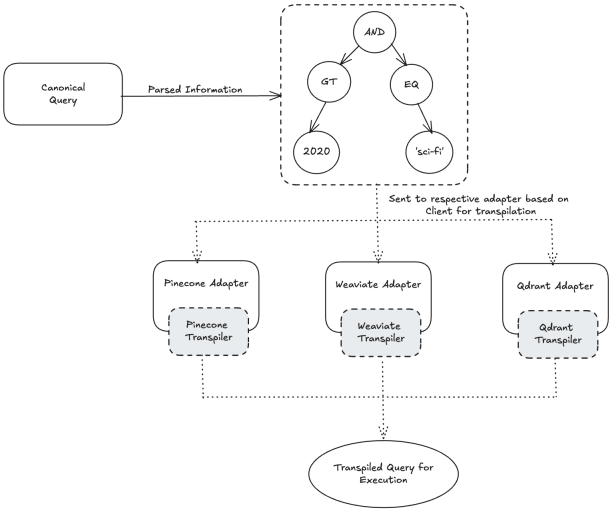

图 3：查询翻译引擎流程

C. 面向代表性向量数据库的适配器实现

为验证该架构的实用性，我们为三种常用向量数据库实现了适配器，每种数据库都代表一种不同的 API 范式。

Pinecone 适配器 Pinecone 适配器使用官方 Pinecone-client Python SDK，该 SDK 主要采用 gRPC。_插入或更新方法会将 VextraRecord 对象列表映射为 Pinecone 的 $index.upsert()$ 方法所需的格式；该方法接收一个元组或字典列表，其中包含 id、值和元数据 [6]。对于查询，适配器的“Pinecone 转译器”组件负责处理过滤器翻译。QTE 首先将 Vextra 过滤器字典解析为通用 AST。随后，“Pinecone 转译器”遍历该 AST，构造 $index.query()$ 过滤器参数所需的嵌套 JSON 过滤器对象。尽管 Pinecone 的语法最初启发了 Vextra 的 DSL [26]，，这种 AST 到 JSON 的转译步骤将二者解耦，确保即使 Pinecone 的过滤器语法发生变化，也只需更新转译器，而无需修改 Vextra 的核心逻辑。

Weaviate 适配器由于 Weaviate 使用 GraphQL 进行查询 [27]，Weaviate 适配器面临更为复杂的转换挑战。该适配器的 _查询方法必须动态构建 GraphQL 查询字符串 [28]。这一过程由其“ Weaviate 转译器”负责：

- 它从“Get”操作的基础查询模板开始。
- 它会添加一个包含查询向量的 ‘nearVector’ 子句。
- 转译器遍历传入的抽象语法树，而不是原始的 Vextra 字典，并递归构建相应的 GraphQL ‘where’ 过滤器字符串 [28]。例如，抽象语法树中的 ‘AND’ 节点会被转译为 Weaviate 的 ‘operands’ 结构，并带有 ‘operator: And’；而表示 ‘$gte’ 的抽象语法树节点则会转换为 ‘operator: GreaterThanEqual’ 语法 [29]。
- 它会将生成的 GraphQL 查询发送到 Weaviate 的 ‘/v1/graphql’ 端点执行。
- 它会解析 JSON 响应，并将结果映射回 ‘QueryResult’ 对象。

Qdrant 适配器 Qdrant 适配器使用适用于 Python 的 ‘qdrant-client’ [30]，与 Qdrant 的 REST API 通信。过滤转换逻辑同样独特，由其 ‘Qdrant 转译器’负责处理。该组件遍历抽象语法树，为 Qdrant 构建一个 ‘Filter’ 对象。它将抽象语法树中的逻辑 ‘AND’ 节点映射为 Qdrant 的 ‘must’ 条件，将 ‘OR’ 节点映射为 ‘should’ 条件，并将比较节点（如 ‘$eq’）映射为 Qdrant 的 ‘FieldCondition’ 和 ‘MatchValue’ 对象 [30]。随后，该转换后的 ‘Filter’ 对象会被传递给客户端 ‘search’ 方法的 ‘query_filter’ 参数。

