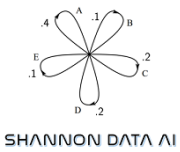
ShannonBase





面向 AI 时代的新一代数据库 —— 基于 MySQL 生态的统一 HTAP、数据库内机器学习、向量检索与生成式 AI

商业 / 技术白皮书

版权所有 © 2026 年，Shannon Data AI

目录

面向对象 5
摘要 6
构建现代数据平台面临的挑战 7
我们的解决方案-ShannonBase 8
架构简介 8
双引擎设计原则 8
HTAP 与智能查询路由 10
查询优化与代价模型 11
实时变更传播 12
Rapid 引擎 —— 内存列式存储 13
三层存储架构 14
Storage Index（存储索引） 15
MVCC 与事务一致性 15
向量化执行与 SIMD 加速 15
并行查询执行（Morsel） 16
检查点与恢复 16
资源治理与可观测性 16
并行加载 16
Autopilot —— 智能自管理内存引擎 17
表重要性评分 18
基于时间的重要性衰减 19
四阶段自动加载算法 19

版权所有 © 2026 年，ShannonBase Project Contributors / Public

内存阈值管理 19
系统静默检测 20
用户加载保护 20
后台协调线程 20
元数据自举 20
Autopilot 能力总览 21
AutoML —— 数据库内机器学习 21
数据库中机器学习面临的现有挑战 22
AutoML 实现方式 22
使用示例 23
训练与推理性能 24
向量检索与生成式 AI 24
原生 VECTOR 数据类型 24
数据库内向量嵌入生成（Embedding） 25
数据库内 LLM 推理 25
RAG 架构 26
JavaScript 多语言执行环境（MLE） 27
数据库内的 Native-Agent 28
设计动机：弥合数据与智能之间的鸿沟 29
架构 29
意图路由与执行路径 31
对话记忆与多轮上下文 31
Schema 感知的 SQL 生成 33
事务写安全 33
插件扩展体系 34

版权所有 © 2026 年，ShannonBase Project Contributors / Public

直接访问 ML 基础能力 35
与 ShannonBase 架构的集成 36
典型应用场景 37
安全与数据治理 37
Native-Agent 能力总览 38
ShannonBase 部署 39
部署模式 39
系统参数与可配置性 40
安全、可观测性与可管理性 41
ShannonBase 目标负载与应用场景 41
应用场景 41
结论 42
资源 42
联系我们 42

面向对象

本文档属于 ShannonBase 技术白皮书，用来描述 ShannonBase 基本技术原理和基本概念。ShannonBase 是一款基于 MySQL 8.4.11 LTS 构建的开源 HTAP 数据库管理系统。

文档面向工程师、架构师与技术决策者，用来阐述系统架构、Rapid 内存列式引擎、变更传播与恢复、AutoML、向量检索与生成式 AI、JavaScript MLE、Native-Agent，以及部署方式与系统参数等模块基本概念和相关技术原理。

版权所有 © 2026 年，ShannonBase Project Contributors / Public

摘要

ShannonBase 是一款基于 MySQL 生态构建的开源、HTAP（Hybrid Transactional/Analytical Processing） 数据库管理系统。当前版本 v0.1.0（基于 MySQL 8.4.11 LTS 内核构建）。它在同一实例内提供 OLTP 事务处理、实时 OLAP 分析、数据库内机器学习（AutoML）、向量数据类型及向量相似度检索、生成式 AI 推理以及基于 JavaScript Agent-Runtime 的 Native-Agent。上述功能通过标准 MySQL 接口访问。本版本定位为单节点 HTAP 与只读副本横向扩展；跨节点共享 Rapid 内存的集群形态将在企业版本中支持。

通过在 InnoDB 之外嵌入自研的基于内存的列式存储引擎-Rapid，ShannonBase 借助透明查询路由实现了真正的 HTAP：分析型负载自动卸载至列式引擎，事务型操作继续在行存引擎上执行。两引擎之间通过 DML Notification（默认）或 COMMITTED_BINLOG（已提交 binlog 解码）保持同步；同时支持解析 Redo log 方式来进行同步（适用于 Cloud-Native 架构）；分析查询使用与 InnoDB 相同的 ReadView 以及查询快照所需的变更水位（Change Watermark），绝不读取未提交或部分应用的事务。Rapid 的数据可由 InnoDB 加 binlog 的方式来实现完整重建。因此，列式引擎只承担加速，不承担持久性与提交。

ShannonBase 将自身定位为 AI 时代的 MySQL—— 在提供熟悉的 MySQL 接口、协议与生态的同时，交付现代企业数据平台所需的分析深度、机器学习广度与生成式 AI 能力。

版权所有 © 2026 年，ShannonBase Project Contributors / Public

构建现代数据平台面临的挑战

当前，组织在构建现代数据平台时候正面临着传统架构难所面临的挑战：不显著增加复杂度、成本与运维负担的前提下解决的多重问题：

- **数据复杂性：数据量的爆炸式增长、格式的多样性（结构化、半结构化、非结构化）以及数据源的异构性，带来了 ETL（Extract/Transform/Load）难以规模化承受的集成负担。**
- **服务分散：传统架构将不同负载分隔到各自独立的系统中 —— OLTP（Online Transactional Processing）数据库、OLAP（Online Analytical Processing）数据仓库、ML（Machine Learning）平台、向量数据库，LLM（Large Language Model）服务等 —— 每个系统都需要独立的资源配置、安全策略与运维周期。**
- **实时数据处理：传统 ETL 会在事务发生与数据可供分析之间引入延迟。现代应用要求对实时事务数据进行实时分析。**
- **总体拥有成本（Total Cost of Ownership，TCO）高：运行多个专用服务会成倍增加基础设施成本、工程投入与运维复杂度。当 LLM 依赖外部服务时，生成式 AI 负载还会进一步增加 GPU 资源成本。**
- **数据安全与治理风险：每一次跨系统边界的数据搬迁都会扩大攻击面并带来合规风险，例如 GDPR 等标准严格限制数据离开数据域。**

版权所有 © 2026 年，ShannonBase Project Contributors / Public

我们的解决方案-ShannonBase

ShannonBase 是一款完全开源、可用于生产环境的数据库管理系统，使组织能够高效运行 OLTP 事务、实时 OLAP 分析、数据库内机器学习、向量相似度检索以及生成式 AI 推理——所有操作均可通过单一的 SQL 语句完成，无需 ETL，也无需数据迁移。

架构简介

ShannonBase 采用分层、模块化架构，在 MySQL 内核基础上扩展了面向列式分析、机器学习与 AI 推理的专用组件：

层级	组件
SQL 接口层	MySQL 兼容解析器、DDL/DML 处理器、标准 ODBC/JDBC 连接器
查询处理层	在 MySQL Parser 与 Optimizer 基础上扩展了 HTAP 成本模型与二级引擎路由钩子
HTAP 路由层	PrepareSecondaryEngine、ML 卸载路由、RapidOptimize 计划改写、代价比较与安全回退
执行层	InnoDB 行存执行器（OLTP 路径）；Rapid 批式（向量化）迭代器：扫描/过滤/连接/聚合/排序/窗口，支持 morsel 并行
存储层	InnoDB 引擎；Rapid IMCS：ART 索引、Storage Index、字典/Bit Pack/压缩、列级 MVCC 版本
变更传播	DML Notification 或 COMMITTED_BINLOG（已提交 binlog 解码）→ Rapid IMCU 写入
Autopilot 层	SelfLoadManager：重要性评分、衰减、优先级队列加载/淘汰、静默检测、schema 自举
ML 与 AI 层	AutoML（LightGBM）+ ONNX Runtime（嵌入/生成/PII）+ 原生 VECTOR 类型与 DISTANCE 距离函数
MLE 层	JerryScript JavaScript 引擎（可执行 SQL、发送结果集）
Agent Engine	sys.shannon_chat：34 个工具、L1–L4 插件分发、事务写安全与审批

双引擎设计原则

- 透明性：应用使用标准 MySQL SQL，无需任何代码改动；引擎选择对使用者不可见。

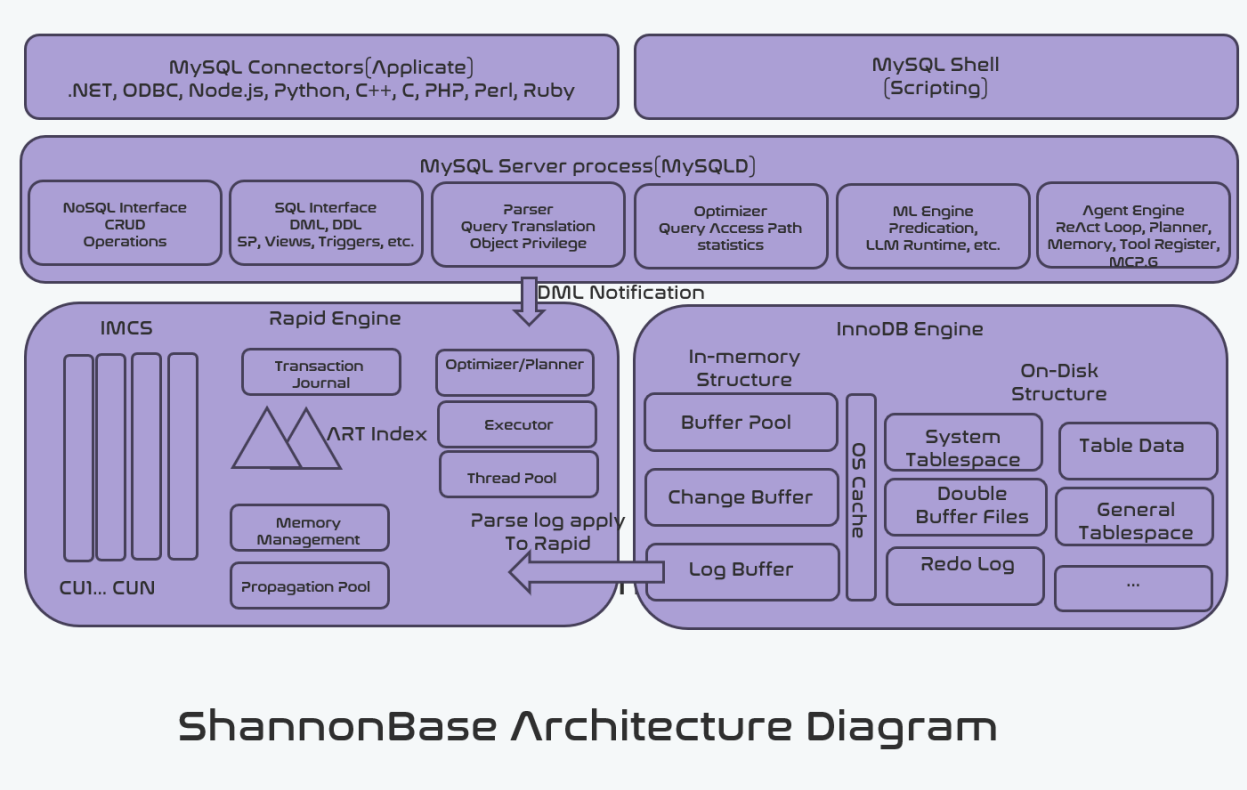
版权所有 © 2026 年，ShannonBase Project Contributors / Public

