愚戏·教学实训智能体

**产品总体设计文档**

作品名称： 愚戏·教学实训智能体

作　　者：愚戏师

目 录

**[第一章](#_Toc205233077)****[产品概述](#_Toc205233077)** [3](#_Toc205233077)

[1.1产品开发背景与目的 3](#_Toc205233078)

[1.2产品架构设计 3](#_Toc205233079)

**[第二章](#_Toc205233080)****[产品详细设计](#_Toc205233080)** [8](#_Toc205233080)

[2.1知识库构建与数据库核心实体关系设计 8](#_Toc205233081)

[2.2后端关键算法与技术 14](#_Toc205233082)

[2.3后端系统功能设计 19](#_Toc205233083)

[2.4后端技术难点解决方案 20](#_Toc205233084)

[2.5 模型集成设计 21](#_Toc205233085)

[2.6前端设计 24](#_Toc205233086)

[2.7前端界面设计 25](#_Toc205233087)

**[第三章](#_Toc205233088)****[产品设计严格满足赛题需求说明](#_Toc205233088)** [34](#_Toc205233088)

[3.1 非功能性需求满足说明 34](#_Toc205233089)

[3.2 基础功能满足说明 40](#_Toc205233090)

[3.3技术与创新性汇总说明 63](#_Toc205233091)

[3.4教育实用与创新性说明 68](#_Toc205233092)

**[第四章](#_Toc205233093)****[接口设计](#_Toc205233093)** [70](#_Toc205233093)

[4.1教师模块 70](#_Toc205233094)

[4.2 AI 大模型交互模块 71](#_Toc205233095)

[4.3学生模块 72](#_Toc205233096)

[4.4课件管理模块 73](#_Toc205233097)

[4.5用户管理模块 74](#_Toc205233098)

[4.6仪表盘模块 75](#_Toc205233099)

[4.7代码分析模块 76](#_Toc205233100)

[4.8通用说明 77](#_Toc205233101)

**[第五章](#_Toc205233102)****[产品总结](#_Toc205233102)** [78](#_Toc205233102)

[5.1开发过程中的挑战与突破 78](#_Toc205233103)

[5.2能力提升关键点 78](#_Toc205233104)

[5.3项目升级演进规划 79](#_Toc205233105)

[5.4商业推广价值 79](#_Toc205233106)

# **产品概述**

### 1.1产品开发背景与目的

当前高校实训教学体系在效率与效果层面面临两个结构性挑战：

（1）教师端备课负担重： 高度依赖人工进行课程体系设计、教学资源整合与适配，难以高效应对学生个体差异化学习需求（如知识基础、学习风格），导致教学资源配置效率低下。

（2）学生端实践支持薄弱： 课后缺乏及时、精准的指导机制，学生实操过程中遇到的问题无法获得实时诊断与反馈，制约实践能力的深度构建与学习效果巩固。

针对上述痛点，本项目基于开源大型语言模型（LLM）技术，研发高校实训教学智能体解决方案。核心功能目标包括：智能化资源生成， 动态生成高度契合教学目标与进度的多元化教学资源（如课件、习题、案例）。实时交互式辅导， 构建7x24实时答疑辅导机制，并提供精细化错误分析与学习路径建议。个性化学习定制，依托学生数据模型与AI分析能力，智能生成个性化学习内容与训练路径。自动化流程覆盖，高效处理师生高频交互场景（如基础答疑、进度跟踪、共性知识推送），实现教学管理流程自动化。

本产品的核心价值在于：赋能教师生产力，显著减轻教师在重复性、事务性工作上的负荷，释放其专注力于更高阶的教学设计、科研与个性化指导。提升学生学习效能，提供持续、及时、个性化的实践支持，深化技能掌握，优化学习体验与成果。驱动教学范式转型，促进教育资源的智能化生成、精准化匹配与高效化复用，助力高校实训教学向数智化、个性化方向跃迁。

### 1.2产品架构设计

本系统是基于Spring Boot的教育辅助平台，集成大语言模型实现智能化教学功能，主要提供课件管理、学习记录分析、智能题目生成、自动批改和学情报告等核心功能。系统采用分层架构与微服务化设计，支持学生、教师、管理员三类角色如表 1产品角色领域表所示，通过前后端分离实现高内聚低耦合，并结合关系型数据库和向量数据库实现结构化与非结构化数据管理。整体架构分为如下四层：

（1）前端交互层：基于角色权限的多端界面

（2）API网关层：RESTful接口统一路由

（3）业务服务层：领域驱动的微服务集群

（4）基础设施层：AI引擎与数据持久化

前端交互层、API网关层如 图 1产品前端和接口层架构设计所示，业务服务层、基础设施层如图 2产品服务与持久化架构所示。

#### 1.2.1前端交互层

前端架构采用模块化设计思想，根据用户角色（学生/教师/管理员）划分功能域，实现专业化操作体验：

通过动态路由系统实现按角色权限加载功能模块，学生端聚焦学习路径，教师端侧重教学管理，管理员掌控系统配置。路由加载器在认证阶段识别JWT中的角色标识，动态注入专属路由配置。