这三种实现表明，尽管每个转译器内部的逻辑并不简单，但适配器模式结合基于抽象语法树的 QTE，成功封装了这种复杂性。这种受编译器启发的设计确保核心中间件和终端用户应用程序保持简单，并真正与数据库无关。

V. 实验评估

抽象层是否可行，一个关键衡量指标是其性能。抽象虽然在可移植性和开发者生产力方面带来显著优势，但如果中间件引入过高的延迟或降低吞吐量，这些优势便会荡然无存。本节将介绍对 Vextra 的严格实验评估，旨在量化其性能开销，通过实际案例研究展示其可移植性优势，并探讨其统一 API 的表达能力。

A. 实验设置

为确保结果的可复现性和有效性，所有实验均在受控环境中进行。

- **硬件**：所有数据库实例和客户端应用程序均运行在同一 AWS 区域和可用区内的独立 c5.2xlarge 实例上，以最大限度地减少网络波动。每个实例配备 8 个虚拟 CPU 和 16 GiB 内存。
- **软件**：实验使用了测试时可用的各软件组件最新稳定版本：Vextra 1.0、Python 3.14、Pinecone Python 客户端 7.3.0、Weaviate Python 客户端 4.17.0、Qdrant Python 客户端 1.15.1。向量数据库后端通过其官方 Docker 镜像运行：Pinecone（无服务器）、Weaviate 1.33.1 和 Qdrant v1.15.5。

- **数据集**：我们使用了 GloVe-25-angular 数据集 [45]，这是一个标准基准数据集，包含 120 万个 25 维向量。该数据集常用于近似最近邻（ANN） [31] 基准测试研究，可为性能比较提供真实依据 [32]。对于每个数据库，我们都创建了一个集合，并在执行任何查询测试之前将 120 万个向量全部写入其中。

B. 性能开销分析

本实验的主要目标是测量与直接使用原生数据库 SDK 相比，Vextra 中间件引入的延迟和吞吐量开销。

方法：

对于三个后端（Pinecone、Weaviate、Qdrant），我们针对两种最关键的向量数据库操作——插入或更新和查询——执行了一系列基准测试。每项测试均在两种配置下运行：

- **直接 API**：客户端应用程序使用数据库的原生 Python SDK 直接发起调用。
- **Vextra API**：客户端应用程序向 VextraClient 发起调用，随后由它转换请求并将其转发至同一数据库后端。

我们测量了以下指标：

- **平均延迟（毫秒）**：从客户端角度测量完成单次操作所需的平均时间。每项测试运行了 10,000 次迭代，并计算了平均延迟。
- **吞吐量（次操作/秒）**：每秒可完成的操作数。我们通过在 60 秒的时间窗口内，使用 16 个并发客户端线程尽可能多地执行操作来测量此指标。

这些操作在各种条件下进行了测试：

- **插入或更新**：分别使用批次大小为 1、100 和 1,000 条记录进行测试，以衡量单条记录写入与高吞吐量摄取的性能。
- **查询**：使用 top_k = 10 进行测试。评估了两种场景：仅搜索单个向量，以及更复杂的搜索场景，其中包含元数据过滤器，用于检索特定元数据键对应五种可能值之一的向量（例如，“category”: “$in”:）。

结果：性能评估结果总结如下表所示。开销计算如下：

$$\frac{(Vextra\ Latency - Direct\ Latency)}{Direct\ Latency} * 100$$

吞吐量测量结果呈现出类似的低开销模式。对这些结果的分析揭示了中间件对性能影响的一个关键特征。Vextra 引入的绝对开销——即在中间件中用于验证和查询翻译的时间——对于给定的操作类型而言相对恒定。因此，对于大型批次操作或本身成本较高的操作，