- 一致性：基于 InnoDB ReadView 与传播水位的 MVCC，确保分析结果始终与语句的事务快照一致。
- 智能化：基于成本模型的路由分类器为每条查询选择最优引擎，并根据数据就绪状态与查询复杂度动态调整。
- 高效性：向量化执行、SIMD、Storage Index 跳读、位打包（Bit Pack）列与字典掩码，最大化每单位内存的分析吞吐量。
- 安全性：Rapid 的数据全部可由 InnoDB 与 binlog 重建；持久性与提交始终由 InnoDB 承担。
- **可恢复性：Rapid 使用不可变列式检查点（manifest v3，保留两代）加已提交 binlog 重放实现快速重启；检查点缺失或校验失败时自动回退为从 InnoDB 重新加载，恢复完成前该表不提供 Rapid 读取。**

该系统基于 MySQL 代码库构建，与 MySQL 协议、SQL 语法、ACID 以及周边的连接器、ORM 与管理工具生态 100%兼容。现有 MySQL 应用可以逐步采用 ShannonBase 的分析与 AI 能力，而无需对应用层做任何修改。

模块	功能	核心价值
HTAP 引擎	InnoDB 与 Rapid 之间的智能双引擎查询路由	无需 ETL 的实时分析
Rapid 引擎	支持 ReadView MVCC、向量化、ART 索引、Storage Index 与位打包列的内存列式存储	TPC-H 查询约一个数量级的加速
变更传播	DML Notification 或 COMMITTED_BINLOG 解码实现 InnoDB→Rapid 实时同步	毫秒级数据新鲜度（按查询水位等待）
AutoML	通过 SQL 实现数据库内机器学习训练、推理与可解释性分析	无需导出数据，无需 Python
向量检索	原生 VECTOR 类型、相似度函数与混合 SQL 查询	数据库内 RAG 与语义检索
JavaScript MLE	基于 JerryScript 的数据库内 JS 存储过程/函数	数据库内 ETL、特征工程与 AI 后处理
GenAI / LLM	进程内嵌入与 LLM 推理（ONNX Runtime 本地、Ollama 或云 API）	数据主权可选：本地推理全程不离开实例

版权所有 © 2026 年，ShannonBase Project Contributors / Public



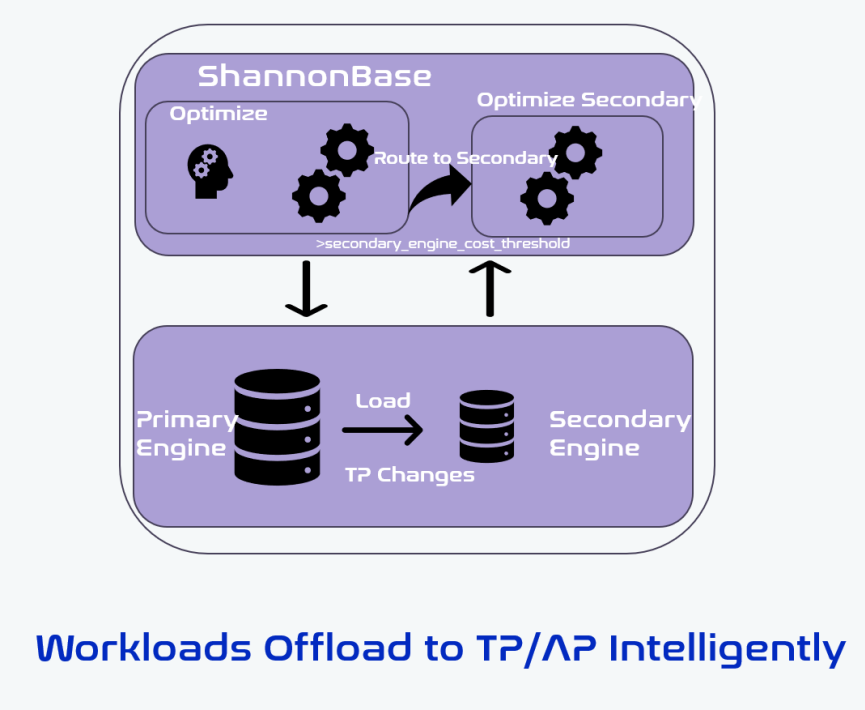
ShannonBase Architecture Diagram

HTAP 与智能查询路由

ShannonBase 的 HTAP 能力通过双引擎架构实现：InnoDB 作为事务型负载的主行存引擎，Rapid 引擎作为分析型负载的二级内存列式引擎。MySQL 8.X 的二级引擎接口被实现为四个钩子：（1）PrepareSecondaryEngine（硬性前置条件与资源准入）、（2）SecondaryEnginePrePrepareHook（基于代价阈值或 ML 分类器的卸载决策）、（3）optimize_secondary_engine（RapidOptimize 计划改写）与 （4）compare_secondary_engine_cost（Rapid 代价估算）。

只有隔离级别为 READ COMMITTED 或 REPEATABLE READ 的语句才会被卸载；FORCED 模式跳过代价比较，但仍执行全部硬性检查。

版权所有 © 2026 年，ShannonBase Project Contributors / Public



路由决策对应用完全透明。标准 SQL 通过 MySQL Connector 提交至 ShannonBase；系统会自动选择最优执行路径，用户无需添加 hint 或修改配置。若 Rapid 尚未追上查询快照所需的传播水位、目标表处于隔离（quarantine）状态，或 Rapid 计划无法转换为完整的查询路径，查询语句则会回退到 MySQL/InnoDB 的原始计划——从而能够保证正确性。

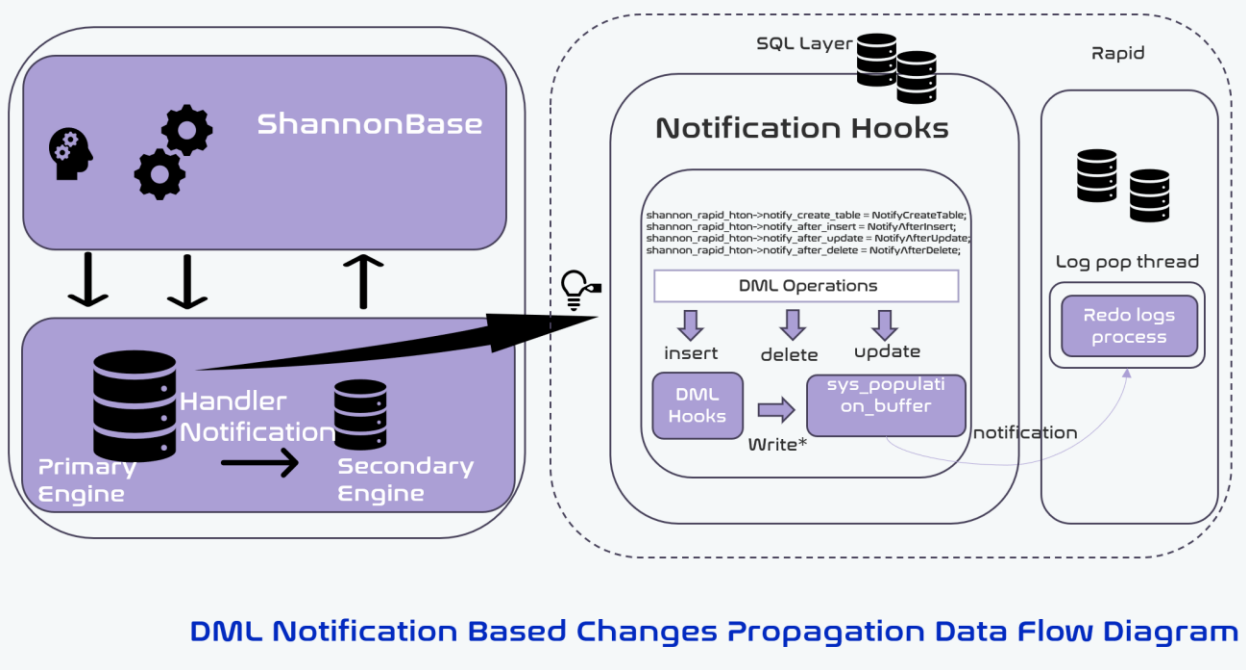
负载类型	执行引擎
点查询、INSERT/UPDATE/DELETE	InnoDB（行存、MVCC、B+树索引）
全表扫描、聚合运算、复杂 JOIN	Rapid 引擎（内存列式、向量化）
混合 HTAP 查询	由优化器针对每条查询自动做出路由决策
Rapid 尚未追上查询所需水位，或表被隔离	自动回退至 InnoDB（安全保证）

查询优化与代价模型

Rapid 优化器在 MySQL 生成的 AccessPath 之上运行：先把 Item 条件翻译为 Rapid 谓词，再按固定顺序应用规则——谓词下推、Storage Index 剪枝、投影裁剪、LIMIT/OFFSET 下推、聚合下推（仅当连接类型与多重性证明等价）与代价重估，连接重排。代价模型使用 IMCS 自身的统计信息（行数、NDV、直方图、分位数、逐 IMCU 的 zone map），并对两阶段读取、溢写与并行度分别计价。若 Rapid 计划无法转换为可执行迭代器，优化器保留 MySQL 的原计划。

