AGENTIC · RAG · v1.0

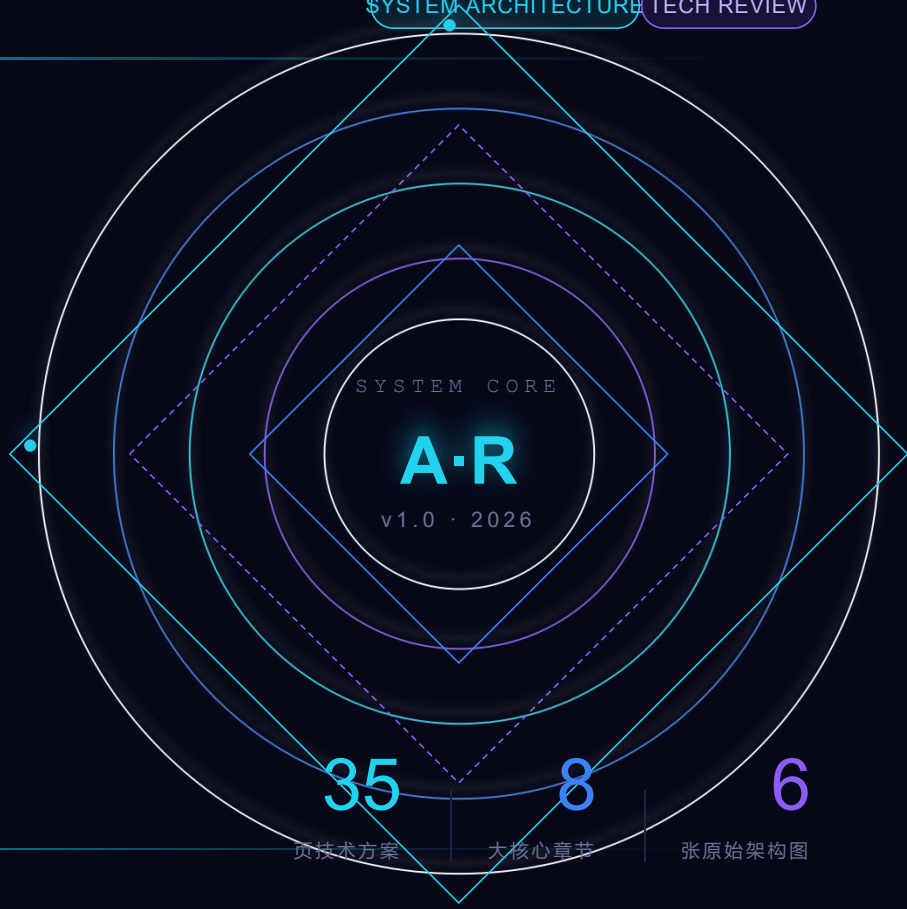
SYSTEM ARCHITECTURE TECH REVIEW

企业级 · ENTERPRISE-GRADE

Agentic RAG System

企业级智能检索增强生成系统 · 完整架构方案

Knowledge · Reasoning · Agent · Generation —— 一体化融合引擎



35 页技术方案

8 大核心章节

6 张原始架构图

2026.06 · 技术评审版

目录 · Table of Contents

8 CHAPTERS · 35 PAGES

本次汇报范围 · Agenda & Scope

总时长 · 60 MIN

60:00

√IN SCOPE 涵盖内容 5 项

01 完整系统架构与分层设计

02 RAG 基础六步与高级六阶段映射

03 StreamChatPipeline 8 阶段流水线详解

04 GraphRAG + DeepSearch 融合机制

05 三阶段落地计划与风险预案

⊖OUT OF SCOPE 不涵盖内容 3 项

01 具体业务场景的 RAG 数据集构建细节

02 模型 fine-tune 实现路径

03 商务报价与采购方案

汇报节奏 · Timeline

4 PHASES

PHASE 01 背景与现状 5min

PHASE 02 架构详解 25min

PHASE 03 关键技术与融合 15min

PHASE 04 落地计划与 Q&A 15min

Q&A 集中在最后阶段

∑ 60 min

CHAPTER

01

背景与目标

背景与目标

Background & Goals · 为什么我们需要 Agentic RAG

从「检索+生成」到「Agent 驱动」的范式跃迁

A paradigm shift from retrieval-augmented to agent-driven intelligence.

RAG 范式演进 · From Naive RAG to Agentic RAG

2022 → 2025+
4 PARADIGMS

2022 · 2023 · 2024 · 2025+

STAGE 01 · 2022
Naive RAG
朴素 RAG

特征
▶ 单次检索 + 直接生成

问题
⚠ 检索召回低、无意图识别、无反思

代表技术
`LangChain` `LlamaIndex`

STAGE 02 · 2023
Advanced RAG
高级 RAG

特征
▶ 检索前后两阶段优化（Query 改写 / Rerank / CRAG）

问题
⚠ 仍依赖固定 Pipeline，缺乏自主决策

代表技术
`RAG-Fusion` `Self-RAG`

STAGE 03 · 2024
Modular RAG
模块化 RAG

特征
▶ 检索/生成/记忆模块解耦，支持路由与多通道并行

问题
⚠ 模块间协同仍需人工编排

代表技术
`LightRAG`

STAGE 04 · 本方案 · 2025+
Agentic RAG
智能体 RAG · 当前阶段

特征
⚡ Agent 驱动 + 多步推理 + 工具调用 + 动态反思

突破
✓ 自主决策 · 短路与多通道并行 · 可演进架构

代表技术
`GraphRAG` `Fusion GraphRAG`

◆ 本方案融合 Advanced RAG + Modular RAG + Agentic 三层范式，构建可演进的统一架构。

系统目标与核心指标 · Goals & KPIs

4 GOALS · 6 KPIS

优先级：准确率 > 延迟 > 扩展 > 治理

GOAL 01 P0

准确率优先 Accuracy First

引用准确率 > 92%，幻觉率 < 5%

GOAL 02 P1

低延迟响应 Low Latency

P95 < 3s（首 Token < 800ms）

GOAL 03 P1

可扩展性 Scalability

水平扩展，支持 100+ 业务知识库

GOAL 04 P2

可观测可治理 Observable & Governable

全链路 Trace + 多租户隔离 + 配额限流

CORE KPI · 核心指标

>92%	<5%	<800ms	<3s	10K+	99.9%
引用准确率 ACCURACY	幻觉率 HALLUCINATION	首 Token 延迟 FIRST TOKEN	P95 完整响应 P95 LATENCY	QPS 峰值 PEAK QPS	SLA 可用性 AVAILABILITY