采用状态库实施业务数据与UI状态分离策略：

业务数据层：维护课件体系、知识点拓扑等教育核心数据，支持服务端持久化与同步

UI状态层：独立管理导航展开状态、响应式断点等界面交互状态，确保视图与控制解耦

该设计通过角色路由+状态分层+响应式矩阵形成教育系统的模块化内核，既满足多角色场景的功能隔离需求，又保障跨设备体验的专业性和一致性。

表 1产品角色领域表



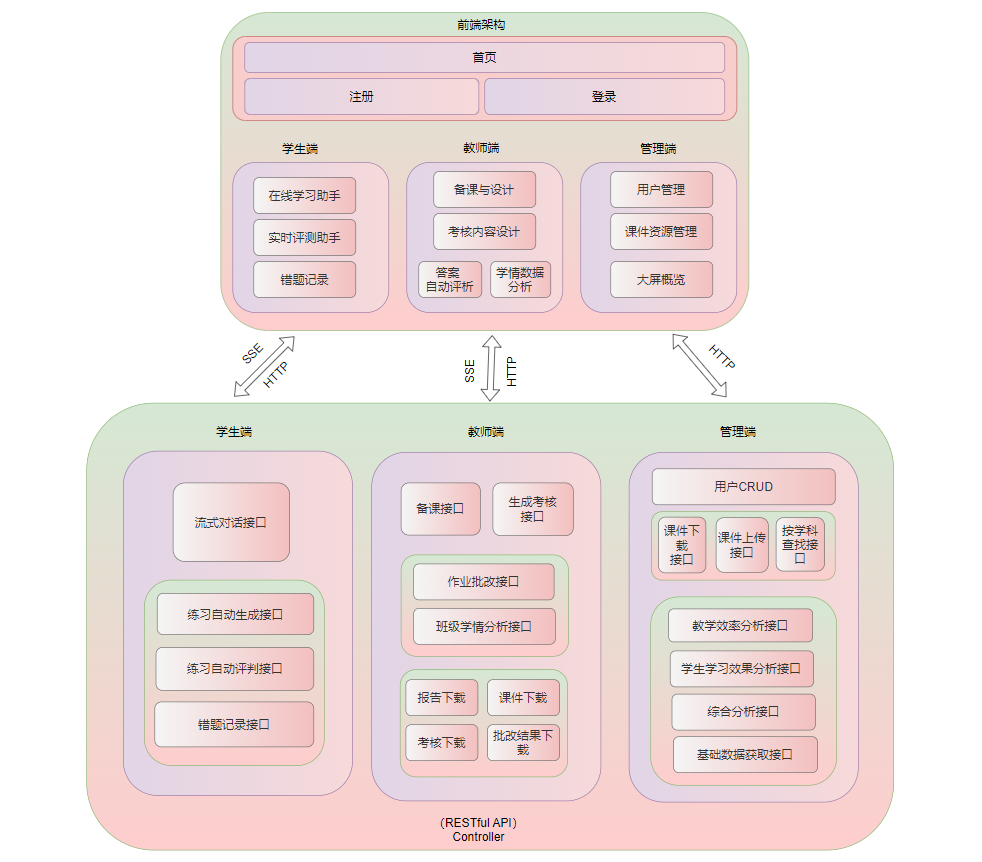
****

图 1产品前端和接口层架构设计

#### 1.2.2API网关层

网关层所有接口均采用 RESTful 设计风格，以实现接口的可读性、可扩展性及一致性。具体遵循以下规则：

资源命名：以名词复数形式定义资源，如 “courses”，精准对应课件、习题等教学资源，避免使用动词，确保资源语义清晰。

HTTP 方法映射：通过标准 HTTP 方法表达对资源的操作，GET 用于查询（如获取课件列表）、POST 用于新增（如提交学习记录）、PUT 用于全量更新（如修改习题内容）、PATCH 用于部分更新（如更新学情数据）、DELETE 用于删除（如删除过期习题）等。

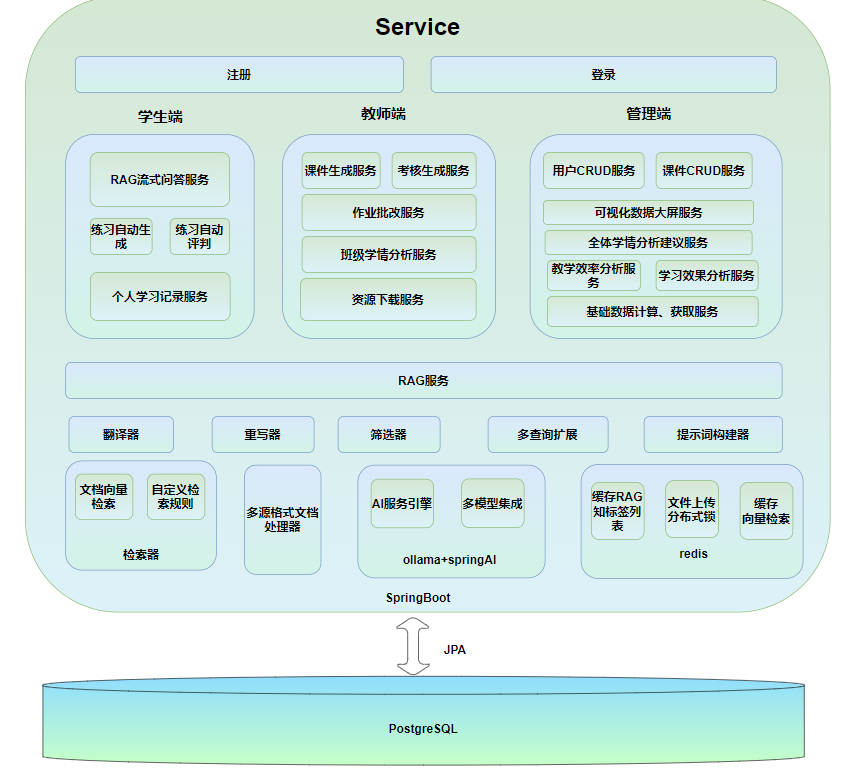
为适配学生、教师、管理员三类角色的功能隔离需求，网关层为不同角色分配独立 API 命名空间，并通过路由规则实现请求的精准转发。除此之外还集成 JWT 令牌鉴权功能。

图 2产品服务与持久化架构

#### 1.2.3业务服务层

基于领域驱动设计（DDD）原则，根据业务能力边界划分核心微服务如表 2微服务设计表所示：

表 2微服务设计表

|  |  |  |
| --- | --- | --- |
| **服务类型** | **核心服务** | **依赖组件** |
| **通用服务** | 注册登录服务 | JWT认证 |
| **学生域** | RAG流动问答服务 | AI引擎/Redis缓存 |
|  | 练习自动生成/评析服务 | 多查询扩展器/ AI引擎 |
|  | 错题记录服务 | PostgreSQL |
| **教师域** | 课件/考核生成服务 | 多源文档处理器/AI引擎 |
|  | 班级学情分析服务 | 数据计算服务/ AI引擎 |
|  | 作业批改服务 | AI评分引擎 |
| **管理域** | 用户CRUD服务 | 权限验证 |
|  | 教学效率/学生学习效果分析服务 | 数据聚合服务/ AI引擎 |
|  | 大屏可视化服务 | Echarts数据管道/ AI引擎 |