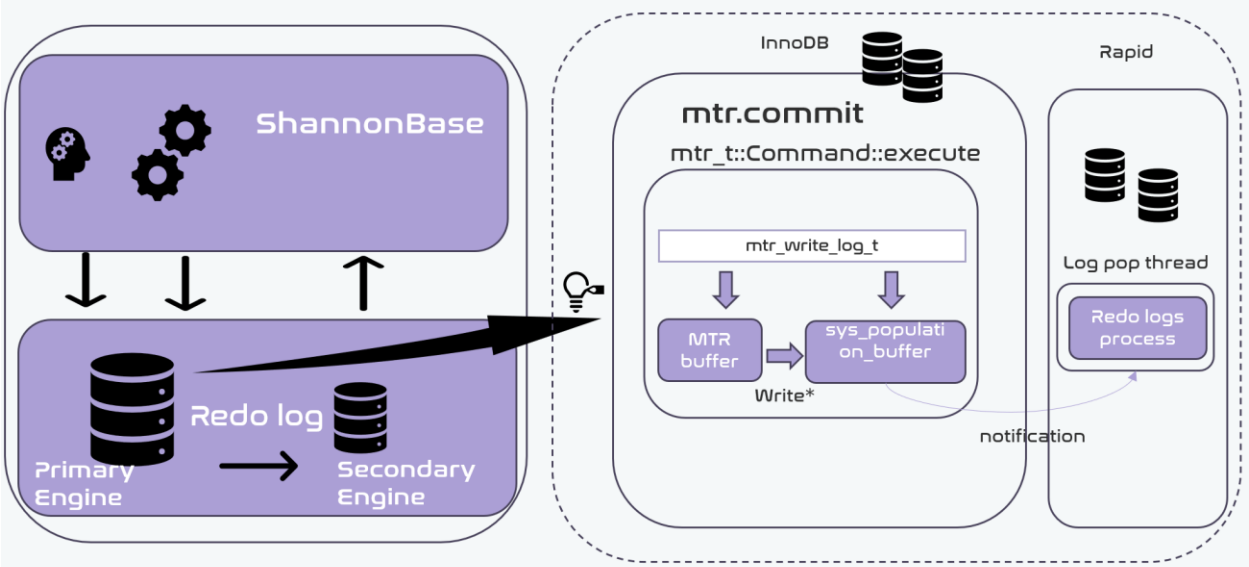
实时变更传播

ShannonBase 提供两种变更传播模式，由 rapid_change_propagation_mode 选择（仅可在没有任何已加载表时切换）：



DML Notification Based Changes Propagation Data Flow Diagram

- CAPTURE（默认，DML Notification）：InnoDB DML 通知钩子（NotifyAfterInsert/Update/Delete）在提交前把新行数据（更改后行数据）、旧行数据（更改前行数据）以及 Off-page 的 BLOB 类型数据打包为变更记录，写入 16 分片的环形缓冲区（Ring Buffer）；传播工作线程异步应用这些变更记录。记录携带源 InnoDB 事务 ID，并以 ACTIVE 版本进入 Rapid 的 MVCC；源事务提交或回滚的结果由 SourceTxnTracker 发布，决定该版本最终提交还是撤销。正常写入负载下传播延迟为个位数毫秒级。



Redo Log Based Changes Propagation Data Flow Diagram

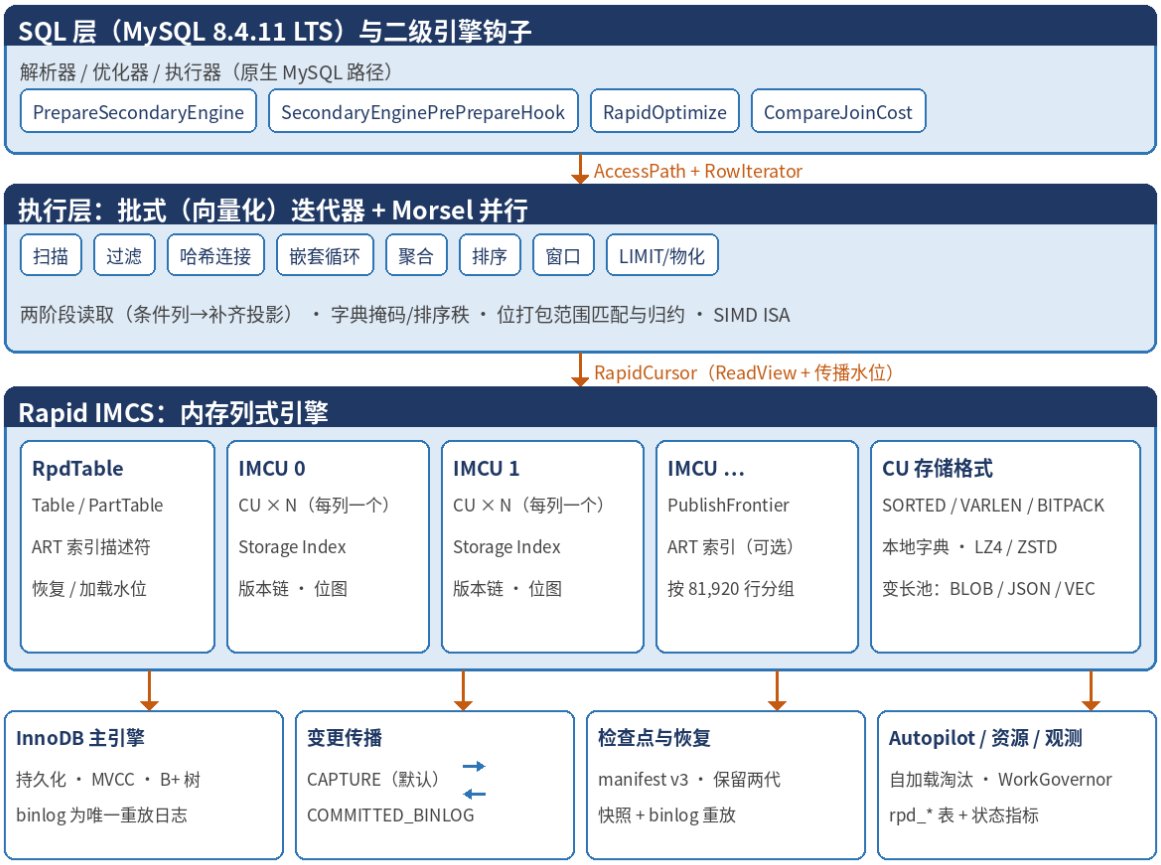
- COMMITTED_BINLOG：后台 binlog Reader 解码已提交的源事务，并构造出 CAPTURE 形式相同格式的行记录，改场景适用于只读副本或希望与源事务路径完全解耦的场景。该模式要求可用的 binlog 配置（例如 binlog_order_commits=ON）；配置不满足时相关表会给出明确错误提示。

两种模式都不改变源事务的提交结果：Rapid 端写入失败只会阻止该表继续被二级引擎读取，即不会将涉及到该表的分析负责 offload 到 rapid 引擎，而不会让已提交的源事务失败。分析查询在启动时按固定顺序取得 InnoDB ReadView、捕获该表当前的变更传播水位并只等待到该水位，因此不会读取未提交或部分应用的事务。

Rapid 引擎 —— 内存列式存储

Rapid 引擎是 ShannonBase 自研的二级存储引擎。它采用：Table → IMCU → CU 的三层内存列式结构，配合批式（向量化）执行、ART 索引与 Storage Index，在保持与 InnoDB 事务一致性的同时最大化分析吞吐。

版权所有 © 2026 年，ShannonBase Project Contributors / Public



图：Rapid 引擎分层架构

三层存储架构

Rapid 将数据组织为三个层级：

- 表（RpdTable）：管理表级元数据、每个 MySQL 索引对应一个 ART 索引、Storage Index 以及该表的 IMCU 集合。支持普通表（Table）与分区表（PartTable）；分区表在加载时按分区并行扫描，并以加载水位（load watermark）避免重复应用已包含在扫描结果中的变更。
- IMCU（In-Memory Column Unit）：逻辑行分组单元，默认每 81,920 行一个 IMCU。每个 IMCU 维护删除位图、逐列 NULL 位图、轻量事务日志（事务 ID/SCN/操作类型）、逐列 Storage Index 与行目录；读写可见性由发布器（PublishFrontier）保证——只有连续完成写入的前缀行对读者可见。
- CU 层（列单元）：每个 IMCU 中每列对应一个 CU，槽位宽度为归一化后的内部表示宽度（字符串列为 4 字节字典 ID）。列编码由列注释声明：RAPID_COLUMN=ENCODING=SORTED|VARLEN|BITPACK；未声明时使用普通槽位加

LZ4/ZSTD 块压缩（压缩收益低于 10% 时保持未压缩）。字符串列使用本地字典编码，BLOB/TEXT/JSON/VECTOR 等变长值存放在共享变长池中。

- **ART 索引：每个 MySQL 键编译为一棵 Adaptive Radix Tree。键按列类型选择编码：KEY_IMAGE（仅等值）、SORTABLE_VALUE（数值/十进制保序）或 COLLATION_WEIGHT（字符串按排序规则权重保序）；只有所有键部分都保序时才宣告范围扫描，否则仅用于点查。索引并发由树级读写锁与版本号保护，游标在版本变化时按键重新定位。**
- **位打包列（BITPACK）：整数列可按值域以 1–32 位编码存储，支持位打包流上的范围匹配、批量解码以及聚合归约（COUNT/SUM/MIN/MAX 直接对编码归约而不物化列）。位打包流可随检查点持久化；值域超过 32 位时该列保持普通槽位。**

Storage Index（存储索引）

每个 IMCU 都会自动维护列级 zone map（最小值/最大值/NULL 数/NDV 估计），这一技术类似 Oracle 的 Storage Index。

执行期间优化器用这些 IMCU 的统计信息与谓词比对，跳过不可能命中的 IMCU；

表级 ColumnStatistics 还提供等高直方图、HyperLogLog NDV、Pool 采样与分位数，用于选择率与代价估算。

任何 UPDATE/DELETE 都会使受影响的 zone map 失效（invalidate_pruning），以保证剪枝始终是保守的；

MVCC 与事务一致性

Rapid 的 MVCC 由两部分组成：IMCU 的轻量事务日志记录事务标识与提交 SCN，CU 的列级版本链保存 UPDATE 前的列数据。可见性判定使用与源语句相同的 InnoDB ReadView（以及已提交 binlog 视图的水位），而不是仅凭 SCN 比较；未提交的事务不会对任何读者可见。GC 按最小活跃 SCN 清理不再可达的旧版本；删除行以删除位图标记，并在整个 IMCU 全可见时走 O(1) 快速路径。

向量化执行与 SIMD 加速

Rapid 使用批式（向量化）执行流水线：扫描、过滤、哈希连接、嵌套循环、聚合、排序、窗口、LIMIT、物化与结果适配器都以列块（ColumnChunk）为单位处理数据，并尽可能把表达式与连接保留在批处理路径上。字典编码列可在编码 ID 上直接做等值/不等值与 LIKE 匹配，位打包列可在编码流上做范围匹配与归约；SIMD 内核通过统一的 ISA 抽象层在 x86 与 ARM 上分发。

两阶段读取：过滤算子先只请求条件列并保留命中的物理行号，待选择确定后再补齐投影列，避免为被过滤掉的行读取全部列；补齐与条件读取使用同一扫描上下文与 ReadView，两者之间不允许重定位游

版权所有 © 2026 年，ShannonBase Project Contributors / Public

标。

并行查询执行（Morsel）

符合条件的扫描（Scan）、过滤（filter）与哈希连接（Hash join）流水线可以按 morsel 并行执行：ParallelQueue 以 4,096 行为粒度从单一原子游标动态领取区间，避免静态均分造成的长尾；工作线程使用独立内部 THD、私有列缓冲与私有表达式缓冲，只读协调者的游标 pin、谓词与 ReadView。

是否并行基于真实可参与计算的行数而非优化器估算值，并由 WorkGovernor 预留槽位：rapid_parallel_workers 为会话级上限（默认 1），rapid_parallel_max_workers 为进程级上限（启动时只读）。