表 II：Vextra 操作的性能开销（平均延迟，单位：ms）

Operation	Database	Direct API Latency (ms)	Vextra Latency (ms)	Overhead (%)
Upsert (batch=1)	Pinecone	15.2	15.9	4.6%
	Weaviate	12.8	13.7	7.0%
	Qdrant	8.5	9.1	7.1%
Upsert (batch=100)	Pinecone	45.1	46.8	3.8%
	Weaviate	41.3	43.5	5.3%
	Qdrant	28.9	30.2	4.5%
Upsert (batch=1000)	Pinecone	210.6	215.4	2.3%
	Weaviate	195.7	202.1	3.3%
	Qdrant	135.2	138.9	2.7%
Query (no filter)	Pinecone	25.4	26.1	2.8%
	Weaviate	30.1	31.5	4.6%
	Qdrant	18.3	18.9	3.3%
Query (with filter)	Pinecone	38.6	39.5	2.3%
	Weaviate	45.2	47.1	4.2%
	Qdrant	29.8	30.8	3.4%

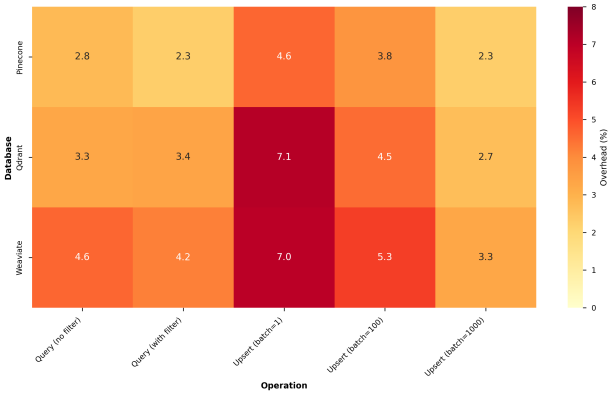

图 4：不同数据库和操作的开销百分比对比

例如大型批次插入或更新，或复杂的过滤查询，这项固定成本占端到端总耗时的比例更小。例如，在 Weaviate 上，单条记录插入或更新的延迟开销为 7.0%，而 1000 条记录的批次插入或更新则降至 3.3%。这表明，对于生产级人工智能系统中常见的高吞吐量、面向批处理的工作负载，抽象层带来的相对性能成本会逐渐降低，因此性能与可移植性之间的权衡非常有利[38]。

C. 可移植性评估：检索增强生成应用案例研究

为定性且有力地展示 Vextra 的主要优势，我们开展了一项涉及简单检索增强生成应用 [37] 的案例研究。该应用程序的功能是根据一组文本文档回答问题。其数据访问逻辑仅需使用 Vextra API 编写一次。