#### 1.2.4. 基础设施层

##### 1.2.4.1存储方案设计

表 3存储方案设计表

|  |  |  |
| --- | --- | --- |
| **数据类型** | **存储方案** | **用途** |
| **用户数据** | PostgreSQL | 账号/权限/个人资料 |
| **学习行为数据** | PostgreSQL时序扩展 | 练习记录/错题集 |
| **课件资源** | 对象存储(MinIO) | 课件文件版本管理 |
| **RAG缓存** | Redis | 高频问答结果缓存 |
| **向量数据** | PGVector扩展 | 文档语义检索 |

##### 1.2.4.2 AI引擎集成设计

本系统集成 DeepSeek-R1:7b/32b模型和qwen2.5-coder:3b模型，并根据场景需求配置不同规格的模型实例：

（1）教师/管理端模型

表 4教师/管理端模型设计表



（2）学生端模型

设计意图说明：

轻量级部署：7B 参数版本适合学生端大量并发请求，降低服务器资源消耗；稳定性优先：temperature=0.1 确保答案准确、一致，避免创造性但可能错误的回答；实时响应：针对学生交互场景优化，在保证质量的前提下减少生成延迟。

表 5学生端模型设计表



（3）代码批改模型

表 6代码批改模型设计表



# **产品详细设计**

### 2.1知识库构建与数据库核心实体关系设计

#### 2.1.1本地知识库构建方案



图 3本地知识库构建流程图

##### 2.1.1.1. 核心技术组件 - 开源大模型集成

Ollama嵌入模型（nomic-embed-text）：

// OllamaConfig.java

OllamaEmbeddingModel.builder()

.ollamaApi(ollamaApi)

.defaultOptions(OllamaOptions.builder().model("nomic-embed-text").build())

.build();

亮点：采用轻量级开源嵌入模型，专为文本语义优化，显著提升知识片段表征准确性。

##### 2.1.1.2. 知识库构建流程

步骤1：多源文档解析

// RagServiceTeacherImpl.java

TikaDocumentReader documentReader = new TikaDocumentReader(file.getResource());

List<Document> documents = documentReader.get();

亮点：通过Apache Tika解析PDF/Word/PPT/TXT等格式，突破单一文本限制可支持多格式的教学资源解析

步骤2：动态知识切片

TokenTextSplitter tokenTextSplitter = new TokenTextSplitter(); // 基于语义的分块

List<Document> documentSplitterList = tokenTextSplitter.apply(documents);

亮点：按语义而非固定长度分块，避免知识点割裂；自动处理中英文混合文本（TokenTextSplitter适配多语言）

步骤3：元数据增强存储

documents.forEach(doc -> doc.getMetadata().put("knowledge", ragTag));

pgVectorStore.accept(documentSplitterList);

亮点：为每个知识片段注入ragTag元数据；按学科标签精准检索

步骤4：分布式标签管理

// Redis维护全局唯一标签

RList<String> elements = redissonClient.getList("ragTag");

if (!elements.contains(ragTag)) {

elements.add(ragTag);

}

亮点：基于Redisson的分布式列表实现标签原子化操作；避免集群环境下的标签重复问题。

##### 2.1.1.3. 创新性设计亮点

①双向量库隔离设计

// OllamaConfig.java

@Bean("pgVectorStore") // 本地教学资源知识库

@Bean("pgVectorStoreQuestion") // 学生问题专用库

分离知识点存储与问题库，避免交叉污染

②轻量级知识库（<100M）保证

通过语义分块压缩冗余信息，PgVectorStore的二进制向量存储优化空间占用。

③关联性保障机制

检索阶段：向量相似度+元数据过滤双重校验

生成阶段：限定大模型仅使用检索到的知识片段

##### 2.1.1.4. 针对赛题需求功能性设计总结



##### 2.1.1.5. 效果验证方案（测试阶段验证保证）

###### 2.1.1.5.1关联性测试验证算法设计

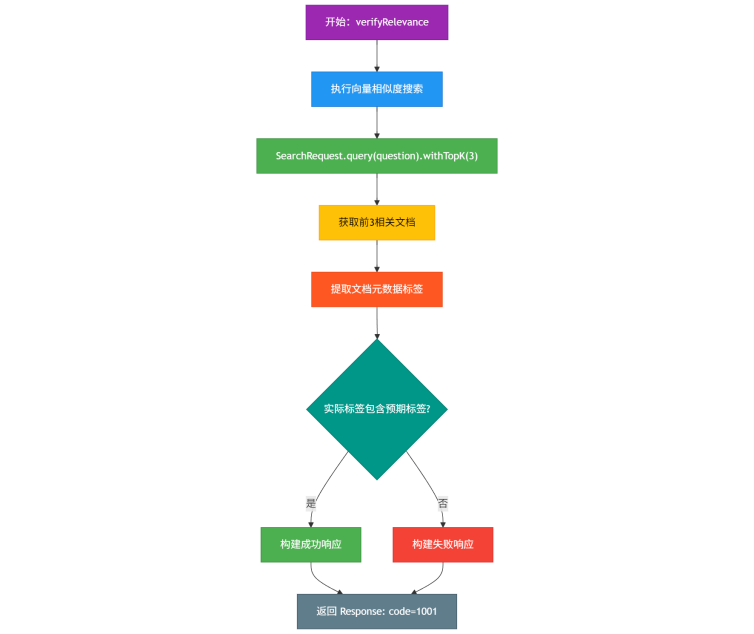


图 4关联性测试验证算法

/\*\*

\* 验证问题与知识库的关联性

\* @param question 测试问题

\* @param expectedTag 预期标签

\* @return 验证结果

\*/

public Response<Boolean> verifyRelevance(String question, String expectedTag) {

// 执行向量相似度搜索

List<Document> results = vectorStore.similaritySearch(

SearchRequest.query(question).withTopK(3)