以上 KPI 是后续 MVP / Beta / GA 三阶段验收的核心依据

CHAPTER

02

RAG 基础与高级六阶段

RAG 基础与高级六阶段

RAG Fundamentals · 从 6 步基础链路到 6 阶段高级 Pipeline

> 先回到 RAG 的本质，再谈如何升级到 Agentic
>
> Back to the fundamentals before evolving to agent-driven intelligence.

基础 RAG 六步链路 · Naive RAG Pipeline

6 STAGES · LINEAR FLOW
Ingest → Generate

```
Answer = LLM( Query, Retrieve( Index, Embed( Chunk( Ingest( Data ) ) ) ) )
```

01 Ingest
导入

从各种来源导入原始数据（PDF / Word / HTML / DB）

TYPICAL TOOL
Tika

›

02 Chunk
分块

将文档拆分为更小的块（按字符 / 语义 / 标题）

TYPICAL TOOL
LangChain TextSplitter

›

03 Embed
向量化

将文本块转换为向量表示

TYPICAL TOOL
bge-m3

›

04 Index
索引

将向量存储到向量索引中

TYPICAL TOOL
Milvus

›

05 Retrieve
检索

从索引中检索相关文本块（带相似度分数）

TYPICAL TOOL
pgvector

›

06 Generate
生成

利用检索到的上下文生成答案

TYPICAL TOOL
GLM-4

⚠ 常见误解澄清 | ① 检索的是向量 embedding 而非原始文本 ② Chunk 在 Embed 之前完成 ③ Index 存的是向量 + 原文，不只是向量

高级 RAG 六阶段总览 · Advanced RAG 6 Phases

6 PHASES · BENTO VIEW

Query Construction → Translation → Routing → Indexing → Retrieval → Generation

01 PHASE

Query Construction
查询构造

- Text-to-SQL
  自然语言 → SQL / PGVector
- Text-to-Cypher
  → 图数据库 Cypher
- Self-query Retriever
  自动生成 metadata 过滤器

02 PHASE

Query Translation
查询翻译

- Query Decomposition
  Multi-query / Step-back / RAG-Fusion 分解或重述问题
- HyDE
  生成假想文档提升检索匹配

03 PHASE

Routing
路由

- Logical Routing
  LLM 基于问题选择数据库
- Semantic Routing
  Embedding + 选择最相似 Prompt

04 PHASE

Indexing
索引

- Semantic Splitter
  优化 chunk 大小
- Multi-representation
  摘要 → 父文档
- Specialized Embeddings
  fine-tune / ColBERT
- RAPTOR
  分层摘要树

05 PHASE

Retrieval
检索

- Re-Rank
  RankGPT / RAG-Fusion 按相关性排序
- CRAG
  文档精炼压缩
- Active Retrieval
  不相关时重新检索或扩展到 Web

06 PHASE

Generation
生成

- Self-RAG / RRR
  用生成质量反馈问题重写或重检索
- Active Retrieval
  生成时动态补充检索

本方案的 StreamChatPipeline 即在此六阶段基础上，融合 Agent 决策 · 多通道并行 · 短路机制，演进为 Agentic 形态。

Phase 1-2 · 查询构造与翻译详解 · Query Construction & Translation

5 SUB-MODULES · 2 COLUMNS

⚒PHASE 01 Query Construction · 查询构造

3 SUB-MODULES

→ 将自然语言问题转换为特定数据库的可执行查询

① Text-to-SQL

SUB 01

FLOW	Question→LLM→SQL
适用	关系型数据库
技术	DIN-SQL / CODES
本方案	用于结构化业务数据查询

② Text-to-Cypher

SUB 02

FLOW	Question→LLM→Cypher
适用	图数据库 Neo4j
本方案	用于 GraphRAG 的实体关系查询

③ Self-query Retriever

SUB 03

FLOW	LLM 自动从问题抽取 metadata filter
适用	向量库带属性过滤
本方案	用于知识库多标签筛选

⇄PHASE 02 Query Translation · 查询翻译

3 SUB-MODULES

→ 重写或分解问题以提升检索召回与精度

① Multi-query

✓ 本方案已落地

机制	LLM 生成多个语义等价问题
流程	并行检索后融合
本方案	Stage 2 已实现

② Step-back

SUB 02

机制	抽象出更高层概念问题
流程	先答宏观再答细节

③ HyDE

可选增强

机制	LLM 先编一个假想答案文档
流程	用其 embedding 检索真实文档
适用	问题与文档措辞差异大的场景

◆ 本方案在 StreamChatPipeline Stage 2 实现 Query 改写与拆分，失败时规则降级；HyDE 作为可选增强项。

Phase 3-6 · 路由 / 索引 / 检索 / 生成 详解 · Routing · Indexing · Retrieval · Generation

2×2 · 4 PHASES · 11 STRATEGIES

PHASE 03 Routing · 路由

2 STRATEGIES

- **Logical Routing**
  LLM 基于问题语义选择目标 DB（关系 / 图 / 向量）
  → 本方案 Stage 3 意图识别即 Logical Routing 的实现
- **Semantic Routing**
  问题 Embedding 后与多个 Prompt 模板 embedding 计算相似度，选最相似模板
  → 适合多场景 Prompt 切换