检查点与恢复

开启 rapid_reload_on_restart 后，后台按表生成检查点：快照加 manifest v3（记录表标识、schema 指纹、SCN、已提交 binlog 边界与逐 IMCU 校验和），默认保留两代。启动时先做异步恢复：校验并加载检查点，再重放记录边界之后的已提交 binlog；任何一步失败都丢弃该镜像并从 InnoDB 重新加载。

检查点创建要求无待处理通知、无未提交版本，并在最终校验失败时整代丢弃。恢复完成前该表不会出现在 performance_schema.rpd_table_id 中。

资源治理与可观测性

查询与算子内存以授予（grant）方式管理。排序、哈希、窗口与物化（Materialized）在超出授予时自动溢写（Spill out）；

rapid_resource_isolation 支持 OFF/OBSERVE/SOFT/STRICT_CPU/AUTO 五种模式，从而可以把分析、传播与维护工作放入不同的资源组。

可观测性包括六张 performance_schema.rpd_* 表、13 个状态指标等以及 Debug 构建下每条语句的 RAPID_STATS 算子计数输出。

并行加载

大表通过对 InnoDB 分区（或非分区表的分片）进行多线程并行扫描加载到 Rapid 引擎中。

每个工作线程克隆独立的 handler 游标，拥有自己的 THD 与 TABLE 上下文，仅共享 TABLE_SHARE 元数据；是否并行由 rapid_parallel_load_threshold / rapid_parallel_part_load_threshold 决定，并在资源治理处于限制模式时退回串行模式进行加载。

加载扫描运行在 READ COMMITTED 模式，只读取已提交数据；加载结束会冻结统计信息、刷新 Storage Index，并对 BITPACK 列执行打包。

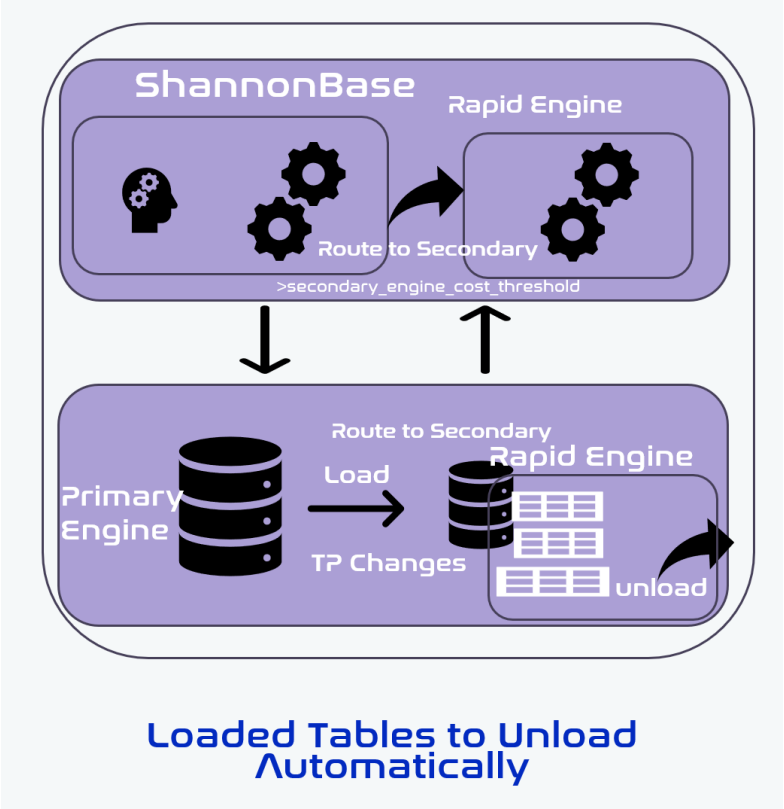
版权所有 © 2026 年，ShannonBase Project Contributors / Public

Autopilot —— 智能自管理内存引擎

ShannonBase Autopilot 是内置于 Rapid 引擎中的自主内存管理子系统。它持续监控查询负载，为每张候选表的访问重要性打分，并自动决定将哪些表加载进或从 Rapid 引擎中淘汰 —— 全程无需 DBA 介入。其指导原则很简单：Rapid 内存池中应始终保存最常用的数据。

```
mysql> show variables like "rapid_self_load%";
+-----------------------------------------------------------+---------+
| Variable_name                                             | Value   |
+-----------------------------------------------------------+---------+
| rapid_self_load_base_relation_fill_percentage | 70      |
| rapid_self_load_enabled                                   | OFF     |
| rapid_self_load_interval_seconds                          | 86400 |
| rapid_self_load_skip_quiet_check                          | OFF     |
+-----------------------------------------------------------+-------+
4 rows in set (0.01 sec)
```

版权所有 © 2026 年，ShannonBase Project Contributors / Public



传统的列式加速系统需要管理员手动指定要加载哪些表，这会随着负载的演变导致配置过时、在冷数据上浪费内存，并在数据量超出容量预估时引发加载失败。Autopilot 通过持续的、由负载驱动的自动化机制消除了所有这些失效模式。

表重要性评分

Autopilot 决策机制的核心是一个按表维护的重要性评分，该评分会在每次 SQL 查询执行后更新。当一条查询完成时，系统会识别该查询所涉及的所有表，并使用以下模型更新每张表的重要性评分：

```
importance_delta = (table_size / total_query_size) × query_execution_time × engine_weight
new_importance = old_importance × (1 − α) + importance_delta × α
```

有两个设计选择使该模型在实践中更加稳健。首先，引擎权重因子区分了路由至 InnoDB 主引擎的查询（权重 = 1.0，表明如果启用 Rapid 加速本可获益）与已经在 Rapid 二级引擎上执行的查询（权重 = 0.5，因为更短的执行时间否则会低估其重要性）。其次，指数加权移动平均（EWMA）平滑机制可以防止短暂的访问高峰导致内存布局出现不必要的抖动。

基于时间的重要性衰减

Autopilot 在每个调度周期都会对所有重要性评分应用指数时间衰减。衰减因子按照自该表最近一次被查询以来经过的天数施加：

```
decayed_importance = current_importance × DECAY_FACTOR ^ days_since_last_query
```

一旦某个评分衰减到低于最小阈值，就会被置零，从而将该表标记为冷数据候选。重要性归零且超过 COLD_TABLE_DAYS 未被查询的表，会在下一周期的冷表卸载阶段被自动从 Rapid 内存中淘汰，从而为更热的数据释放内存。

四阶段自动加载算法

在每个调度周期中，Autopilot 会依次执行四个阶段：

1. 重要性衰减：对所有被跟踪的表应用基于时间的指数衰减，反映访问相关性随时间自然衰退的过程。
2. 冷表淘汰：识别重要性已归零且最近查询时间超过冷数据阈值的自管理表。在做出加载决策之前主动淘汰这些表以回收内存。
3. 优先级队列构建：构建一个最大优先级加载队列（按重要性降序排列，包含重要性 > 0 的未加载表）以及一个最小优先级淘汰队列（按重要性升序排列，包含作为置换候选的已加载自管理表）。
4. 内存感知的加载/淘汰执行：从加载队列中取出重要性最高的候选表。如果当前 Rapid 内存使用量加上该候选表的估算大小仍在配置的内存阈值之内，则立即加载；如果内存不足，则先从淘汰队列中淘汰重要性最低的表，然后重试。该过程持续进行，直到加载队列耗尽或内存阈值被完全占用。

内存阈值管理

Autopilot 对 Rapid 内存池强制执行一个可配置的填充上限：

```
memory_threshold = rapid_memory_size_max × self_load_base_relation_fill_percentage ÷ 100
```

默认填充比例为 70%，为 Rapid 内存池保留 30% 作为缓冲区，用于用户主动发起的手动加载、变更传播写缓冲区以及临时查询执行内存。这一预留空间可防止 Autopilot 的后台活动抢占前台分析查询所需的工作内存。

版权所有 © 2026 年，ShannonBase Project Contributors / Public

系统静默检测

为防止后台加载与高峰时段的 OLTP 与 HTAP 流量相互竞争，Autopilot 会在每个调度周期执行前进行系统静默检测。如果任何被监控的表在过去 QUERY_QUIET_MINUTES 分钟内曾被查询，或任何表从 InnoDB 到 Rapid 的变更传播仍在进行中，则系统被视为处于活跃状态。若系统处于活跃状态，Autopilot 会以可配置的间隔（QUIET_WAIT_SECONDS）等待，最多尝试 MAX_QUIET_WAIT_ATTEMPTS 次，之后完全跳过本轮周期，等待下一个调度间隔。

对于离线或批处理型部署，可通过设置 rapid_self_load_skip_quiet_check = ON 来绕过此机制。

用户加载保护

Autopilot 严格区分两种表加载来源—— USER（由管理员通过 ALTER TABLE ... SECONDARY_LOAD 显式加载）与 SELF（由 Autopilot 自动管理）。

当淘汰算法选中一张 USER 加载的表作为置换候选时，Autopilot 不会将其淘汰。相反，它会将该表的加载类型降级为 SELF，同时保留其常驻内存，并发出警告日志通知 DBA。要彻底淘汰一张用户加载的表，必须使用显式的 SECONDARY_UNLOAD 命令。这一保护机制确保 DBA 有意做出的加载决策绝不会被静默覆盖。

后台协调线程

Autopilot 以专用后台线程（self_load_coordinator）的形式运行，具备完整的生命周期管理。调度间隔由 rapid_self_load_interval_seconds 参数控制（默认值：86400 秒 / 24 小时），使管理员能够将 Autopilot 周期安排在业务低峰时段。线程生命周期的转换是原子性的，并通过条件变量进行协调，确保在不阻塞 MySQL 服务器前台线程池的前提下实现干净的启动与关闭。

元数据自举

启动时，Autopilot 扫描四张系统目录表，构建其对所有候选表的完整视图，无需任何外部配置：

被扫描的系统表	提取的信息
mysql.schemata	所有用户数据库的 Schema ID → schema 名称映射
mysql.table_stats	行数、数据大小、索引大小 → 估算每张表的内存占用
information_schema (DD API)	每张表的 InnoDB 内部表 ID（se_private_id）