);

// 提取实际标签

List<String> actualTags = results.stream()

.map(doc -> doc.getMetadata().getOrDefault("knowledge", "").toString())

.collect(Collectors.toList());

// 验证标签匹配

boolean isRelevant = actualTags.contains(expectedTag);

return Response.<Boolean>builder()

.code(isRelevant ? "0000" : "1001")

.info(isRelevant ? "关联性验证通过" : "关联性验证失败")

.data(isRelevant)

.build();

}

###### 2.1.1.5.2准确性测试验证算法设计

代码如下所示，算法流程如图 5准确性测试验证算法流程所示。

/\*\*

\* 验证答案与知识库的准确性

\* @param question 问题

\* @param generatedAnswer 生成的答案

\* @return 验证结果

\*/

public Response<Boolean> verifyAccuracy(String question, String generatedAnswer) {

// 1. 检索相关知识片段

List<Document> relevantDocs = vectorStore.similaritySearch(question);

// 2. 检查生成答案是否包含引用

boolean hasCitations = checkCitations(generatedAnswer, relevantDocs);

// 3. 验证答案准确性

boolean isAccurate = verifyContentAccuracy(generatedAnswer, relevantDocs);

return Response.<Boolean>builder()

.code(hasCitations && isAccurate ? "0000" : "1002")

.info(hasCitations && isAccurate ?

"准确性验证通过" : "准确性验证失败")

.data(hasCitations && isAccurate)

.build();

}

// 检查答案是否包含引用标记

private boolean checkCitations(String answer, List<Document> docs) {

return docs.stream().anyMatch(doc ->

answer.contains("[doc:" + doc.getId() + "]"));

}

// 验证答案内容准确性

private boolean verifyContentAccuracy(String answer, List<Document> docs) {

return docs.stream().anyMatch(doc ->

answer.contains(doc.getContent().substring(0, 20)));

}

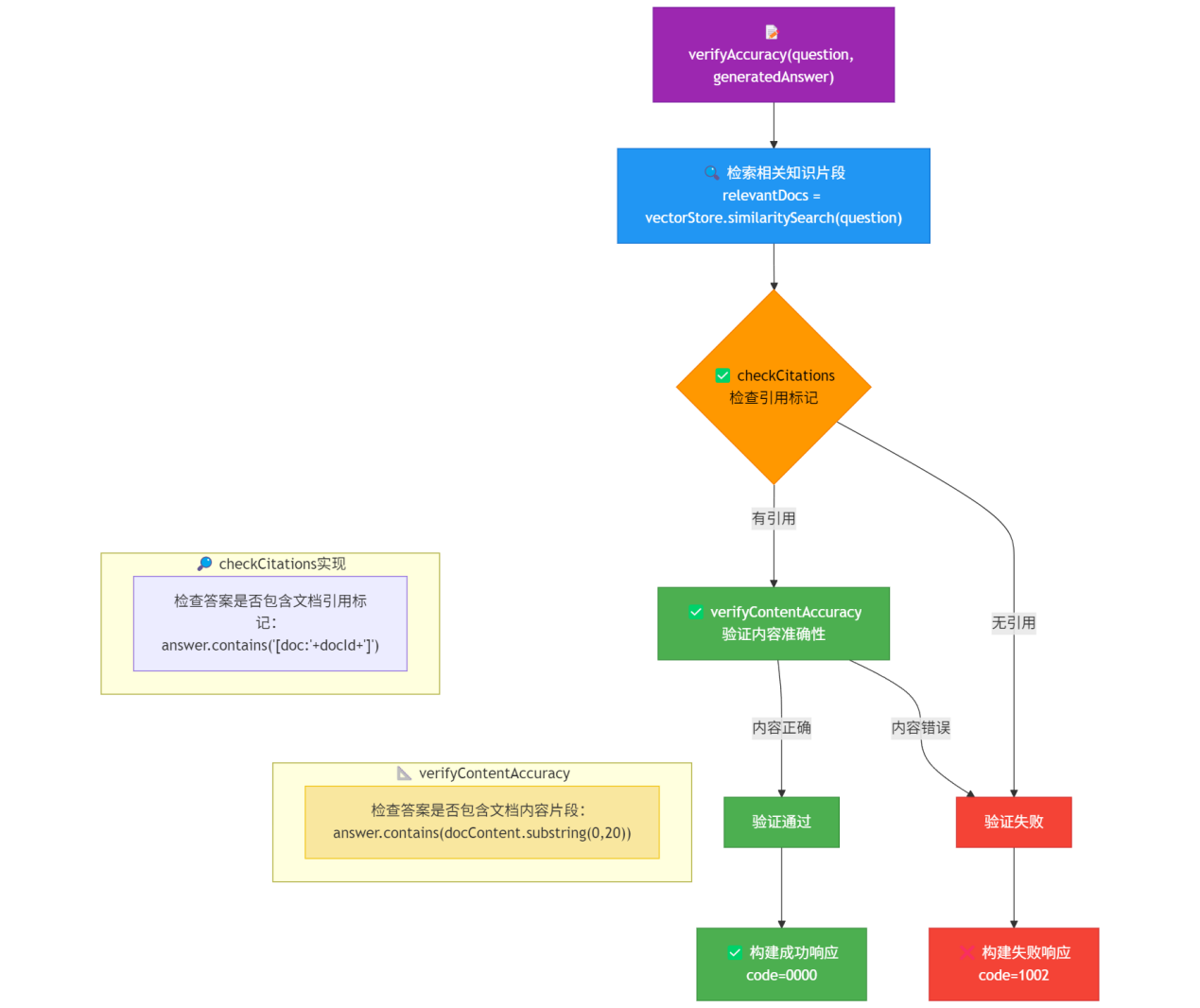


图 5准确性测试验证算法流程

###### 2.1.1.5.3容量控制算法设计（正式版本保留）

代码如下，算法流程如图 6容量控制算法所示。

/\*\*

\* 获取知识库当前大小(MB)