PHASE 04 Indexing · 索引

4 STRATEGIES

- **Semantic Splitter**
  按语义边界切分而非固定字符
  → 本方案默认策略
- **Specialized Embeddings**
  领域 fine-tune 或 ColBERT
  → 后续 vLLM 扩展支持
- **Multi-representation**
  子块检索 + 父块返回
  → 本方案 Stage 6 已支持
- **RAPTOR**
  分层摘要树，支持跨层级检索
  → 适合长文档深度问答

PHASE 05 Retrieval · 检索

3 ENHANCEMENTS

- **Re-Rank**
  RankGPT / BGE-Reranker 对 Top-K 重排
  → 本方案 Stage 6 默认开启
- **CRAG**
  检索结果质量评估，不相关时压缩或重检索
- **Active Retrieval**
  检索为空时自动扩展 Web 搜索
  → 本方案 Stage 6 短路 C 触发

PHASE 06 Generation · 生成

2 ENHANCEMENTS

- **Self-RAG**
  生成时模型自评是否需要再检索
  → 本方案 Stage 8 动态温度策略参考
- **RRR（Re-Retrieve-Reflect）**
  生成失败时反馈到 Query 重写 + 重检索

四阶段构成本方案 StreamChatPipeline Stage 3-8 的理论根基；实际实现融合了多种策略组合。

CHAPTER

03

系统总体架构

系统总体架构

System Architecture · 六层架构 + 四层模块分层

自顶向下 · 自底向上 · 端到端贯通

Top-down · Bottom-up · End-to-end coherent architecture.

第三章 · 系统架构 END-TO-END STACK

六层系统架构 · End-to-End Stack

Java 17 + Spring Boot 3.5 + React 18 全栈技术体系

TOTAL LAYERS
06

LAYER 01 · BLUE
Frontend
前端

- Chat 对话界面 — React 18 / TypeScript / Vite 5 / Tailwind CSS
- Admin 管理后台 — Radix UI / Zustand / React Router / Axios

LAYER 02 · GREEN
API & Streaming
接口与流式

- RESTful API — Spring Web
- SSE 流式推送 — 实时 Token 输出
- 认证鉴权 — SA-Token

LAYER 03 · PURPLE
RAG Pipeline
检索增强流水线

1 会话记忆加载 › 2 Query 改写与拆分 › 3 意图识别与分类 › 4 歧义引导判断 › 5 多通道并行检索 KB 向量 + MCP 工具 › 6 Prompt 组装与流式生成

LAYER 04 · ORANGE
Platform Services
平台服务

- 知识库管理
- 文档 ETL Pipeline — Tika 解析
- 会话记忆管理 — 摘要压缩
- 模型路由与容错 — 优先级调度 + 熔断降级
- 全链路 Trace 追踪
- 并发控制与限流

LAYER 05 · CYAN
Infra-AI
AI 基础设施

- ① LLM 多模型 — 通义千问 BaiLian + SiliconFlow GLM-4 + Ollama 本地 · 优先级路由 + 自动降级
- ② Embedding & Rerank — 向量化模型 + 语义重排序模型
- ③ MCP 协议 — MCP SDK 1.1 · 天气 / 销售数据 / 工单工具

LAYER 06 · GRAY
Storage & Infra
存储与基础设施

- PostgreSQL + pgvector
- Milvus 2.6
- Redis + Redisson
- RocketMQ
- S3 对象存储

四层模块分层 · Module Layering

接入层 · 应用层 · 基础设施层 · 资源层 — 清晰的职责边界

LAYERS 04

L1 · BLUE 接入层 Access Layer

外部入口

- **frontend**
  前端应用（React + Vite + TS）
- **mcp-server**
  MCP 扩展模块（外部工具集成）

frontend → bootstrap · 虚线　　mcp-server → bootstrap · 实线

L2 · VIOLET 应用层 Application Layer

业务编排

- **bootstrap**
  启动入口 · 配置 · 拦截器

bootstrap → framework · 实线　　bootstrap → infra-ai · 实线蓝

L3 · ORANG 基础设施层 Infrastructure Layer

通用框架与 AI 能力

- **framework**
  通用框架
- **infra-ai**
  AI 基础设施层（Chat / Embedding / Rerank / 模型路由）