版权所有 © 2026 年，ShannonBase Project Contributors / Public

mysql.tables	主引擎 = InnoDB 过滤条件；SECONDARY_ENGINE = RAPID 标志；分区检测

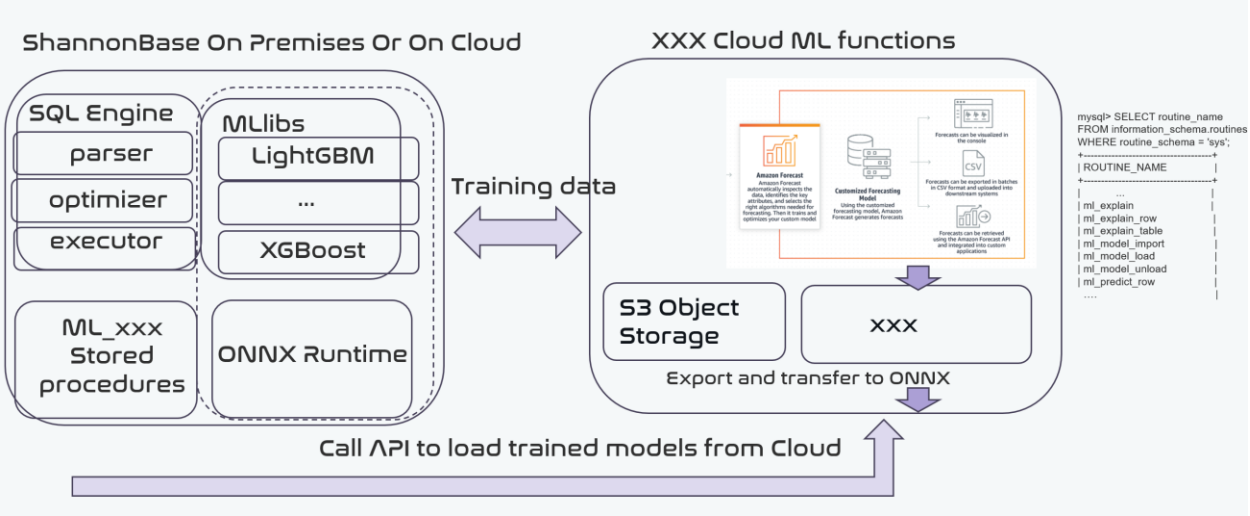
自举完成后，镜像注册表（RpdMirror::Registry，performance_schema.rpd_mirror）通过 DDL 事件钩子保持最新：CREATE TABLE 会新增一个条目，DROP TABLE 会移除对应条目，ALTER TABLE 会更新二级引擎分配状态。

Autopilot 能力总览

Autopilot 子能力	实现机制
热表检测	按查询计算重要性评分：表大小占比 × 执行时间 × 引擎权重，并做 EWMA 平滑
冷数据淘汰	指数时间衰减 + 冷数据时长阈值 → 自动从 Rapid 内存中淘汰
内存感知的加载决策	填充比例上限；加载任何表前先进行内存预检查
优先级驱动的内存置换	最大堆加载队列 + 最小堆淘汰队列；优先置换重要性最低的表
防止 OLTP 干扰	系统静默检测：若近 N 分钟内有表被查询或变更传播仍在进行，则跳过本轮周期
用户加载保护	尝试淘汰时将 USER 类型表降级为 SELF 类型；绝不静默移除
Schema 感知	启动时从 mysql.* 系统表自举；DDL 事件钩子保持实时一致性
可配置调度	调度间隔、填充比例、静默检测与静默等待均可通过系统参数配置

AutoML —— 数据库内机器学习

ShannonBase AutoML 提供了一套完全集成、可通过 SQL 访问的机器学习流水线，能够在数据库引擎内完成模型训练、生成预测与可解释性分析 —— 无需将数据抽取到外部 ML 平台，终端用户无需 Python 依赖，除数据库实例本身外也不产生额外成本。



数据库中机器学习面临的现有挑战

使用数据库数据进行机器学习的传统方式通常要求用户执行以下操作：

- 通过 ETL 将训练数据从 OLTP 数据库导出
- 将数据加载到独立的数据科学环境中（Jupyter、Spark、SageMaker）
- 选择算法、进行特征工程、调优超参数并验证模型
- 将训练好的模型部署到独立的推理服务上
- 编写自定义集成代码，以便从应用 SQL 中调用该推理服务

这一多步骤流程带来了数据延迟、成倍增加的基础设施成本、安全暴露风险，并且需要绝大多数数据库团队并不具备的深厚机器学习专业知识。

AutoML 实现方式

ShannonBase AutoML 消除了所有这些步骤。训练数据直接来自 InnoDB 或 Rapid 表。整个机器学习流水线 —— 数据预处理、算法选择、特征选择、超参数优化、解释器训练以及模型持久化 —— 全部在数据库引擎内针对实时数据集执行。

训练好的模型存储在用户 schema 中（ML_SCHEMA_<username>.MODEL_CATALOG），并参与标准的数据库备份、恢复与访问控制机制。

能力	描述
全自动化流水线	算法选择、特征工程、超参数调优与解释器训练全部实现端到端自动化

SQL 接口	sys.ML_TRAIN / ML_MODEL_LOAD / ML_MODEL_UNLOAD / ML_MODEL_IMPORT / ML_MODEL_EXPORT / ML_MODEL_ACTIVE / ML_SCORE / ML_PREDICT_TABLE / ML_EXPLAIN / ML_EXPLAIN_TABLE，以及原生 ML_MODEL_* 函数
支持的任务类型	分类、回归、时间序列预测、异常检测（PCA/GLOF/GKNN）、推荐、主题建模
核心引擎	LightGBM 训练；ONNX Runtime 推理与外部 ONNX 模型导入
可解释性	为每一次预测生成特征归因评分，通过集成的解释器模型实现
模型元数据	MODEL_CATALOG 记录任务、目标列、训练表、列名、模型质量与训练漂移指标等元数据
安全性	模型与数据默认留在实例内；仅显式配置云 provider 时提示词发送到所选服务
成本	无额外费用；随 ShannonBase 实例一同提供

使用示例

仅需标准 SQL 即可完成分类模型训练并生成预测：

```
-- 训练
SET @model = 'census_model';
CALL sys.ML_TRAIN('shannon_ml.census_train', 'revenue',
                  JSON_OBJECT('task','classification'), @model);

-- 加载并打分
CALL sys.ML_MODEL_LOAD(@model, NULL);
CALL sys.ML_SCORE('shannon_ml.census_test', 'revenue', @model,
                  'balanced_accuracy', @score, NULL);

-- 行级与表级预测
SELECT sys.ML_PREDICT_ROW(JSON_OBJECT('age',41,'education','Bachelors'),
                          @model, NULL);
CALL sys.ML_PREDICT_TABLE('shannon_ml.census_test', @model,
                          'shannon_ml.census_pred', NULL);
```

版权所有 © 2026 年，ShannonBase Project Contributors / Public

训练与推理性能

AutoML 的训练吞吐量取决于数据规模与特征维度。由于训练数据直接从 InnoDB 或 Rapid 读取，消除了导出与加载环节；算法选择、特征工程、超参数搜索与解释器训练均在实例内完成。

在线推理方面，ML_PREDICT_ROW() 在批量预测路径上缓存已加载的 booster，避免重复加载模型；单行推理通常在毫秒级完成，可与事务查询组合使用。

向量检索与生成式 AI

ShannonBase 原生集成了：向量数据存储、相似度检索、数据库内 Embedding 以及本地 LLM 推理。这些能力全部内置于数据库内核中，这使得检索增强生成（RAG）与语义检索应用能够完全在数据库内构建，而无需部署独立的向量数据库或外部 LLM API。

原生 VECTOR 数据类型

ShannonBase 在 MySQL 类型系统中引入了原生 VECTOR 类型，并提供 TO_VECTOR / STRING_TO_VECTOR、FROM_VECTOR / VECTOR_TO_STRING、VECTOR_DIM 以及 DISTANCE(v1, v2, 'COSINE'|'DOT'|'EUCLIDEAN'|'MANHATTAN')函数。

向量类型的列，一起按批式扫描求值；相似度检索以距离表达式加列式扫描实现。

```
CREATE TABLE embeddings (
   id INT PRIMARY KEY,
   description TEXT,
   embedding VECTOR(10)) secondary_engine=rapid;


INSERT INTO embeddings (id, description, embedding)
VALUES (1, 'Example text', TO_VECTOR("[0.12, -0.34, 0.56, 0.78, -0.91, 0.23, -0.45, 0.67, -0.89, 1.23]"));


SELECT LENGTH(embedding), FROM_VECTOR(embedding) FROM embeddings WHERE id = 1;
SELECT DISTANCE(STRING_TO_VECTOR("[1.01231, 2.0123123, 3.0123123, 4.01231231]"),
STRING_TO_VECTOR("[1, 2, 3, 4]"), "COSINE");
SELECT DISTANCE(STRING_TO_VECTOR("[1.01231, 2.0123123, 3.0123123, 4.01231231]"),
STRING_TO_VECTOR("[1, 2, 3, 4]"), "DOT");
```

版权所有 © 2026 年，ShannonBase Project Contributors / Public

```
SELECT DISTANCE(STRING_TO_VECTOR("[1.01231, 2.0123123, 3.0123123, 4.01231231]"),
STRING_TO_VECTOR("[1, 2, 3, 4]"), "EUCLIDEAN");
```

数据库内向量嵌入生成（Embedding）

sys.ML_EMBED_ROW 与 sys.ML_EMBED_TABLE 使用 ONNX Runtime 在数据库进程内执行嵌入(Embedding)推理。model_id 即：模型名（例如 multilingual-e5-small、all-MiniLM-L12-v2）；模型默认位于 <basedir>/llm-models，可通过 --llm-dir 指定。文本在执行 Embedding 过程中不会离开数据库进程。

```
SELECT sys.ML_EMBED_ROW('What is artificial intelligence?', JSON_OBJECT('model_id', 'all-MiniLM-L12-v2'));

CALL sys.ML_EMBED_TABLE('sh_embedding.docs.body', 'sh_embedding.docs.embedding',
JSON_OBJECT('model_id', 'all-MiniLM-L12-v2', 'batch_size', 500));
```

数据库内 LLM 推理

ShannonBase 支持通过 ML_GENERATE()在数据库进程内执行 LLM 推理：本地后端为 ONNX Runtime（KV cache、CPU/GPU 由宿主探测决定，无需用户干预），也可选择 Ollama，或通过 model_options.provider 使用 MaaS 云服务（deepseek / dashscope / anthropic / qianfan）。

本地 ONNX 模型（如 Qwen3.5-2B-ONNX）不依赖外部 API；选择 MaaS 云服务时提示词（Prompt）会发送到所选服务，需由用户显式配置。（可能会造成上下文中的敏感数据泄露）

```
SELECT sys.ML_GENERATE(
  "What is ShannonBase, gvie me some explaination in common?",
  JSON_OBJECT(
    "task", "generation",
     "model_id", "Qwen3.5-2B-ONNX",
    "language", "en",
    "temperature", 0.8,
     "top_p", 0.9,
```

```
        "repeat_penalty", 1.1,
      "max_tokens", 256
   )
);
SELECT sys.ml_generate(
       "Contact John Doe at john.doe@example.com or call 555-867-5309.",
       JSON_OBJECT("task", "pii_detect", "model_id", "multilang-pii-ner-ONNX", "output_format", "json")
     ) AS pii_report;
```