\*/

public double getCurrentSizeMB() {

final String SQL = "SELECT pg\_total\_relation\_size('vector\_store\_ollama\_deepseek') / (1024.0 \* 1024.0) AS size\_mb";

try (Connection conn = dataSource.getConnection();

PreparedStatement stmt = conn.prepareStatement(SQL);

ResultSet rs = stmt.executeQuery()) {

if (rs.next()) {

return rs.getDouble("size\_mb");

}

} catch (Exception e) {

e.printStackTrace();

}

return 0;

}

/\*\*

\* 容量检查并触发警报

\*/

@Scheduled(fixedRate = 60000) // 每分钟检查一次

public void checkCapacity() {

double currentSize = getCurrentSizeMB();

if (currentSize > maxSizeMB) {

triggerAlert("CRITICAL: 知识库大小超过 " + maxSizeMB + "MB! 当前大小: " + currentSize + "MB");

} else if (currentSize > warningThresholdMB) {

triggerAlert("WARNING: 知识库大小接近上限! 当前大小: " + currentSize + "MB");

}

}

// 触发警报（记录日志+Redis通知）

private void triggerAlert(String message) {

// 1. 记录系统日志

System.err.println("[知识库告警] " + message);

// 2. 发送到Redis通知频道

redissonClient.getTopic("knowledgebase-alerts").publish(message);

}

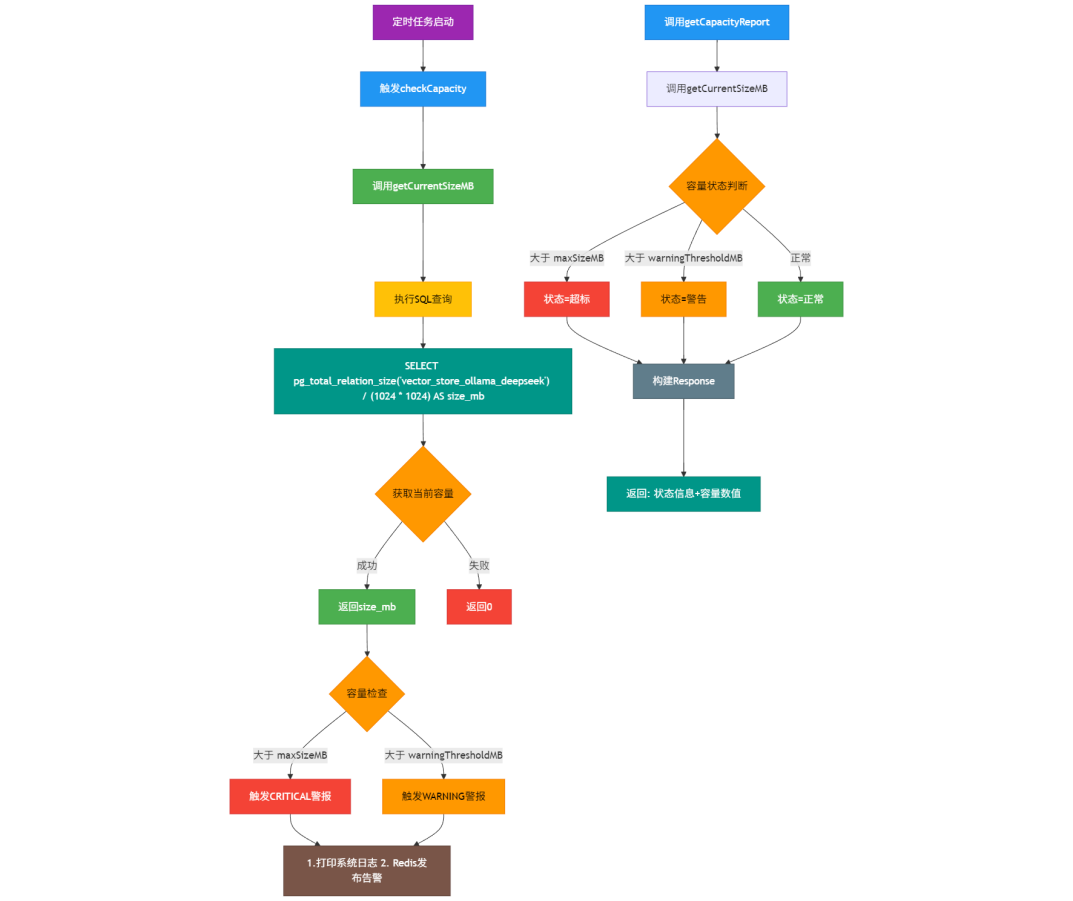
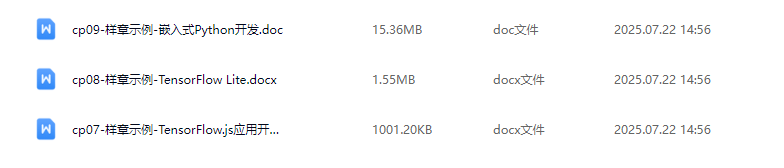


图 6容量控制算法

##### 2.1.1.6数据文件使用说明

严格采用赛事提供的样例文件构建知识库



##### 2.1.1.7总结

通过动态语义分块+元数据增强+双库隔离设计，实现轻量级（<100M）知识库的高效构建，结合Ollama开源模型与分布式存储，在保障知识点关联性的同时满足教育场景精准检索需求。