迁移场景

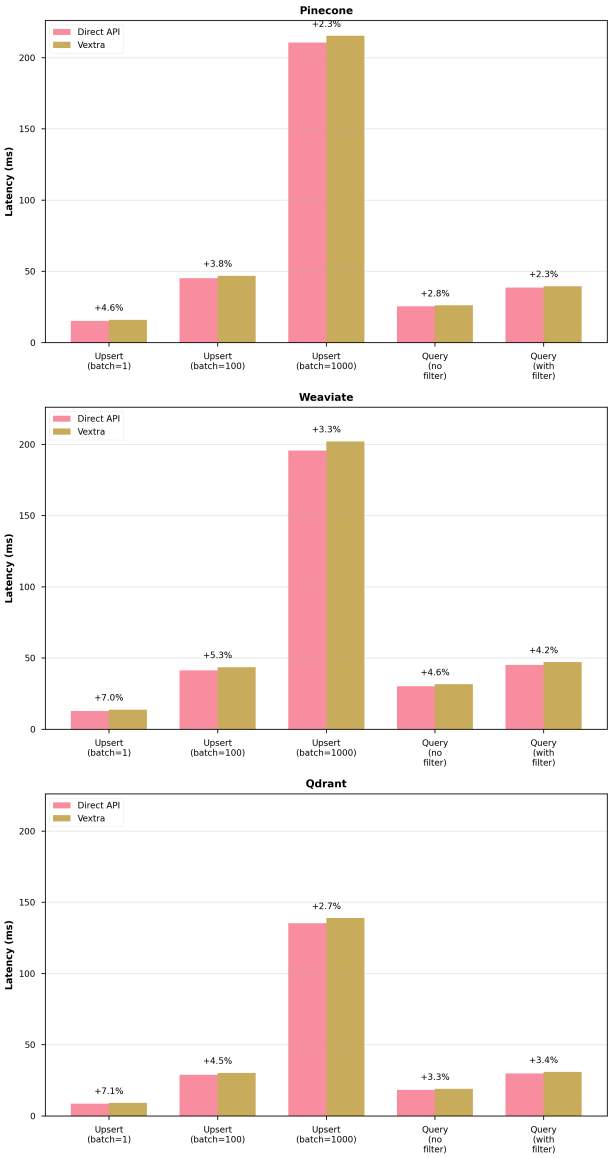

图 5：不同数据库的延迟对比

```python
from vextra_client import VextraClient

# Connect to the database specified in the config
client = VextraClient.connect(config)

def ingest_documents(docs):
    #... code to chunk docs and generate embeddings...
    records =
    client.upsert("rag_collection", records)

def answer_question(query):
    #... code to generate query embedding...
    results = client.query(
        "rag_collection",
        vector=query_embedding,
        top_k=3
    )
    #... code to build context from results and call LLM...
    return llm_response
```

图 6：简化后的应用程序逻辑

最初，应用程序配置为使用 Pinecone 作为其后端。端到端系统的配置对象定义如下：

```
config = {
          "provider": "pinecone",
          "api_key" : "...",
          "environment": "..."
         }
```

应用程序成功运行，文档已导入，问题也得到了成功解答。随后，为了将应用程序迁移到自托管的 Qdrant 实例，只需修改配置对象：

```
config = {
          "provider": "qdrant",
          "host" : "localhost",
          "port": "6333"
         }
```

只需进行这一行更改，重启应用程序后，它便能以 Qdrant 作为向量存储，继续以完全相同的方式运行。无需修改 ingest_documents 或 answer_question 函数。

相比之下，如果不使用 Vextra，迁移就需要大幅重写数据访问逻辑。开发者需要将 pinecone-client 库替换为 qdrant-client，更改连接和初始化代码，将插入或更新调用改为使用 Qdrant 的 PointStruct 对象，并彻底重构查询调用，以使用 Qdrant 的搜索方法和过滤器语法。这项案例研究清楚而具体地展示了使用 Vextra 抽象层所节省的工程工作，以及由此实现的应用程序可移植性。此外，这种轻松切换后端的能力还带来了一项重要的附加优势：开发者可以针对特定工作负载，对不同的向量数据库进行实证基准测试。重写数据访问层的高昂工程成本往往会使项目受制于最初选择的数据库。Vextra 将这项成本降至接近于零，促进市场形成更充分的竞争与透明度，让开发者能够根据性能和成本，而非仅凭历史惯性，选择最优数据库。

D. 表达能力与功能覆盖

Vextra API 旨在满足绝大多数常见的向量数据库使用场景。统一过滤语法支持逻辑运算符（$and、$ or）和一系列比较运算符（$eq、$gt、$in 等），足以构建复杂的元数据查询。不过，我们承认，通用抽象不可避免地需要在功能完整性方面做出权衡。对于高度专业化、特定于后端的功能，例如 Weaviate 基于 GraphQL 的生成式查询，开发者可以在查询方法中使用提供方专用参数这一“逃生舱”。虽然这会牺牲该次调用的可移植性，但能确保 Vextra 不会妨碍使用底层数据库的高级功能。这种务实的设计

为 95% 的操作提供了稳健、可移植的基础，同时仍允许访问其余 5% 的功能。

E. 事务完整性与错误统一

异构系统以不同方式报告错误，使用各自独特的错误代码、异常类型和消息格式。Vextra 中间件的一项重要职责是拦截这些各不相同的错误，并将其映射为一组统一、规范的异常。例如，Pinecone 的身份验证失败（如 HTTP 401）、Weaviate 的模式验证错误，以及自托管 Milvus 实例的连接超时，都应由相应的适配器捕获，并重新抛出为标准化的 Vextra 异常，例如 vex- tra.errors.AuthenticationError、vextra.errors.SchemaError 和 vextra.errors.ConnectionError。

这种统一不只是映射错误代码，还包括转换错误的语义。完善的实现会为这些规范化的表达补充结构化元数据，例如错误是否具有暂时性、是否建议重试（如速率限制错误），或是否属于永久性故障（如无效 API 密钥）。这样，调用应用程序便能构建健壮且与后端无关的错误处理和弹性逻辑（如重试机制、断路器），而无需解析错误字符串，也无需了解各个数据库的故障模式。