framework → resources · 虚线　　infra-ai → resources · 虚线紫

L4 · CYAN 资源层 Resource Layer

基础资源

- **resources**
  基础资源层（Docker Compose / S3 / 配置）

设计原则

4 PRINCIPLES

四大架构准则贯穿全栈分层

01 **单向依赖**
上层只依赖下层，禁止反向

02 **接口隔离**
层间通过接口通信，便于替换实现

03 **资源无感知**
业务层不直接操作 DB / Storage

04 **AI 能力下沉**
infra-ai 封装所有模型调用细节

实线 = 直接依赖　　虚线 = 间接 / 异步依赖

技术选型矩阵 · Technology Stack

11 个维度 · 20+ 组件 · 全栈开源 + 主流云原生

DIMENSIONS 11　COMPONENTS 20+

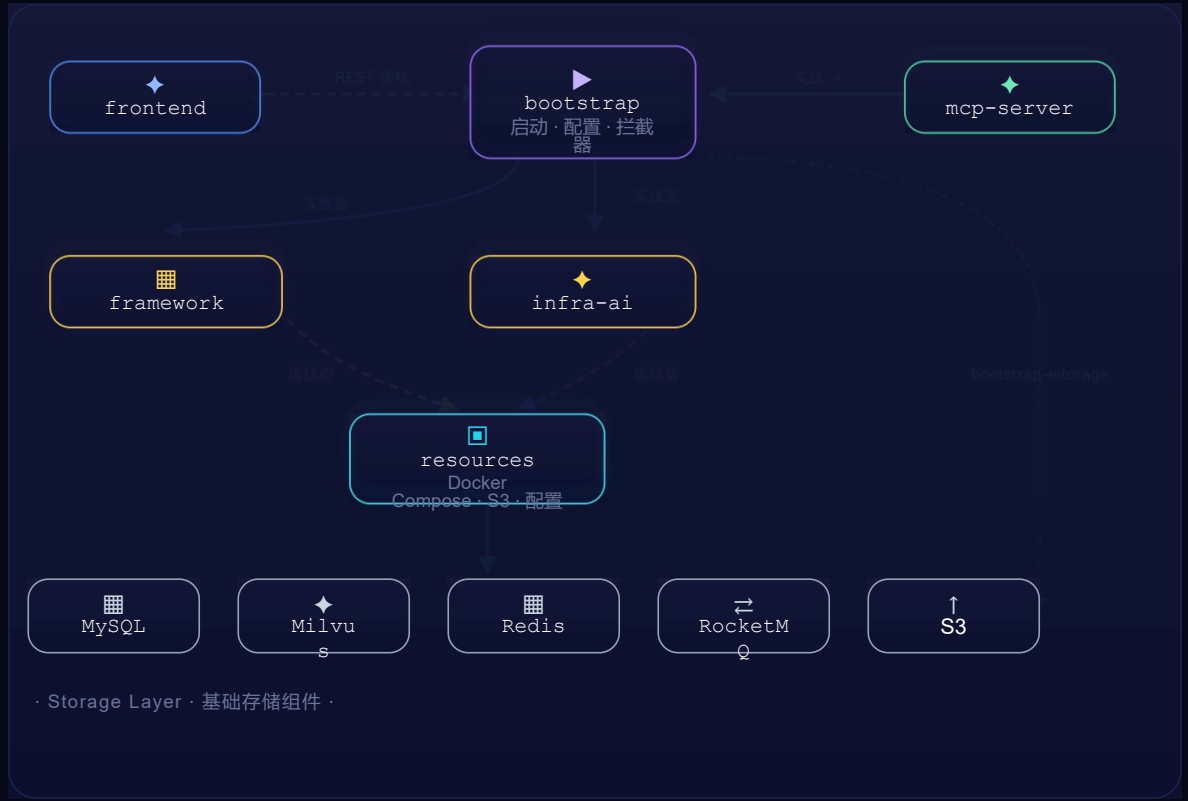
◉维度	✓技术选型	◆选型理由
01后端框架	Java 17 · Spring Boot 3.5.7 · MyBatis Plus	企业级稳定性 + 生态完整 + AI SDK 友好
02前端框架	React 18 · Vite · TypeScript	组件生态丰富 + 类型安全 + 构建速度快
03关系数据库	MySQL（20+ 业务表）	成熟稳定 + 团队熟悉度高
04向量数据库	Milvus 2.6 / pgvector	Milvus 主索引 + pgvector 业务轻量场景
05缓存 / 限流	Redis + Redisson	分布式锁 + 限流 + 会话缓存三合一
06对象存储	S3 兼容存储（RuoYiFS）	标准协议 + 私有化部署友好
07消息队列	RocketMQ 5.x	高吞吐 + 顺序消息 + 事务消息支持
08文档解析	Apache Tika 3.2	支持 1000+ 文件格式 + 元数据提取
09模型供应商	百炼（阿里云）· SiliconFlow · Ollama · vLLM（扩展）	多供应商容灾 + 本地 + 云混合
10认证鉴权	Sa-Token	轻量 + 注解式鉴权 + 多端 SSO
11代码规范	Spotless（自动格式化）	统一代码风格 + Git Hook 强制