#### 2.1.2数据库核心实体关系设计

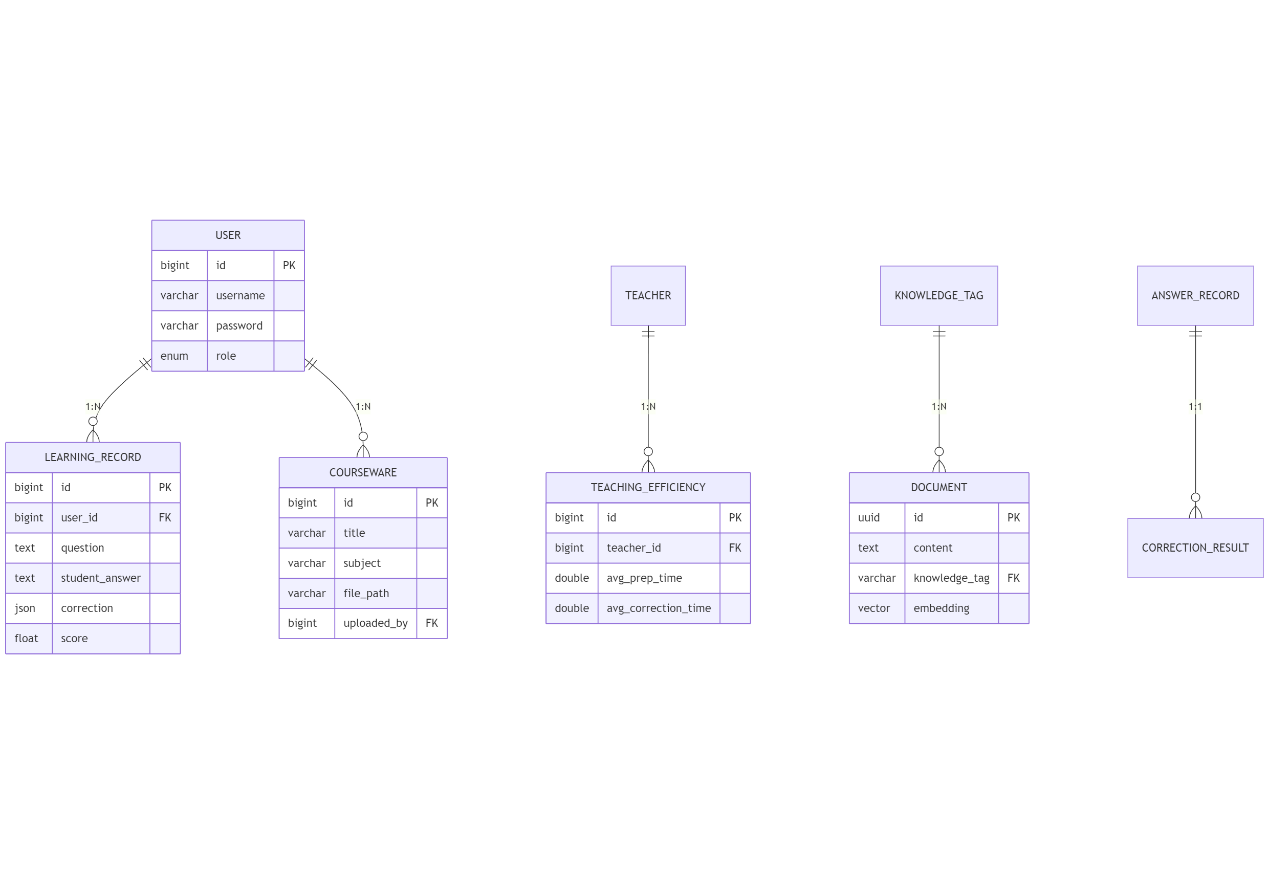


图 7核心实体关系图

### 2.2后端关键算法与技术

#### 2.2.1创新技术亮点概览

表 7创新技术亮点表



#### 2.2.2.多段查询优化算法（语义增强检索）

// AiServiceImpl.java

public Flux<ChatResponse> generateStreamRag(String ragTag, String message) {

// 1. 查询翻译

Query translationQuery = translationQuery(message);

// 2. 查询重写

Query rewrittenQuery = rewriteQuery(translationQuery.text());

// 3. 多查询扩展

List<Query> expandedQueries = multiRewriteQuery(rewrittenQuery.text());

// 4. 向量检索

List<Document> documents = retriever.retrieve(expandedQueries);

// 5. RAG增强生成

return chatModel.stream(new Prompt(buildRagMessages(documents)));