VI. 讨论与未来工作

Vextra 的开发与评估证明了，在异构向量数据库生态系统中采用统一抽象层切实可行且颇具优势。不过，这项工作也凸显了其固有局限，并为后续研究开辟了颇具前景的方向。

A. 统一抽象的局限与权衡

任何中间件的主要权衡都在于性能开销与可移植性。我们在 V-B 节中的评估对此进行了量化，结果显示延迟虽有所增加，但增幅较小。对于延迟要求极为严格、必须低于毫秒级的应用程序，直接集成原生 SDK 仍可能是首选方案。

另一个重大挑战是维护负担。向量数据库格局正以惊人的速度演变，API 和功能也频繁更新。要维护整套 Vextra 适配器，确保其与各后端的最新版本兼容，需要持续投入专门的人力。例如，数据库 API 的破坏性变更将要求相应更新并发布其 Vextra 适配器，以确保用户服务不中断。

最后，“逃生舱”机制虽然是访问非标准功能的一种务实方案，但也代表着对抽象层的刻意妥协。应用程序过度依赖这一功能，会削弱可移植性带来的优势。长期目标应是观察哪些提供商特定功能得到了广泛采用，然后将其纳入标准化的 Vextra API 核心，逐步减少对“逃生舱”机制的需求。

B. 高级查询优化与规划的机遇

引入 Vextra 这样的中间件层，可在数据路径中创建一个具有战略意义的控制与观测点，从而实现一类在应用程序层难以甚至无法实施的优化。

- **智能查询缓存**：Vextra 非常适合实现查询缓存层。它可以在可配置的时长内缓存相同查询调用（即相同的查询向量和过滤器）的结果，从而大幅降低高频重复查询应用程序的延迟和成本 [39]。

- **自动请求批处理**：许多应用程序可能会发出多个小型并发查询。更先进的 Vextra 版本可以在极短的时间窗口（例如 5–10 毫秒）内缓冲这些独立请求，并自动将它们合并为一个更大的请求发送到后端数据库。由于大多数向量数据库在执行批处理操作时效率要高得多，这可以显著提高整体吞吐量 [40]。

- **联邦环境中的基于成本的查询优化**：未来版本的 Vextra 可以扩展为同时管理多个数据库后端，充当联邦查询引擎。在这种架构中，如果数据复制或分片存储在不同类型的数据库中（例如，高性能内存存储和低成本基于磁盘的存储），Vextra 就可以实现基于成本的优化器。它可以根据延迟要求、查询复杂度，甚至不同平台上查询所产生的货币成本，将查询路由到最合适的后端 [41]。

C. 迈向自配置可扩展中间件

当前的适配器模型要求为每个新数据库手动实现适配器。Vextra 的长期愿景是发展为一个更加自配置、由社区驱动的生态系统。

- **动态能力发现**：随着向量数据库日趋成熟，它们可能开始公开标准化的元数据端点，用于描述其功能、支持的过滤操作和索引选项（例如通过 OpenAPI 或 Swagger 规范）。未来的 Vextra 可以查询这些端点，动态配置适配器的部分功能，从而减少支持新数据库或现有数据库新版本所需的人工工作量 [42]。

- **社区驱动的适配器框架**：为确保 Vextra 能够跟上向量数据库快速增长的步伐，我们计划规范适配器开发流程，并为社区贡献提供清晰的框架和工具。通过创建一个简单且文档完善的新适配器开发、测试与提交流程，Vextra 将能够借助开源社区的集体力量，更快速地扩展其支持范围。

VII. 结论

向量数据库的不断涌现一直是当前人工智能创新浪潮的关键推动力，但也无意中造成了碎片化且充满挑战的开发环境。这些关键数据系统之间缺乏 API 标准化，给开发者带来了沉重负担，阻碍了可移植性，增加了维护成本，并造成了不受欢迎的供应商锁定。