更多信息请参阅相关文档。

RAG 架构

一套完整的检索增强生成流水线可以完全通过 ShannonBase SQL 整个 RAG 和推理过程：

1. 将源文档存储为表中的行
2. 使用 ML_EMBED() 生成并存储向量 Embedding 值
3. 使用 SQL 中的 VECTOR 距离函数执行相似度检索
4. 将检索到的上下文与 ML_GENERATE()结合，生成有依据的 LLM 响应

```
INSERT INTO demo_embeddings (segment, text, document_name, segment_number, metadata)
VALUES
('What is AutoML?', ('AutoML explanation example text'), 'doc1', 1, JSON_OBJECT('category', 'AI')),
('AutoML automates model building', ('AutoML automates model building'), 'doc1', 2,
JSON_OBJECT('category', 'AI')),
('Machine learning basics', ('Basics of machine learning'), 'doc2', 1, JSON_OBJECT('category', 'ML')),
('ShannonBase is an AI-native SQL engine', ('ShannonBase overview'), 'doc3', 1, JSON_OBJECT('category',
'DB'));

CALL sys.ML_EMBED_TABLE('sh_rag.demo_embeddings.text', 'sh_rag.demo_embeddings.embedding',
JSON_OBJECT('model_id','all-MiniLM-L12-v2','batch_size',500));

SET @options = JSON_OBJECT(
```

```
    'vector_store', JSON_ARRAY('sh_rag.demo_embeddings'),
    'n_citations', 2,
    'embed_model_id', 'all-MiniLM-L12-v2',
    'vector_store_columns', JSON_OBJECT(
        'segment', 'segment',
        'segment_embedding', 'embedding',
        'document_name', 'document_name',
        'metadata', 'metadata',
        'segment_number', 'segment_number'
    )
);
```

由于以上四个步骤均在同一数据库引擎内执行，因此不存在进程间通信、数据序列化开销，也没有外部 API 速率限制。整个 RAG 流水线都能受益于 Rapid 的向量化执行与并行扫描能力。

JavaScript 多语言执行环境（MLE）

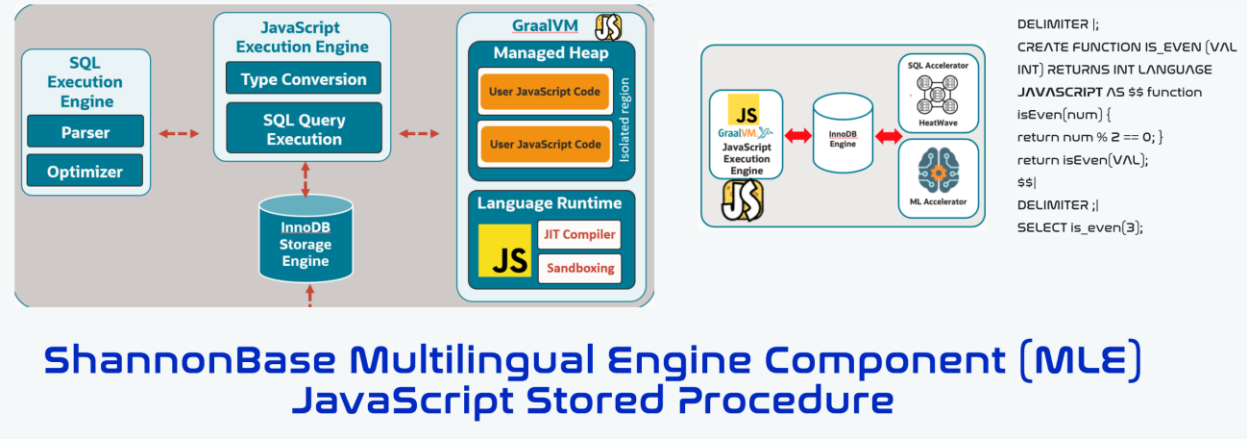
ShannonBase 通过内置 JerryScript 引擎实现数据库内 JavaScript 执行，用户可以用 JavaScript 编写存储过程与函数，并在其中直接执行 SQL、发送结果集。这满足了让复杂业务逻辑、数据预处理、特征工程与 AI 后处理贴近数据的需求。

维度	JavaScript MLE 对比 C/C++ UDF
开发成本	低（JavaScript）对比 高（C/C++，需具备 MySQL 内核知识）
安全性	高 —— 沙箱执行环境，对比低 —— 直接的进程级访问
热加载	修改后重建例程即可生效（无需重新编译或重启服务器），对比 需要重新加载服务器插件
复杂逻辑	原生支持字符串/JSON/正则表达式，对比 需手动实现
AI 集成	可直接访问 ML_EMBED、ML_GENERATE、向量运算，对比需要自定义 C 绑定

JavaScript MLE 参与 ShannonBase 统一的执行上下文。JavaScript 函数可以访问 InnoDB 事务数据、发起 Rapid 分析查询、调用 ML_EMBED()与 ML_GENERATE()，并通过 exec_sql() 执行 SQL、通过 send_result_set() 直接向前端返回结果集。

```
DELIMITER |;
CREATE FUNCTION IS_EVEN (VAL INT) RETURNS INT
LANGUAGE JAVASCRIPT AS $$
function isEven(num) {
    return num % 2 == 0;
}
return isEven(VAL);
$$|
DELIMITER ;|

SELECT is_even(3);
```



数据库内的 Native-Agent

ShannonBase Native-Agent 是一个直接内嵌于 ShannonBase 数据库内核中的 AI 原生对话式智能体。它通过单一的、可通过 SQL 访问的自然语言接口，暴露出数据库的全部能力 —— 分析查询、运维诊断、数据变更、知识检索以及多步推理 —— 无需外部中间件、无需额外部

署，也无需修改应用层。

该智能体通过任意 MySQL 兼容客户端都可发起的单个 SQL 函数调用来触发：

```
-- 核心调用
SET @chat_options = JSON_OBJECT('conversation_id', UUID());
CALL sys.shannon_chat('<natural language instruction>');  -- user message
```

sys.shannon_chat 是一个 SQL PROCEDURE；结果以结果集形式返回（包含 response，按路径还可能包含 trace_id 与 citations），而不是通过 AS 别名的标量返回值。会话配置（对话 ID、模型与 RAG 选项、审批策略等）通过@chat_options 会话变量传入。

```
-- 示例
SET @chat_options = JSON_OBJECT('conversation_id', 'session-dba-20240601');
CALL sys.shannon_chat('Show me the top 5 tables by row count in the current database.');
```

设计动机：弥合数据与智能之间的鸿沟

传统数据库要求用户掌握 SQL、理解 schema 结构，并手动编写查询。随着数据库开始承载机器学习模型、向量嵌入（Embedding）以及进程内 LLM 推理，其操作面急剧扩大。

DBA、数据分析师与应用开发者必须同时掌握越来越多的 SQL 原语 —— ML_TRAIN、ML_GENERATE、ML_EMBED、向量距离函数、系统状态表 —— 仅仅是为了从数据平台中提取有价值的数据。

Native-Agent 消除了这一鸿沟。通过将推理智能体直接嵌入内核，ShannonBase 使任何用户都能够使用自然语言，通过单次 SQL 调用与整个数据平台交互 —— HTAP 查询、AutoML 训练、RAG 检索、系统诊断以及事务性变更。

查询执行的专业知识、schema 感知能力与模型调用逻辑都存在于数据库内部，而非应用代码中。当然，用户也可使用插件机制，通过编写 JS 程序来实现满足自身业务逻辑的智能体。

架构

Native-Agent 被实现为一个完全运行于 ShannonBase 进程内部的分层流水线。下表展示

了从自然语言用户消息到基于 SQL 的可靠响应的内部请求生命周期。

层级	组件	作用
入口点	sys.shannon_chat(msg) — PROCEDURE	SQL PROCEDURE —— 唯一的公开接口，以 CALL 方式发起。接收自然语言消息；会话 ID 与其余配置通过 @chat_options 会话变量传入，结果以结果集返回（response，按路径可能含 trace_id 与 citations）。
插件分发	插件发现机制（L1–L4）	按优先级顺序检查会话级、注册表级、约定命名或默认的智能体实现。
请求路由器	意图分类器	先处理待审批指令；其余请求路由至 Route A（目录精确匹配）或 Route D（LLM Agent 循环）。
Schema 检索	ML_RETRIEVE_SCHEMA_METADATA	对预先构建的 schema Embedding 执行语义相似度检索，在不注入完整 DDL 的情况下识别相关的表与列。
SQL 生成	LLM（ML_GENERATE）	针对 InnoDB 或 Rapid 组装并执行 SQL。校验语法；在执行出错时通过纠错提示进行重试。
知识检索	ML_RAG / ML_EMBED	针对用户建立索引的知识表执行向量相似度检索，用于回答知识类查询。
记忆管理器	mysql.agent_memory	以 conversation_id 为键持久化各轮对话（角色、内容、思维链）。每一轮对话都会被加载，用于构建多轮上下文窗口。
写安全	事务守护（Transaction Guard）	在任何 DML 之前强制执行 begin_tx。校验 UPDATE/DELETE 语句是否包含 WHERE 子句。在智能体出现异常时自动回滚。
响应组装	结果格式化器	将 SQL 结果集、RAG 片段与 LLM 生成的文字组合成连贯的自然语言答案，返回给调用方。

版权所有 © 2026 年，ShannonBase Project Contributors / Public

意图路由与执行路径

请求路由器，先检查是否存在待审批指令，然后把消息分为两类执行路径，并在任何 LLM 调用之前完成分类，从而为高频运维类查询提供快速路径：

执行路径	触发信号	机制	是否调用 LLM
A：Catalog 精确匹配	catalog_match() 命中 schema/对象名	直接执行预置 SQL，经统一工具守卫；不调用 LLM	否
D：LLM Agent 循环	其余所有请求	工具调用循环（最多 10 轮）：schema/知识上下文 → LLM → 工具 → 安全网	是
待审批恢复	消息含 Approve / Reject / Modify:SQL	恢复审批状态机，按步推进计划（CAS 保护）	否（仅执行已批准步骤）

Catalog 快速路径查询完全绕过 LLM，通过直接查询预先已知的系统视图，在毫秒内返回结果。这对于响应延迟至关重要的 DBA 监控工作流尤为有价值。

对话记忆与多轮上下文

Native-Agent 将对话状态持久化在 mysql.agent_memory 系统表中。每一轮对话都以一行记录的形式存储，包含角色（用户或助手）、消息内容、内部推理链（thought）以及创建时间戳，以用户提供的 conversation_id 为键；此外还有 agent_memory_audit、agent_sql_trace、agent_review_history、agent_tx_lease、agent_policy 与 shannon_agent_plugins 等表记录审计、审批、事务租约与插件注册。

在每一轮对话中，智能体会从 agent_memory 加载最近的 N 轮对话（默认值：5，对应 memory_options.short_term.recent_turns，可通过 chat_options 配置），用于构建传递给 LLM 的上下文窗口。这使得连贯的多轮对话成为可能，后续问题可以通过自然语言引用前文结果。

```
SET @chat_options = JSON_OBJECT('conversation_id', UUID());
```