✓ 全栈开源　云原生　✦ 私有化友好

* 所有组件均支持私有化部署，无强云依赖

模块依赖与调用关系 · Module Dependencies

实线 · 同步　虚线 · 异步



同步调用链

`frontend → bootstrap → framework → resources`

典型耗时：< 50ms（除 AI 调用）

异步调用链

bootstrap → RocketMQ → 异步任务执行器 → infra-ai → resources

典型场景：文档解析、长对话压缩、向量索引重建

AI 调用链

bootstrap → infra-ai → LLM / Embedding / Rerank → 外部模型 API

典型耗时：500ms - 5s（流式输出）

接口抽象 · 便于 Mock

所有跨层调用均通过接口抽象，便于单元测试与 Mock。

— CHAPTER

04

STREAMCHAT · PIPELINE

CORE ENGINE | 8 阶段 · 3 短路 · 多通道并行

StreamChatPipeline 核心引擎

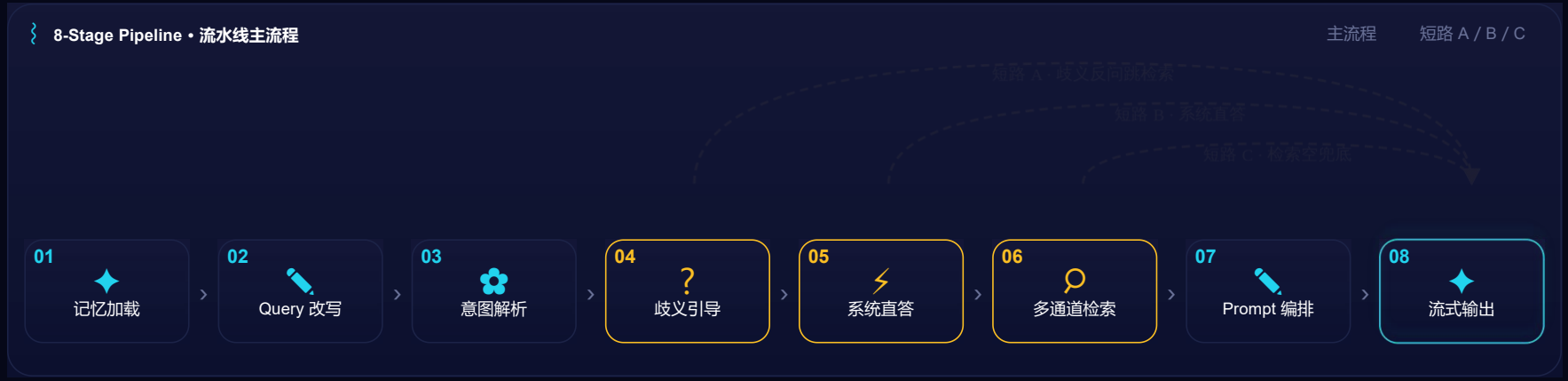
Core Engine · 8 阶段流水线 + 3 条短路 + 多通道并行

「这是整个 Agentic RAG 系统的心脏」

08	03	02
流水线阶段 STAGES	短路机制 SHORTCUTS	检索通道 KB + MCP

StreamChatPipeline 总览 • 8 阶段 + 3 短路

PIPELINE
8 + 3



接入层 • Access Layer — 请求进入 Pipeline 前的 5 道关卡

5 STAGES BEFORE PIPELINE

关卡 ① 多端接入
Web / App / SDK，统一入口

→

关卡 ② 长连接通道
WebSocket / SSE 双协议

→

关卡 ③ 全局流控与排队
ChatQueueLimiter，防雪崩

→

关卡 ④ 会话级互斥锁
Session Lock，防并发冲突

→

关卡 ⑤ 鉴权与路由
AuthN/AuthZ · Route，Sa-Token 实现

→ 进入 StreamChatPipeline

接入层保证请求有序、合法、可控地进入核心流水线

Stage 1-3 · 会话记忆 + Query 改写 + 意图解析

01 会话记忆加载

并行

3 个子任务

1. 拉取历史消息与摘要
2. 摘要注入上下文
3. 超长对话异步压缩

与 Stage 2、3 并行执行；压缩任务通过 RocketMQ 异步触发；摘要由 LLM 生成并缓存到 Redis。

历史拉取延迟
< 50ms（Redis 命中）

02 查询改写与拆分

并行

3 个子任务

1. LLM 改写问题（指代消解、口语化转书面化）
2. 复合问题自动拆分（如「A 和 B 的区别」→ 拆为 2 个子问题）
3. 失败时规则降级（LLM 超时 / 异常时使用正则规则）

本阶段对应高级 RAG 的 Query Translation（Multi-query / Step-back）；规则降级保证可用性。

改写延迟
< 500ms

03 意图解析（并行）

并行

3 个子任务

1. 子问题并行分类
2. 命中 [redacted] / [redacted] / [redacted] 三大类
3. 过滤与 TopN 截断

意图树配置中心动态可调；分类使用 LLM + 规则双重判定；KB = 知识库检索、MCP = 工具调用、SYSTEM = 系统直答。

分类准确率
> 90%

Stage 1-3 全部并行执行，总耗时取最慢者，约 **600ms - 1s**

Stage 4-5 · 歧义引导 + 系统直答（短路 A / B）

04 歧义引导

短路 A ?

3 个子任务

1. 候选意图筛选 — 多个意图分数接近时触发
2. 边界分数二次确认 — LLM 二次判定 + 阈值校验
3. 命中系统关键词则跳过 — 如「你好」「帮助」等直接进入 Stage 5

触发条件
意图分数差 < 0.15 或 Top1 置信度 < 0.7

输出
① 反问用户澄清 ② 直接进入 Stage 5

⚡ 短路 A

Stage 4 命中反问场景 → ~~Stage 5~~ ~~Stage 6~~ ~~Stage 7~~ → Stage 8

直接 Stage 8 流式输出反问，跳过 Stage 5-7

05 系统直答

短路 B ⚡

3 个子任务

1. 全为 SYSTEM 意图时触发 — 如打招呼、问能力、闲聊
2. 跳过检索 — 不进入 Stage 6
3. 直接流式输出 — Prompt 使用 SYSTEM 模板，无证据注入

触发条件
所有子问题意图均为 `SYSTEM`

输出
直接进入 Stage 8，使用 SYSTEM Prompt 模板

⚡ 短路 B

Stage 5 触发 → ~~Stage 6~~ ~~Stage 7~~ → Stage 8

直接 Stage 8 流式输出，跳过 Stage 6-7

⚡ 短路 A/B 是性能优化关键，避免无意义的检索与编排开销；典型命中场景占总流量 **15-25%**

Stage 6 · 多通道并行检索（KB + MCP）+ 短路 C

PARALLEL CHANNELS
2 //

KNOWLEDGE BASE RETRIEVAL
KB 检索

并行

3 个子任务

1. **向量 / 关键词 / 混合并行检索**
   三路并行，Milvus 向量 + ES 关键词 + 混合排序
2. **去重 / Rerank / 截断**
   BGE-Reranker 重排 + Top-K 截断（K = 5）
3. **按意图分组输出**
   不同子问题的检索结果分别打包

关键组件 Milvus 2.6 · BGE-Reranker · bge-m3 Embedding

MCP TOOL INVOCATION
MCP 工具调用

并行

3 个子任务

1. **参数抽取**
   LLM 从用户问题抽取工具调用参数（JSON Schema 约束）
2. **并行调用 MCP Server**
   多工具并发，通过 MCP SDK 1.1
3. **结构化结果回收**
   工具返回 JSON，统一封装为 evidence

典型工具 天气查询 · 销售数据查询 · 工单查询（已实现）；可扩展任意 MCP Server

短路 C

SHORTCUT C

触发条件	KB 检索为空 且 MCP 无可用工具调用
动作	跳过 Stage 7 → Stage 8 输出兜底回复（「未找到相关信息，建议...」）
占比	约 5-8% 流量