本文围绕这一问题做出了三项主要贡献。首先，我们系统地识别并量化了主流向量数据库解决方案中的 API 碎片化问题，为开发者面临的挑战提供了具体证据。其次，我们提出、设计并实现了**Vextra**，这是一种新颖的中间件抽象层。Vextra 基于久经验证的适配器设计模式，为所有核心向量数据库操作提供了统一、直观的 API，从而有效地将应用程序逻辑与特定的后端实现解耦。第三，我们进行了严格的实验评估，验证了所提方法的有效性。结果表明，Vextra 带来的性能开销极小且可以接受：在生产环境风格的工作负载下，通常仅为个位数百分比；与此同时，它显著提升了应用程序的可移植性，这一点已通过无缝后端迁移案例研究得到证明。

Vextra 不只是一款软件工具，更是迈向更成熟、更具互操作性的向量数据库生态系统的一种路径构想。通过提供稳定、通用的接口，它降低了开发者的入门门槛，减少了采用新数据库技术的阻力，并为未来更高层次的查询优化创造了战略支点。我们认为，标准化抽象并非仅仅是为了方便，而是人工智能应用开发持续增长、实现可持续发展与普及所必需的基础设施。

VIII. 人工智能生成内容声明

我们使用 ChatGPT 和 Gemini 作为研究辅助工具，集思广益，搜集文献综述材料，并查找与本文探讨主题相关的潜在关键词和参考文献。例如，我们使用这些工具作为入口，搜索前文讨论过的数据库范式，并借此为研究 API 碎片化问题提供参考。本文中的图表、代码或文字均未使用人工智能技术生成。

参考文献

[1] S. Kukreja, T. Kumar, V. Bharate, A. Purohit, A. Dasgupta 和 D. Guha, 《向量数据库与向量嵌入综述》, 2023年国际人工智能与图像处理研讨会（IWAIIP）, 印度尼西亚日惹, 2023年, 第231–236页, 数字对象标识符: 10.1109/IWAIIP58158.2023.10462847[2] Jing, Z., Su, Y., Han, Y., Yuan, B., Xu, H., Liu, C., Chen, K. 和 Zhang, M., 《大语言模型与向量数据库相遇：综述》, 2024年, arXiv:2402.01763。 [在线]。可访问: https://arxiv.org/abs/2402.01763[3] SNS Insider, Strategy and Stats, 《2032年向量数据库市场规模、份额与趋势报告》, SNS Insider - Strategy and Stats, 2025年3月。[在线]。可访问: https://www.snsinsider.com/reports/vector-database- market-5881