}

代码算法流程如图 8多段查询优化算法流程图所示。

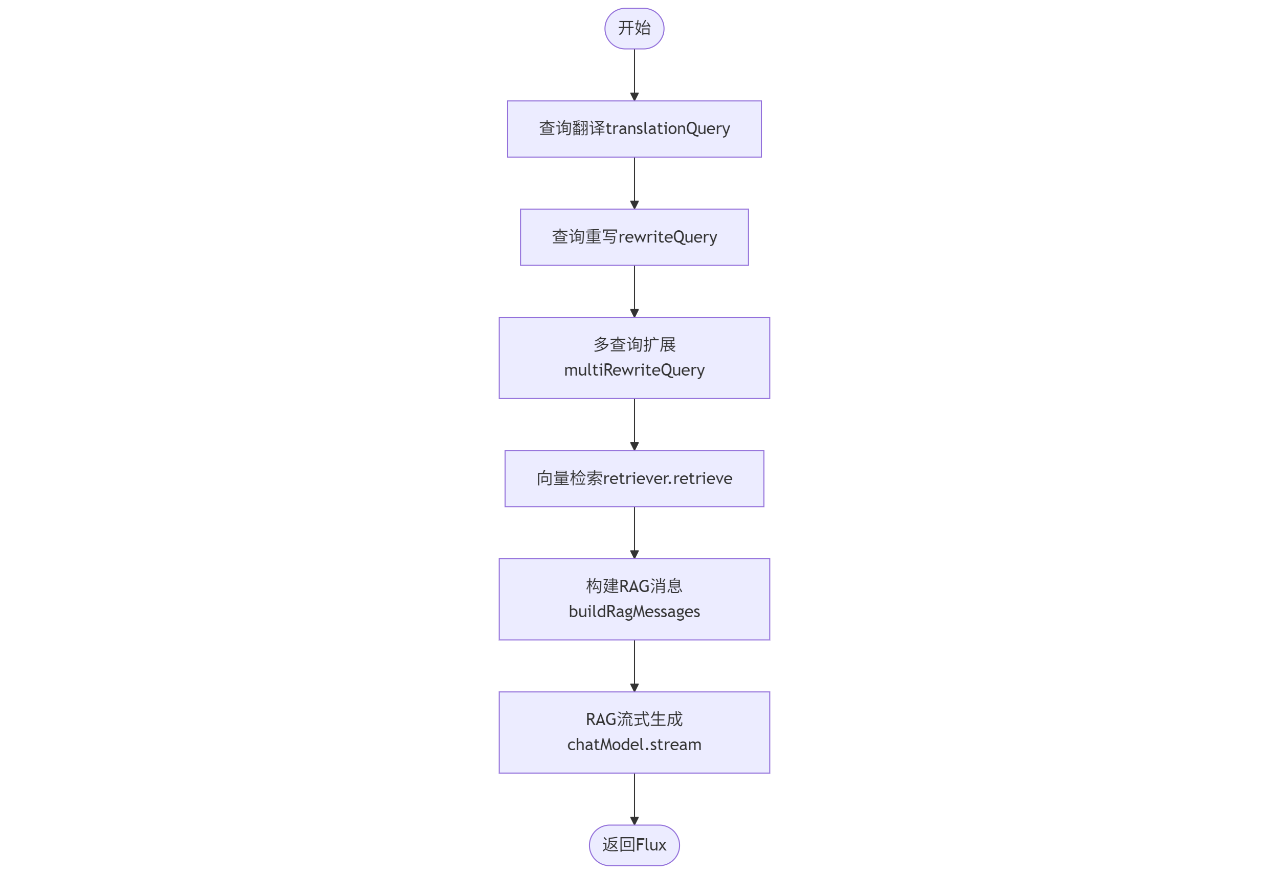


图 8多段查询优化算法流程图

技术创新点说明：查询翻译重写提升非母语问题理解；多查询扩展解决术语差异问题；动态相似度阈值过滤低质量文档

#### 2.2.3自适应题目生成算法（多模态RAG+动态难度调整）

// AiServiceStudentimpl.java

public Flux<ChatResponse> generateRagQuestion(String ragTag, String message) {

// 1. 解析知识点和题目要求

String knowledgePoint = extractKnowledgePoint(message);

// 2. 检索相关文档

List<Document> documents = retrieveDocuments(knowledgePoint);

// 3. 获取历史错题记录

List<LearningRecord> records = getErrorRecords(knowledgePoint);

// 4. 动态调整题目难度

String difficulty = calculateDifficulty(records);

// 5. 生成个性化题目

return ollamaChatModel.stream(buildQuestionPrompt(documents, records, difficulty));

}

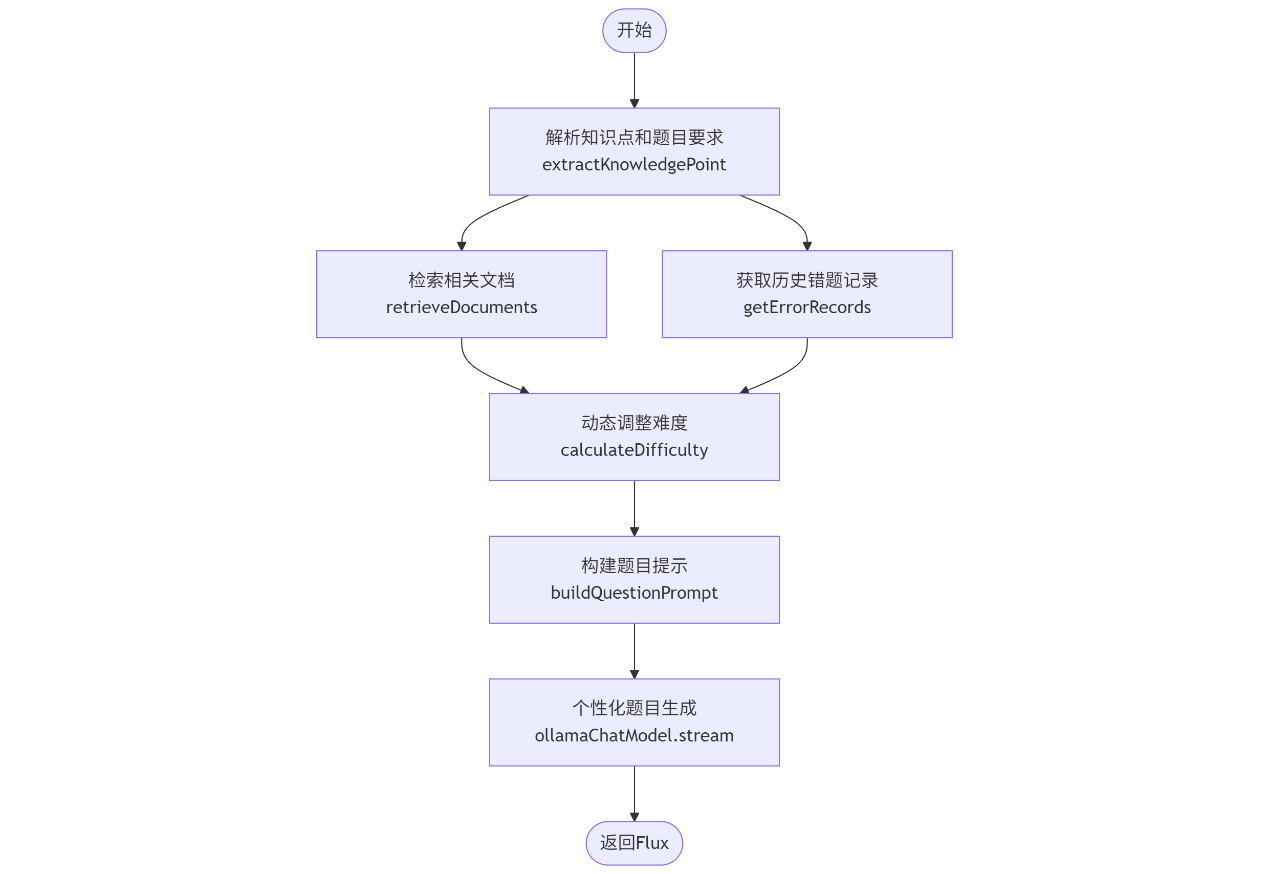


图 9自适应题目生成算法流程表

算法流程如图 9自适应题目生成算法流程表所示。

知识点解析：extractKnowledgePoint(message)从用户消息中提取核心知识点关键词。比如输入："请出一道关于三角函数计算的题" 则输出："三角函数"。