版权所有 © 2026 年，ShannonBase Project Contributors / Public

```
CALL sys.shannon_chat('How many rows are in the orders table?');          -- q1
CALL sys.shannon_chat('Break that down by status.');                      -- q2
CALL sys.shannon_chat('Which status had the highest average value?');     -- q3
CALL sys.shannon_chat('Now show me the monthly trend for that status.');  -- q4
```

智能体将 Q4 中的“that status（该状态）”解析回 Q3 的结果，全程无需任何应用层的状态管理。所有上下文传递都由记忆子系统透明处理。

参数	类型	默认值	作用
history_length	INT	5	纳入 LLM 上下文窗口的历史对话轮数
schema_name	STRING	current DATABASE()	将 schema 向量检索限定在指定 schema 范围内
retrieve_top_k	INT	6	每次知识类查询检索的 RAG 文档片段数量
skip_generate	BOOL	false	为 true 时，若 RAG 检索无结果则跳过 LLM 生成
model_options	JSON	temperature=0.25, max_tokens=1200	LLM 生成参数：temperature、top_p、max_tokens、repeat_penalty
conversation_id	STRING	(延续上次会话)	会话 UUID。省略则自动延续当前连接最近一次对话；设为：new 强制新建会话
model_options.provider	VARCHAR	(本地 ONNX / Ollama)	切换远程 LLM 提供方：deepseek | dashscope | anthropic | qianfan
review_mode	STRING	off	设为 review 后，写操作 / DDL / 高风险 SQL 将暂停并等待人工审批（见"事务写安全"一节）

Schema 感知的 SQL 生成

准确的 SQL 生成要求智能体能够将自然语言引用正确解析为对应的表名与列名。对于表数量较少的数据库，智能体会将完整 DDL 注入 LLM 提示词中。而对于更大的 schema（约超过 20 张表），这种方式会降低提示词质量并增加延迟。

为解决这一问题，Native-Agent 与 ShannonBase 的向量基础设施一起，通过 ML_RETRIEVE_SCHEMA_METADATA 预先构建 schema 的 Embedding 后，智能体会对已完成 Embedding 的 schema catalog 执行语义相似度检索，仅为每次查询获取最相关的表定义。这缩短了提示词长度、提高了 SQL 准确性，并大幅降低了复杂 schema 的 token 消耗：

```
-- 构建 schema 向量索引（一次性操作，大型 schema 约需 1-3 分钟）
CALL sys.ML_RETRIEVE_SCHEMA_METADATA('',@result,
   JSON_OBJECT('schemas', JSON_ARRAY(DATABASE()), 'build_index', true));
-- 验证就绪状态
SELECT COUNT(*) AS indexed_tables
FROM mysql.schema_embeddings
WHERE schema_name = DATABASE() AND status = 1;
```

Schema Embedding 值必须在 DDL 变更（ALTER TABLE、CREATE TABLE、DROP TABLE）后重新构建。

Embedding 值存储在 mysql.schema_embeddings 表中，其参与标准的数据库备份与恢复流程。

事务写安全

Native-Agent 在一个结构化的事务安全框架内，支持由自然语言驱动的数据变更（INSERT、UPDATE、DELETE）。写操作受事务守护（Transaction Guard）强制执行的四项不变量约束：

安全机制	描述
强制事务包裹	所有 DML 语句都在智能体管理的显式 begin_tx / commit_tx 块内发

版权所有 © 2026 年，ShannonBase Project Contributors / Public

安全机制	描述
	出。活动事务之外的写操作将被拒绝。
WHERE 子句强制校验	不带 WHERE 谓词的 UPDATE 和 DELETE 语句会在执行前被自动拒绝，防止意外的全表变更。
异常回滚	如果智能体的处理循环在写序列的任何阶段异常退出，任何未关闭的事务都会在返回响应前被强制回滚。
执行前确认	设置 @chat_options 中的 review_mode="review" 后，写操作 / DDL / 高风险 SQL 会暂停并生成结构化审批提示（计划 ID、SQL、受影响表、预计行数、风险等级），用户回复 Approve / Reject / Modify:<SQL> 后才会继续执行。

用户始终掌握完全的控制权：在写序列的任意阶段发出回滚指令，都会使智能体发出 ROLLBACK 并放弃该操作。

写操作继承了该用户的 MySQL 权限边界；智能体执行权限与 session 用户的权限一致。

插件扩展体系

Native-Agent 支持四级插件发现层级，允许用户与运维人员在不修改内核的情况下，将自定义业务逻辑注入智能体的处理链。插件分发发生在调用默认智能体引擎之前：

优先级	机制	作用范围	配置方式
L1（最高）	会话变量	仅限当前会话	SET @shannon_agent_plugin = 'schema.function'
L2	插件注册表	实例级；支持多插件	CALL sys.shannon_register_plugin(name, schema, func, priority, desc, @result)
L3	约定命名	按 schema 生效，零配置	CREATE FUNCTION shannon_agent(msg TEXT, conv_id VARCHAR(64)) RETURNS TEXT LANGUAGE JAVASCRIPT
L4（兜底）	系统默认	始终可用	无需配置

版权所有 © 2026 年，ShannonBase Project Contributors / Public

插件是通过 ShannonBase MLE 环境执行的 JavaScript 存储过程完成。它们接收原始用户消息与会话 ID，可以在委托（delegate）给默认 agent（sys.shannon_agent_default）之前，用业务特定的上下文（如货币单位、时区规则或访问策略）来完成默认 agent 的替换。

这一架构确保特定领域的定制化永远不需要重新 fork 核心智能体的逻辑代码：

```
-- 插件示例：在调用默认引擎前注入业务上下文
CREATE FUNCTION shannon_agent(
    user_message    TEXT,
    conversation_id VARCHAR(64)
) RETURNS TEXT
SQL SECURITY INVOKER LANGUAGE JAVASCRIPT AS $$
function my_agent(msg, conv_id) {
  function esc(s) { return String(s==null?'':s).replace(/\\/g,'\\\\').replace(/'/g,"\\'"); }
  function query(sql) { try { return JSON.parse(sys.exec_sql(sql)); } catch(e) { return {}; } }
  var enriched = '[Rules] currency_unit=cents; timezone=UTC+8\n' + msg;
  var rows = query("SELECT sys.shannon_agent_default('" + esc(enriched) + "','" + esc(conv_id) + "') AS
r");
  return (Array.isArray(rows) && rows[0]) ? String(rows[0].r || '') : '';
}
return my_agent(user_message, conversation_id);
$$;
```

直接访问 ML 基础能力

对于需要更底层控制的用户，Native-Agent 内部所使用的同一套 ML 基础设施，以 SQL 函数访问获得 ML 能力。

这使得程序化的特征工程、批量推理流水线以及 RAG 构建无需经过对话式接口即可完成：

函数 / 存储过程	签名	描述
sys.ML_GENERATE	ML_GENERATE(prompt TEXT, options	使用任意已加载的 ONNX

版权所有 © 2026 年，ShannonBase Project Contributors / Public

函数 / 存储过程	签名	描述
	JSON)	或远程模型进行文本生成与 LLM 推理
sys.ML_EMBED_ROW	ML_EMBED_ROW(text TEXT, options JSON)	为单条文本生成向量嵌入
sys.ML_EMBED_TABLE	ML_EMBED_TABLE(input_table_column, output_table_column, options)	为整张表的某一文本列批量生成向量嵌入
sys.ML_RAG	ML_RAG(query TEXT, OUT output JSON, options JSON)	检索增强生成：向量检索 + 有依据的 LLM 响应
sys.ML_RETRIEVE_SCHEMA_METADATA	ML_RETRIEVE_SCHEMA_METADATA(q, @out, options)	对 schema 元数据进行语义检索；同时用于构建 schema 向量嵌入索引
DISTANCE / TO_VECTOR / FROM_VECTOR / VECTOR_DIM	DISTANCE(v1, v2, metric)	在标准 SQL 中使用的向量构造函数与距离函数（COSINE/DOT/EUCLIDEAN/MANHATTAN）

与 ShannonBase 架构的集成

Native-Agent 并非一个独立服务 —— 它是 ShannonBase 引擎技术栈中的一等公民，位于 ML & AI 层与 MLE（JavaScript）执行环境之上。

它使用其他子系统作为内部依赖使用：

- Rapid 引擎：智能体生成的分析型查询会经过 HTAP 优化器路由，在有利的情况下卸载至 Rapid 列式引擎，并自动受益于向量化执行、Storage Index 跳读与并行扫描。
- AutoML：智能体可以在多步推理序列中调用 ML_TRAIN、ML_PREDICT_ROW 与 ML_EXPLAIN_TABLE，实现对话式的模型训练与推理工作流。
- 向量检索：智能体的 RAG 路径会调用 ML_EMBED 与相似度函数来完成知识问答。

版权所有 © 2026 年，ShannonBase Project Contributors / Public

- JavaScript MLE：智能体插件以 JavaScript 存储过程的形式编写，由 JerryScript 引擎执行。智能体核心本身（shannon_agent_default）就是用 JavaScript 编写实现，因而其逻辑可以接受审查与扩展。
- InnoDB：支撑对话记忆系统的 mysql.agent_memory 表是一张标准的 InnoDB 表，参与 ACID 事务、MySQL 复制以及标准备份流程。

典型应用场景

应用场景	Native-Agent 的解决方式
临时性分析查询	分析师用自然语言表达问题；智能体从 Rapid 的列式引擎中生成、执行并返回结果，无需具备 SQL 知识。
DBA 运维诊断	诸如“deadlock” “slow queries”或“buffer pool”之类的关键词会触发 Catalog 快速路径查询，在毫秒级时间内返回系统健康数据，该操作不涉及 LLM 推理动作。
多步业务分析	分析师可以跨多轮对话拆解复杂问题；智能体在各轮之间保持上下文，引用先前的结果以构建渐进式分析。
Schema 探索	新加入团队的成员可以询问“有哪些表” “orders 表长什么样”或“customers 与 orders 之间是什么关系”，以对话方式理解数据模型。
受约束的数据变更	运维人员可以用自然语言下达写操作指令；事务守护强制执行安全不变量，同时在 agent_memory 中维护完整的审计轨迹。
领域专属 AI 助手	团队可以部署自定义插件，注入业务术语、访问策略或专属上下文，在不修改内核的情况下打造领域适配的 AI 助手。
RAG 知识库问答	在通过 ML_EMBED_TABLE 对内部文档建立索引后，用户可以提出以专有内容为依据的知识类问题，答案完全在数据库边界内生成。