检索性能指标

P95

KB 向量检索	P95 < 80ms
Rerank	P95 < 200ms
MCP 工具调用	P95 < 1.5s
总 Stage 6	P95 < 2s

证据组装

EVIDENCE

- KB 证据 + MCP 证据合并去重
- 按子问题分组传给 Stage 7
- 每条证据带 [redacted] 标签

Stage 7-8 · Prompt 编排 + 流式输出

07 Prompt 编排

编排

3 个子任务

1. 按场景路由模板 — KB 模式 / MCP 模式 / 混合模式 / SYSTEM 模式，4 套模板
2. 支持 KB / MCP / 混合模式 — 模板内证据区动态拼装
3. 证据组装后生成提示词 — 结构化注入

✦ PROMPT STRUCTURE · 模板示意

```
[SYSTEM]    你是 XX 助手，回答基于证据...
[HISTORY]   {Stage1 历史摘要}
[EVIDENCE]  {Stage6 检索结果}
[QUESTION]  {用户问题}
[OUTPUT]    流式生成，使用引用标记 [1] [2]
```

08 流式输出

并行

3 个子任务

1. 动态温度策略 — 知识问答 temp = 0.1；创意写作 temp = 0.7；分场景调参
2. Token 级 SSE 推送 — 通过 SSE 协议实时推送，前端打字机效果
3. 结果持久化与摘要更新 — 异步写入 DB + 触发 Stage 1 摘要更新

⌇ PERFORMANCE · 性能指标

首 Token 延迟 P95	< 800ms
完整响应 P95	< 3s
并发上限	单实例 200 SSE 连接

SSE STREAM · 流式输出示意

```
token → token → token → token ▉
```

✦ Stage 7-8 是用户体验的关键，流式输出 + 动态温度 + 异步持久化构成完整的对话体验闭环

— CHAPTER

05

GRAPHRAG · DEEPSEARCH · FUSION

FUSION ARCHITECTURE 三方协同

GraphRAG + DeepSearch 融合

Fusion Architecture · 知识图谱 × 深度搜索 × Agent 协调

「让结构化知识与多步迭代推理统一」

PILLAR 01	PILLAR 02	PILLAR 03
GraphRAG	**DeepSearch**	**Agent 协调**
结构化知识图谱	多步迭代深度搜索	智能体调度编排

GraphRAG · 知识图谱构建与双级检索

Green Box · 结构化知识的精确召回

四大核心模块

知识图谱构建 M 01
- 实体关系提取（LLM驱动） — 自动从文档抽取实体+关系三元组
- 增量更新（文件变更检测） — Hash 比对 + 局部重建，避免全量

索引与社区构建 M 02
- 实体索引（向量相似度合并） — 同名实体自动融合
- 社区检测（Leiden/SLPA 算法） — 聚类形成主题社区

图谱检索 M 03
- 本地搜索（社区内精确） — 实体+邻居+关系，召回精确
- 全局搜索（Map-Reduce 模式） — 跨社区汇总，覆盖广

双级检索（LightRAG） M 04
- 低级：实体细节检索 — 回答「X 是什么」类问题
- 高级：主题概念检索 — 回答「X 领域有哪些方向」类问题

关键实现要点

技术栈 TECH STACK
- 图数据库：Neo4j 5.x
- Embedding：bge-m3
- LLM：GLM-4 / 通义千问
- 算法库：networkx + python-louvain

适用场景 USE CASES
- 多实体关系的复杂问答
- 因果链推理
- 跨文档主题归纳
- 「谁 / 什么 / 如何」类问题

性能指标 METRICS

30min	图谱构建 / 10K 文档	<5s	增量更新 / 单文档
<500ms	本地检索 P95	<2s	全局检索 P95

✓ GraphRAG 解决传统向量检索无法处理关系推理的痛点

DeepSearch · 思考引擎与证据链

Orange Box · 多步迭代推理与信息溯源

四大核心模块

思考引擎 M 01

- Chain-of-Thought（多轮推理） — 显式思维链，每步可观测
- 分支推理（多路径思考） — 并行探索多条推理路径，择优合并

查询生成与执行 M 02

- 查询分解（从复杂到简单） — 复杂问题拆为可检索的原子问题
- 后续查询生成（迭代优化） — 基于中间结果动态生成下一轮查询

证据收集与验证 M 03

- 证据链追踪（信息溯源） — 每条结论标注来源，支持审计
- 答案验证（多角度检查） — 交叉验证 + 事实核查

反馈循环 M 04

- 知识空白识别（信息缺失检测） — 检测证据不足时主动追问
- 整合与合成（生成最终报告） — 多源证据合成结构化报告

DEEPSEARCH 工作流

01 **接收复杂问题**
用户提交需多步推理的复杂分析类问题

02 **Chain-of-Thought 分解**
多路径并行，每条路径独立推演

03 **子查询生成 + 并行检索**
同时检索

04 **证据收集 + 证据链构建**
每条证据标注来源，可审计

05 **答案验证（多角度交叉）**
事实核查 + 自一致性检查

06 **知识空白?**
是 → 返回 Step 3 迭代 | 否 → 进入 Step 7

07 **整合合成最终报告（含证据引用）**
输出结构化结论 + 可点击溯源链

典型迭代次数：2-5 轮 · 单次完整 DeepSearch 耗时 5-15s

DeepSearch 让系统具备『像专家一样思考』的能力，适合复杂分析类问题

Agent 协调系统 · Fusion GraphRAG

Purple Box · 知识与推理的统一融合

GraphRAG

GRE EN

- 图谱构建
- 索引社区
- 图谱检索
- 双级检索

Agent 协调系统

PUR PLE

- 检索计划生成器 — 任务分析与优先级排序
- 多路径执行 — 并行执行多种搜索策略
- 社区感知 — 知识聚集结构识别
- Chain of Exploration — 知识图谱自主探索