[4] Kumar, M.,《向量数据库格局终极指南：2025年及未来》, 2025年6月。 [在线] 可访问: https://www.singlestore.com/blog/-ultimate-guide-vector-database-landscape-2024/ [5] T. Taipalus, 《向量数据库管理系统：基本概念、用例与当前挑战》, 《认知系统研究》, 第 85, 101216, 2024, 10 1016 2024 101216 卷, 第页。日[6] 期。数字对象标识符：。/j.cogsys.。《Pinecone数据库 - Pinecone文档》, Pinecone.io, 2025年。https://docs.pinecone.io/ [7] 《Milvus向量数据库文档》, Milvus.io, 2024年。https://milvus.io/docs [8] Chroma-Core, 《Chroma：面向人工智能应用的开源搜索与检索数据库》, 2022年, GitHub代码仓库。 [在线]。可访问: https://github.com/chroma-core/chroma [9] A. Deshpande, 《超越关系：提升至实体关系抽象的理由》, 2025年。 [在线]。可访问: https://vldb.org/cidrdb/papers/2025/p15-deshpande.pdf [10] M. G. Al-Obeidallah、D. G. Al-Fraihat、A. M. Khasawneh、A. M. Saleh 和 H. Addous, 《适配器设计模式对软件可维护性的影响：实证研究》, 2021年信息技术国际会议（ICIT）, 2021年7月, 数字对象标识符: 10.1109/ICIT52682.2021.9491719[11] B. Wan、B. P. Hu、L. Zhou 和 C. .J..R Shi, 《MCAST：一种基于抽象语法树的电路仿真模型编译器》, 第249–252页, 2月。 2004, 10 1109 2003 1249397 数字对象标识符：。/cicc.。[12] A. Rafique、D. Van Landuyt、B. Lagaisse 和 W. Joosen, 《NoSQL数据存储的数据访问中间件对性能的影响：性能与迁移成本之间权衡的研究》, 载于《IEEE云计算汇刊》, 第6卷, 843–856页, 2018年7—9月, 第3期。数字对象标识符：。/TCC.。[13] 25h1756 B. G. Tudorica 和 C. Bucur, 《若干NoSQL数据库的比较及评述》, 2011年第10届RoEduNet教育与研究网络国际会议, 罗马尼亚雅西, 2011, 1 5, 10 1109 2011 5993686 第–页。数字对[14] 象标识符：。/RoEduNet.。C. Zhang 和 J. Xu, 《面向NoSQL数据库的统一SQL中间件》, 2021年4月。 2018, 10 1145 3220199 3220212 数字对象标识符：。/。[15] H. Zhang、C. Zhang、R. Hu、X. Liu 和 D. Dai, 《面向异构数据库的统一SQL查询中间件》, 《物理学杂志：会议系列》, 第1873卷, 第1期, 第012065–012065页, 2021年4月, 数字对象标识符: 10.1088/1742-6596/1873/1/012065[16] 《VectorAdmin - 向量数据库管理与工具套件》, Vectoradmin.com, 2025年。 [在线]。可访问: https://vectoradmin.com/ [17] towhee-io, 《towhee-io/towhee：Towhee是一个致力于简化并加速神经数据处理流水线的框架》, 12月。 05, 2023, [ ] GitHub代码仓库。在线。可访问：https://github.com/towhee- io/towhee[18] NotAndOr, 《GitHub - NotAndOr/jina-ai.jina：借助云原生技术构建多模态人工智能服务》, 2025年, GitHub代码仓库。 [在线]。可访问: https://github.com/NotAndOr/jina-ai.jina [19] A. Levy、B. R. Shalom 和 M. Chalamish, 《相似度度量及其在数据科学应用中的指南》, 《大数据杂志》, 第12卷, 第1期, 2025年7月, 数字对象标识符：10.1186/s40537-025-01227-1 [20]E. Botoeva、D. Calvanese、B. Cogrel 和 G. Xiao, 《MongoDB查询的表达能力与复杂性》, Dagstuhl研究在线出版服务器, 2018年。 [在线]。可访问: https://drops.dagstuhl.de/opus/volltexte/2018/8607/ [21] R. Bhagdev、S. Chapman、F. Ciravegna、V. Lanfranchi 和 D. Petrelli, 《混合搜索：有效结合关键词搜索与语义搜索》, 《计算机科学讲义》, 第554–568页, 数字对象标识符：10.1007/978-3-540-68234-9 41_[22] W. Kong、J. M. Dudek、C. Li、M. Zhang 和 M. Bendersky, 《SparseEmbed：结合上下文嵌入学习用于检索的稀疏词汇表示》, 第45届ACM SIGIR信息检索研究与开发国际会议论文集, 2023年7月, 数字对象标识符：10.1145/3539618.3592065[23] 《Qdrant - 向量数据库》, qdrant.tech。 [在线]。可访问：https://qdrant.tech/ [24] jagchat, 《[BUG]: chroma同步错误》, 2025年2月17日, GitHub代码仓库。 [在线]。可访问：https://github.com/Mintplex-Labs/vector-admin/issues/164[25] M. Manukyan, 《规范模型》, 第16届信息集成与基于网络的应用及服务国际会议论文集, 第320–329页, 2014年12月, 数字对象标识符：10.1145/2684200.2684278