安全与数据治理

由于 Native-Agent 完全在 ShannonBase 进程内执行，适用于直接 SQL 访问的所有权限体系保证同样适用于智能体生成的查询：

版权所有 © 2026 年，ShannonBase Project Contributors / Public

- 权限强制：智能体生成并执行的所有 SQL 均在所连接 MySQL 用户的凭据与权限下运行。智能体无法访问该用户尚未获得授权的表或操作。
- 默认不向外部传输：嵌入、PII 与本地 LLM 推理均在进程内通过 ONNX Runtime 执行。仅当用户显式选择云 provider（deepseek/dashscope/anthropic/qianfan）时，提示词才会发送到所选服务；这一选择由 chat_options 控制并可被审计。
- 写操作审计轨迹：每一轮对话，包括完整的推理链与所执行的 SQL 语句，都会持久化到 mysql.agent_memory 中，为所有由智能体驱动的操作提供完整的审计日志。
- 生产环境加固：对于敏感数据库，可以使用自定义插件为智能体流水线添加数据脱敏、行级访问过滤或额外的审计日志，而无需修改内核。

Native-Agent 能力总览

能力	实现方式
可通过 SQL 访问的接口	单一的 sys.shannon_chat(msg) PROCEDURE，配合 @chat_options 会话变量；任意 MySQL 客户端均可使用，无需额外驱动
意图分类与路由	词法/语义分类器分发至 Route A（目录精确匹配）或 Route D（LLM Agent 循环），并支持审批恢复
Schema 感知的 SQL 生成	对 ML_RETRIEVE_SCHEMA_METADATA 向量嵌入执行相似度检索；仅获取相关 DDL
多轮对话记忆	mysql.agent_memory InnoDB 表；可通过 chat_options.history_length 配置历史窗口
目录快速路径诊断	预先映射的系统查询完全绕过 LLM；针对监控类关键词实现低于 10ms 的响应
事务写安全	begin_tx 强制执行、WHERE 子句校验、异常回滚、执行前范围确认
RAG 知识问答	ML_EMBED + VECTOR 相似度检索 + ML_GENERATE 生成有依据的响应，全程在数据库内完成

版权所有 © 2026 年，ShannonBase Project Contributors / Public

能力	实现方式
插件可扩展性	四级发现层级；JavaScript MLE 插件；零配置的约定命名机制（L3）
HTAP 查询加速	智能体生成的分析型 SQL 通过 HTAP 优化器透明路由至 Rapid 列式引擎
数据主权	所有推理默认在进程内执行（ONNX Runtime）；仅显式配置云 provider 时提示词发送到所选服务；权限边界始终按连接用户强制执行

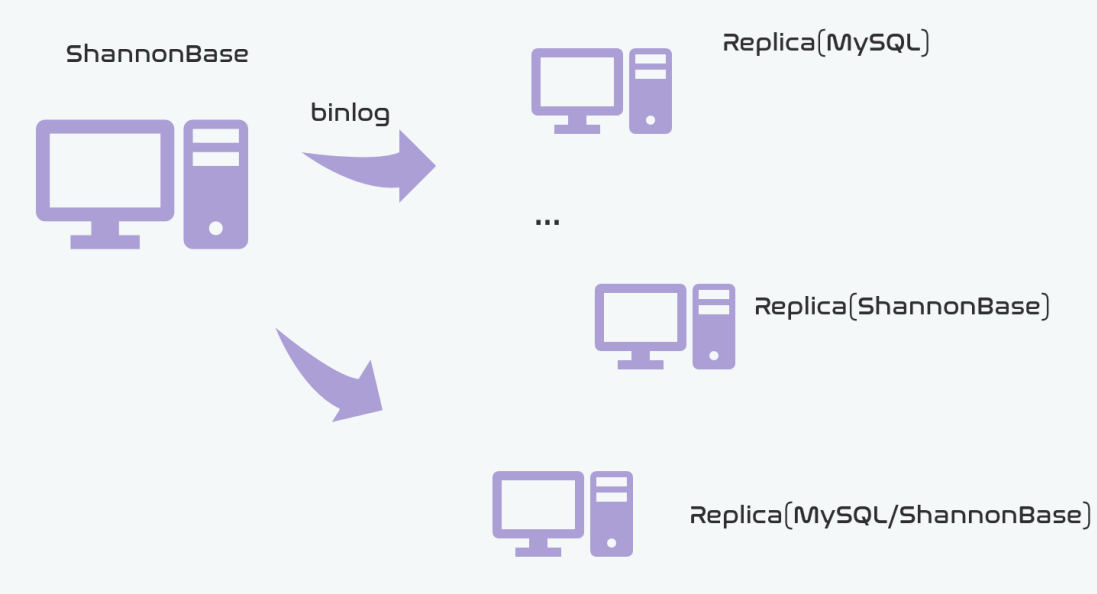
ShannonBase 部署

部署模式

ShannonBase 支持多种部署配置，以适应不同组织的需求：

- **单节点 HTAP：InnoDB 与 Rapid 共处于同一服务器上。Rapid 作为进程内的列式加速器运行。适用于开发、测试以及中等规模的生产负载。**
- **只读副本 / 多实例扩展：ShannonBase 可作为 MySQL 复制拓扑中的只读副本，接收上游 binlog 并在本地维护 Rapid 副本，用于分担分析负载；跨节点共享 Rapid 内存的集群形态（MPP）属于企业版的能力。**
- **MySQL 兼容混合部署：现有 MySQL 部署可以引入启用了 Rapid 的 ShannonBase 作为只读副本，从上游 MySQL 接收基于 binlog 的复制，并在不影响 OLTP 的前提下承担分析型查询。**

版权所有 © 2026 年，ShannonBase Project Contributors / Public



系统参数与可配置性

ShannonBase 提供细粒度的系统参数，用于调优 Rapid 内存分配、并行加载度、变更传播模式、自动加载调度、清理行为以及 GC 触发阈值。主要参数类别包括：

- 内存与资源：rapid_memory_size_max（进程内 Rapid 内存池上限，默认 2G，最大 64G）、rapid_resource_isolation 与 rapid_resource_{ap,sync,maint}_limit、以及 window/sort/materialize 各自的溢写上限。
- 并行：rapid_parallel_workers（会话级，默认 1）、rapid_parallel_max_workers（进程级，启动时只读）、rapid_parallel_load_threshold、rapid_parallel_part_load_threshold 与 rapid_parallel_load_max。
- 变更传播与恢复：rapid_change_propagation_mode（CAPTURE|COMMITTED_BINLOG）、rapid_reload_on_restart 与 rapid_query_sync_timeout_ms。
- 自动加载：rapid_self_load_enabled、rapid_self_load_interval_seconds（默认 86400）、rapid_self_load_skip_quiet_check 与 rapid_self_load_base_relation_fill_percentage（默认 70）。
- GC/清理与观测：rapid_gc_interval_scn、rapid_gc_interval_seconds、rapid_purge_batch_size、rapid_min_versions_for_purge、rapid_purge_efficiency_threshold 与 rapid_max_purger_timeout；配合 performance_schema.rpd_* 与 rapid_* 状态变量使用。

版权所有 © 2026 年，ShannonBase Project Contributors / Public

安全、可观测性与可管理性

- **权限边界：所有 SQL（包括智能体生成的 SQL）都在连接用户的 MySQL 权限下执行；Rapid 不引入独立的授权模型。**
- **数据保护：TLS、静态加密、审计与备份沿用 MySQL/InnoDB 提供的能力；Rapid 只是可重建的内存副本，备份只需覆盖 InnoDB 与 binlog。**
- **推理数据流：嵌入、PII 与本地 LLM 推理默认在进程内完成；只有显式选择云 provider 时才会向该服务发送提示词。**
- **可观测性：performance_schema 提供 rpd_tables、rpd_table_id、rpd_columns、rpd_column_id、rpd_preload_stats 与 rpd_mirror 六张表；状态指标包括 rapid_mempool_*、rapid_pop_*、rapid_query_*、rapid_bg_* 以及 rapid_resource_plan 等渲染器；Debug 构建可用 RAPID_STATS 输出算子级计数。**

ShannonBase 目标负载与应用场景

应用场景

ShannonBase 面向那些需要将多个数据平台层整合为单一 MySQL 兼容系统的组织而设计：

应用场景	所应用的 ShannonBase 能力
实时运营分析	HTAP 路由：对实时数据进行 OLTP 写入与 OLAP 读取，且具备毫秒级传播延迟
实时欺诈检测	通过 ML_PREDICT_ROW() 提供的 AutoML 分类模型，毫秒级推理延迟
客户流失预测	在事务历史数据上进行数据库内回归/分类模型训练，无需导出数据
销售预测	具备自动化季节性与趋势分解能力的 AutoML 时间序列预测
语义文档检索	ML_EMBED() + VECTOR 相似度检索 —— 完整的数据库内 RAG 流水线
推荐系统	AutoML 推荐系统（协同过滤）+ VECTOR 最近邻检索
异常检测	针对无标注运营数据的无监督异常检测（GkNN 算法）
特征工程流水线	用于预处理、转换与推理后处理的 JavaScript MLE 存储过程
具备数据主权的生成式 AI	通过 ML_GENERATE() 实现数据库内 LLM 推理 —— 敏感数据永不离开数据库

版权所有 © 2026 年，ShannonBase Project Contributors / Public

结论

ShannonBase 相较于传统 MySQL 部署与常规 HTAP 方案，代表了一次根本性的架构进步。通过将自研的内存列式引擎（Rapid）、智能自管理内存自动化层（Autopilot）、全自动化的机器学习流水线（AutoML）、原生向量计算、数据库内 LLM 推理以及 JavaScript 多语言执行环境直接嵌入 MySQL 内核，ShannonBase 消除了那种迫使组织必须运维并同步多个专用数据服务的架构碎片化问题。

该系统能够以毫秒级传播延迟对实时事务数据交付实时分析结果；在内部 TPC-H 参考测试中，扫描与聚合类查询相对 InnoDB 行存路径可获得约 1-2 个数量级的加速（具体倍数取决于查询、数据规模与并发），并在数据库安全边界内提供企业级机器学习与生成式 AI 能力——无需导出数据，也无需依赖外部 AP 或 ML 平台。

ShannonBase 已开源，可通过 https://github.com/Shannon-Data/ShannonBase 获取。它专为需要统一、高性能、MySQL 兼容基础平台以应对 AI 时代的数据库内核开发者、数据工程师与 AI 平台架构师而打造。

资源

GitHub 仓库：https://github.com/Shannon-Data/ShannonBase

许可证：GPLv2（兼容 MySQL 的开源许可证）

支持平台：Linux x86-64 与 aarch64（Ubuntu 22.04 / 24.04 / 26.04）；兼容 MySQL 8.4 协议与客户端。

硬件要求：HTAP 负载最低需 8 核 CPU、32GB 内存；生产环境下的 AutoML 与 Rapid 引擎部署建议配置 64GB 以上内存和 GPU

联系我们

Shannon.data.tech@gmail.com，或访问以下网址查找您所在地区的办公室：www.shannonbase.org。

X: https://x.com/ShannonDataAI

版权所有 © 2026 年，ShannonBase Project Contributors / Public