DeepSearch

ORA NGE

- 思考引擎
- 查询执行
- 证据验证
- 反馈循环

Fusion GraphRAG

FUSI ON

集成了图谱结构化知识与深度搜索多步迭代推理，实现知识与推理的统一

基础设施层 · Neo4j · LLM · Embeddings

GraphRAG 优势

GRE EN

- 结构化关系推理
- 跨文档主题归纳
- 社区级精准召回
- 适合「关系型」问题

DeepSearch 优势

ORA NGE

- 多步迭代推理
- 证据链可溯源
- 知识空白检测
- 适合「分析型」问题

Fusion 融合优势

FUS ION

- Agent 自主决策调用何种策略
- 知识图谱自主探索（Chain of Exploration）
- 多路径并行 + 择优合并
- 适合「复杂综合型」问题

Fusion GraphRAG 是本方案的差异化核心 — 让 Agent 像专家一样：先查知识，再深度推理，最终合成可信答案

06

SIX

PLATFORM SERVICES & CAPABILITIES

平台服务与关键能力

Platform Services · 模型路由 · ETL · Trace · 限流 · 记忆

支撑 Pipeline 高可用运行的基础设施

28	29	—
模型路由 + ETL	Trace + 限流 + 记忆	本章 2 页

模型路由与容错 + 文档 ETL Pipeline

✦ 模型路由与容错 ROUTE

优先级调度 + 熔断降级

▦ 层1 · 模型供应商层 SUPPLIERS

通义千问	SiliconFlow	Ollama	vLLM
P1 · BaiLian	P2 · GLM-4	P3 · 兜底	后续扩展

▼

⚙ 层2 · 路由策略 ROUTING

- **优先级路由** — 按优先级顺序尝试
- **自动降级** — 失败率 >10% 熔断
- **健康检查** — 每 30s 心跳探测
- **自动恢复** — 5min 后半流量试探

▼

♥ 层3 · 监控指标 MONITORING

- 各模型成功率 / 平均延迟 / 成本
- 实时熔断状态可视化
- 告警阈值可配置

✦ 文档 ETL Pipeline ETL

Apache Tika 3.2 + 异步任务执行器

⌇ 4 阶段流水线 PIPELINE

01	02	03	04
上传	解析	分块	向量化+索引
S3 对象存储 1000+ 格式	Apache Tika 文本+元数据+表格	语义分块 512 token · 重叠 50	bge-m3 + Milvus 写入向量库

↻ **增量更新**
文件 Hash 比对，避免重复处理

▶ **失败重试**
RocketMQ 死信队列 + 人工干预入口

◉ **进度可观测**
全链路 Trace + 实时进度条

模型路由保证 AI 调用高可用；ETL Pipeline 保证 知识库持续更新

全链路 Trace + 并发限流 + 会话记忆管理

全链路 Trace 追踪
FULL-LINK OBSERVABILITY

TRACE

关键能力

- TraceID 贯穿 8 阶段 — 每个 Stage 自动埋点
- 关键节点耗时统计 — Stage 级 P50/P95/P99
- 异常链路快速定位 — 失败节点高亮+快照
- 集成 SkyWalking — 标准化协议，无需自研

典型 TRACE 数据结构

```
trace_id: abc123
├─ stage1_memory:   45ms ✓
├─ stage2_rewrite: 320ms ✓
├─ stage3_intent:  280ms ✓
├─ stage6_retrieval:1.2s ✓
├─ stage7_prompt:   50ms ✓
└─ stage8_generate: 2.8s ✓
```

并发控制与限流
RATE LIMIT & FALLBACK

LIMIT

3 层限流策略

- 全局限流 — ChatQueueLimiter，令牌桶，1000 QPS
- 用户级限流 — 单用户 10 QPS，防滥用
- 会话级互斥锁 — Session Lock，单会话串行

3 个降级策略

- 队列排队 — 超过阈值进入等待队列
- 快速拒绝 — 队列满时返回 429 + 提示
- 熔断降级 — AI 异常时返回兜底回复

关键指标

200	∞
单实例 SSE 并发	集群横向扩展

会话记忆管理
MEMORY COMPRESSION

MEMORY

3 个核心机制

- 历史消息存储 — Redis 缓存 20 轮 + MySQL 持久化
- 摘要压缩 — 超 20 轮自动触发 LLM 摘要
- 上下文窗口 — 总 token 控制在 4K 以内

记忆压缩示意

[第1-15轮] → LLM 摘要 → "用户咨询了XX，结论是YY" [第16-20轮] → 原文保留 [第21轮] → 当前问题

关键指标

20	10:1	<2s
摘要阈值(轮)	压缩比	异步延迟

三大平台服务共同保障系统可观测 · 可控 · 可记忆

07

SEVEN

ROADMAP & DELIVERY

落地与排期

Roadmap & Delivery · MVP → Beta → GA 三阶段

从概念到生产的可执行路径

三阶段里程碑 · MVP → Beta → GA

总计 6 个月 · 3 个阶段 · 12 个关键交付物

01 MONTH 1-2

02 MONTH 3-4

03 MONTH 5-6

PHASE 01 MVP

最小可用版本

第 1-2 月

目标：核心 Pipeline 跑通，内部 demo 可用

关键交付（4项）

- StreamChatPipeline 8 阶段打通（无短路）
- 单一知识库 + 基础检索（向量+Rerank）
- React Chat 界面 + SSE 流式
- 单模型接入（GLM-4）