[26] E. Akik、M. Vještica、V. Dimitrieski、M. ˇCelikovi´c、S. Kordi´c 和 S. Risti´c，《通过领域特定语言与向量数据库交互》，《Open Computer Science》，第15卷，第1期，2025年1月，数字对象标识符：10 1515 2025 0036 。/comp- -
[27] J. Jie、J. Wang、G. Li 和 J. Pan，《向量数据库管理系统综述》，2023年，arXiv:2310.14021。 [在线]。可访问：https://arxiv.org/abs/2310 14021 。O. Hartig 和 J. Pérez，《GraphQL的语义与复杂性》，2018年万维网会议论文集 - WWW ‘18, 2018, 10 1145 3178876 3186014 数字对象标识符：。/。
[28] 《搜索运算符，Weaviate文档》，Weaviate.io，2020年。[在线]。可访问：https://weaviate.io/developers/weaviate/api/graphql/search-operators
[29] 《API参考，Qdrant》，qdrant.tech，2025年。 [在线]。可访问：https://api.qdrant.tech/api-reference
[30] P. Indyk 和 R. Motwani，《近似最近邻》，第30届ACM计算理论年度研讨会论文集 - STOC ‘98，1998年，数字对象标识符：10.1145/276698.276876
[31] 《选择向量数据库：2023年比较与指南》，Vectorview.ai，2023年。[在线]。可访问：https://benchmark.vectorview.ai/vectordbs.html
[32] C.-O. Truica、F. Radulescu、A. Boicea 和 I. Bucur，《异步复制文档型数据库中增删改查操作的性能评估》，2015年第20届控制系统与计算机科学国际会议，5月。2015, 10 1109 2015 32 数字对象标识符：。/CSCS.。
[33] S. Jiang 和 H. Mu，《面向对象分析与设计中的设计模式》，2011年IEEE第二届软件工程与服务科学国际会议，中国北京，2011年，第326-329页，数字对象标识符：10.1109/ICSESS.2011.5982229
[34] G. Qian、S. Sural、Y. Gu 和 S. Pramanik，《欧几里得距离与余弦角距离在最近邻查询中的相似性》，2004年ACM应用计算研讨会论文集 - SAC ‘04, 2004, 10 1145 967900 968151 数字对象标识符：。/。
[35] A. Duhan、A. Singhal、S. Sharma、Neeraj 和 A. MK，《语义搜索与推荐算法》，2024年，arXiv:2412.06649。 [在线]。可访问：https://arxiv.org/abs/2412 06649 。
[36] Y. Gao 等，《面向大语言模型的检索增强生成：综述》，2023年12月，arXiv:2312.10997。 [在线]。可访问：https://arxiv.org/abs/2312 10997 。
[37] G. M. Lohman 和 J. A. Muckstadt，《批处理操作的最优策略》，《ACM数据库系统汇刊》，第2卷，第3期，第209 222, 1977, 10 1145 320557 320558 –页，9月。数字对象标识符：。/。
[38] T. K. Sellis，《关系数据库系统的智能缓存与索引技术》，《信息系统》，第13卷，第2期，第175–185页，1月。1988, 10 1016 0306 4379 88 90014 2 数字对象标识符：。/ - ( ) - E. A. ELSAYED 和 O. I. UNAL，《自动存储与检索系统的订单批处理算法和行程时间估计》，《国际生产研究杂志》，第27卷，第7期，第1097–1114页，7月。1989, 10 1080 00207548908942610 数字对象标识符：。/
[39] G. Gardarin、F. Sha 和 Z.-H. Tang，《校准IRO-DB这一面向对象联邦数据库系统的查询优化器成本模型》， [在线]。可访问：https://www.vldb.org/conf/1996/P378.PDF
[40] D. Nam、B. Myers、B. Vasilescu 和 V. Hellendoorn，《借助机器学习改进API知识发现：可比较API方法的案例研究》，2023年IEEE/ACM第45届软件工程国际会议（ICSE），澳大利亚墨尔本，2023年，第1890-页。1906, 10 1109 48619 2023 00161 数字对象标识符：。/ICSE .。
[41] M. Douze 等，《Faiss库》，2024年1月16日，arXiv:2401.08281。[ ] 2401 08281 在线。可访问：https://arxiv.org/abs/。
[42] J. Shim、J. Oh、H. Roh、J. Do 和 S.-W. Lee，《利用现代固态硬盘加速向量数据库》，《VLDB基金会会议录》，第18卷，第11期，第4710–4722页，2025年7月，数字对象标识符：10.14778/3749646.3749724
[43] Huggingface.co，GloVe 25角度数据集。 [在线]。可访问：https://huggingface.co/datasets/hhy3/ann-datasets/blob/main/glove-25-angular.hdf5