验收标准

基础问答可用 · P95 < 5s

4 人

1 前端 + 2 后端 + 1 算法

PHASE 02 BETA

内测版本

第 3-4 月

目标：完整功能 + 多知识库 + 平台服务

关键交付（4项）

- 3 条短路路径全部实现（A/B/C）
- MCP 工具协议集成（3个工具）
- 多模型路由 + 容错降级
- 全链路 Trace + 限流 + 记忆压缩

验收标准

内部 10 人日活使用 · P95 < 3s

7 人

+1 前端 +1 运维 +1 测试

PHASE 03 GA ★

正式发布

第 5-6 月

目标：生产可用 + GraphRAG 融合 + 商业化

关键交付（4项）

- Fusion GraphRAG（GraphRAG + DeepSearch + Agentic）
- Admin 管理后台完整功能
- 多租户隔离 + 配额计费
- SLA 99.9% + 监控告警体系

验收标准

100+ 业务知识库 · 10K QPS

10 人

+1 架构师 +1 SRE +1 产品

✓ 每个阶段都有明确的可验收标准，避免无边界延期

团队分工与资源配置 · Team & Resources

团队角色分工

总计：13 人（高峰期）

#	角色	人数	主要职责	关键产出
01	架构师	1	总体架构设计、技术选型、关键技术攻关	架构文档、技术选型清单
02	后端 Lead	1	Pipeline 后端架构、Code Review、技术决策	后端架构图、API 规范
03	后端工程师	3	StreamChatPipeline / 平台服务 / ETL 实现	模块代码、单元测试
04	前端工程师	2	Chat 界面 / Admin 后台 / SSE 对接	前端代码、UI 组件库
05	算法工程师	1	RAG 策略调优 / Prompt 工程 / Rerank	策略文档、评测报告
06	GraphRAG 工程师	1	Neo4j 建模 / 图谱构建 / LightRAG	图谱构建脚本
07	测试工程师	1	功能 / 性能 / 稳定性测试	测试用例、自动化脚本
08	SRE 运维	1	部署 / 监控 / 告警 / 容灾	部署文档、监控大盘

硬件资源 HARDWARE

- GPU 服务器：4× A10（本地推理）
- 应用服务器：8× 16C32G（K8s 集群）
- 存储容量：5TB S3 + 2TB Milvus

估算月成本
约 ¥3-5万 （私有化）

外部依赖 DEPENDENCIES

- LLM API：百炼 + SiliconFlow 双供应商
- Neo4j Enterprise License
- 监控：SkyWalking 开源版
- 日志：ELK 开源版

协作工具 COLLABORATION

代码 Git + GitFlow	项目 Jira / 飞书
文档 Confluence / 飞书	沟通 飞书 + 企微

13 人 × 6 个月 ≈ 78 人月，对应市场价约 ¥400-600 万 投入

风险识别与应对预案 · Risk Management

风险矩阵 · 6 大类风险

高 中 低

#	风险项	类别	概率	影响	应对预案
01	LLM API 不稳定	供应方风险	中	高	多供应商路由 + 本地 Ollama 兜底 + 熔断降级
02	检索召回率不达预期	技术风险	中	高	多路并行检索 + Rerank + HyDE 增强 + 持续评测
03	复杂问题推理超时	性能风险	高	中	短路 C 快速返回 + 异步深度搜索 + 前端进度提示
04	知识库数据质量差	数据风险	高	高	ETL 前数据清洗 + 人工抽检 + 反馈闭环
05	多租户隔离不严	安全风险	低	高	Schema 级隔离 + API 鉴权 + 审计日志

技术预案
TECHNICAL

- 每个关键模块都有降级方案
- 短路机制保证基础可用
- 灰度发布 + AB 测试
- 性能压测每周一次

流程预案
PROCESS

- 双周迭代 + 可验收里程碑
- Code Review 强制 2 人通过
- 上线 Checklist 制度
- 故障复盘 24h 内完成

组织预案
ORGANIZATIONAL

- 关键岗位 AB 角 配置
- 知识库 Wiki 持续沉淀
- 外部专家顾问池
- 应急响应 On-Call 轮值

✓ 风险预案不是文档，而是日常执行的 Checklist

核心优势与创新点 · Key Strengths

ADV.01 AGENTIC

Agentic 决策

Agent 驱动的动态决策

StreamChatPipeline 8 阶段 + 3 条短路，由意图解析动态路由；非固定 Pipeline，按需触发检索 / 工具 / 直答。

↘ 典型场景节省 **30-50%** 延迟

ADV.02 FUSION

Fusion GraphRAG

知识与推理融合

业界少有的 GraphRAG + DeepSearch + Agent 协调 三合一融合架构；既精确召回结构化知识，又支持多步迭代推理。

↗ 复杂问题准确率提升 **25%+**

ADV.03 RESILIENCE

生产级容灾

多模型 + 多通道容灾

LLM 多供应商路由 + 熔断降级；KB + MCP 双通道 并行；3 条短路保证基础可用；全链路 Trace 可观测。

✓ SLA **99.9%** · 可用性 **>99.95%**

ADV.04 EXTENSIBLE

可扩展生态

MCP 协议 + 模块化分层

基于 MCP SDK 1.1，工具即插即用；四层模块分层清晰；新增知识库 / 模型 / 工具均无需改架构。

⚡ 新工具接入 **< 1 人天**

vs Naive RAG

本方案	意图识别 + 多通道 + Rerank + 短路
NAIVE	单次检索 + 直接生成

↑ 准确率 **+30%** · ↓ 延迟 **-40%**

vs LangChain Agent

本方案	生产级容灾 + 全链路 Trace + 多租户
LC	研究框架，缺生产化能力

✓ SLA **99.9%** vs **不可用**

vs 商业 RAG 产品

本方案	开源 + 私有化 + 可定制
商业	黑盒 + 按调用量计费

TCO 降低 **60%+**

本方案的核心价值：把研究级的 Agentic RAG 能力，工程化为生产级可交付系统

END OF PRESENTATION

Thank You

Q & A · 欢迎提问与讨论

让 Agent 像专家一样：先查知识，再深度推理，最终合成可信答案

项目 WIKI
wiki.internal/agent-rag

代码仓库
git.internal/agent-rag

技术答疑群
飞书搜「Agentic RAG」