双路并行处理：文档检索，retrieveDocuments(knowledgePoint)获取知识点相关教学资料；历史分析，getErrorRecords(knowledgePoint)查询该知识点的历史错题记录。

动态难度调整：calculateDifficulty(records)

基于错题记录分析：高频错误 → 降低难度；正确率高 → 提升难度；无历史记录 → 默认中等难度；即正确率<50% → 基础题，50%-80% → 进阶题，80% → 综合应用题，优先选择错误率最高的知识点如图 10动态难度调整所示。

个性化题目生成：buildQuestionPrompt() 整合三要素：new PromptContext(documents, records, difficulty)ollamaChatModel.stream() 流式生成个性化题目响应

表 8本系统与传统RAG创新对比表



创新点：在教育场景中创新，即通过错题记录（LearningRecord）实现"靶向教学"，使题目难度随学生能力动态进化。

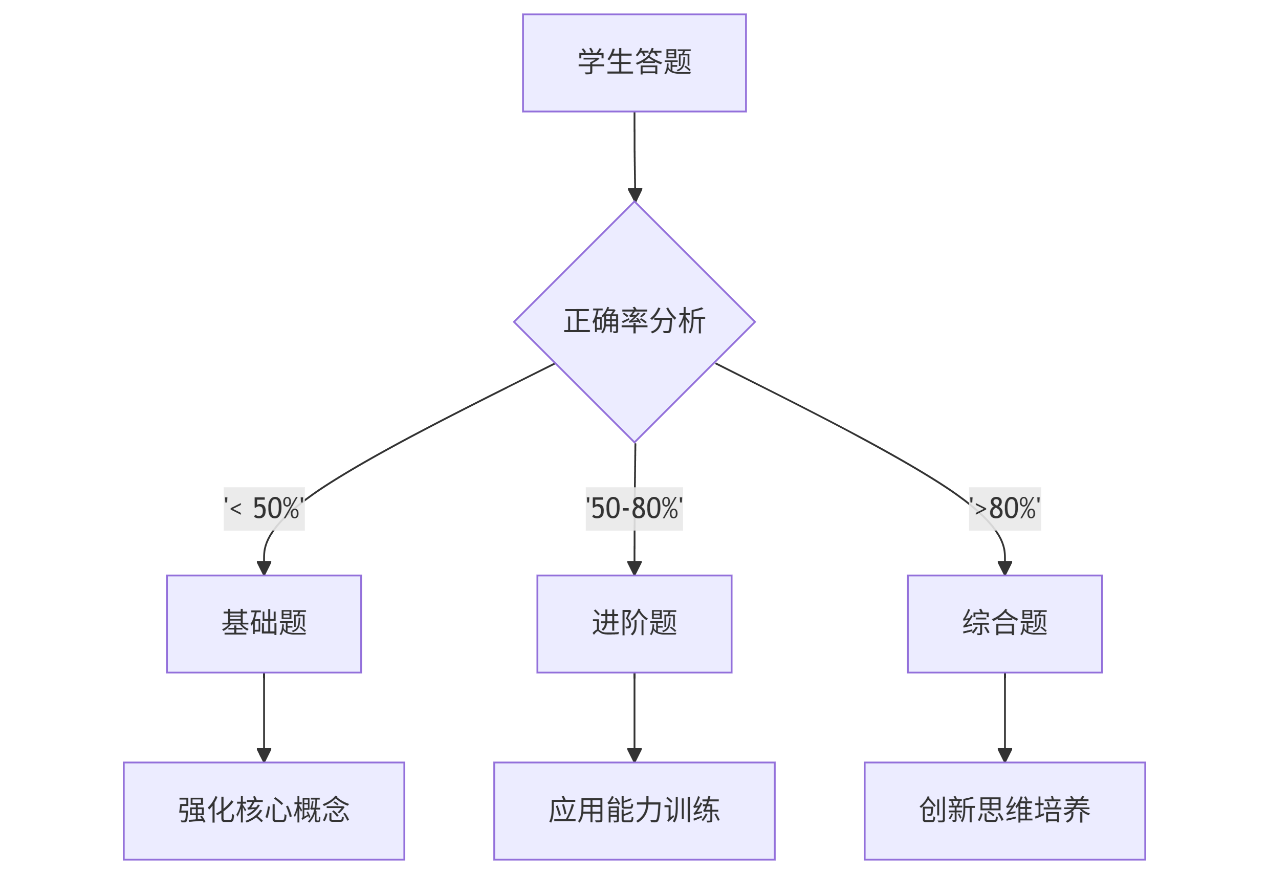


图 10动态难度调整

#### 2.2.4.教学效率分析引擎

// AISmartAnalysisService.java

public String getTeachingEfficiencyAnalysis(Long teacherId) {

// 1. 获取基础数据

EfficiencyAnalysis efficiency = efficiencyService.analyze(teacherId);

Teacher teacher = teacherService.getTeacher(teacherId);

// 2. 构建多维度分析提示

String prompt = buildEfficiencyPrompt(teacher, efficiency);

// 3. 调用大模型生成报告

String response = adminChatModel.call(prompt);

// 4. JSON结果清洗

return cleanAndExtractJson(response);

}

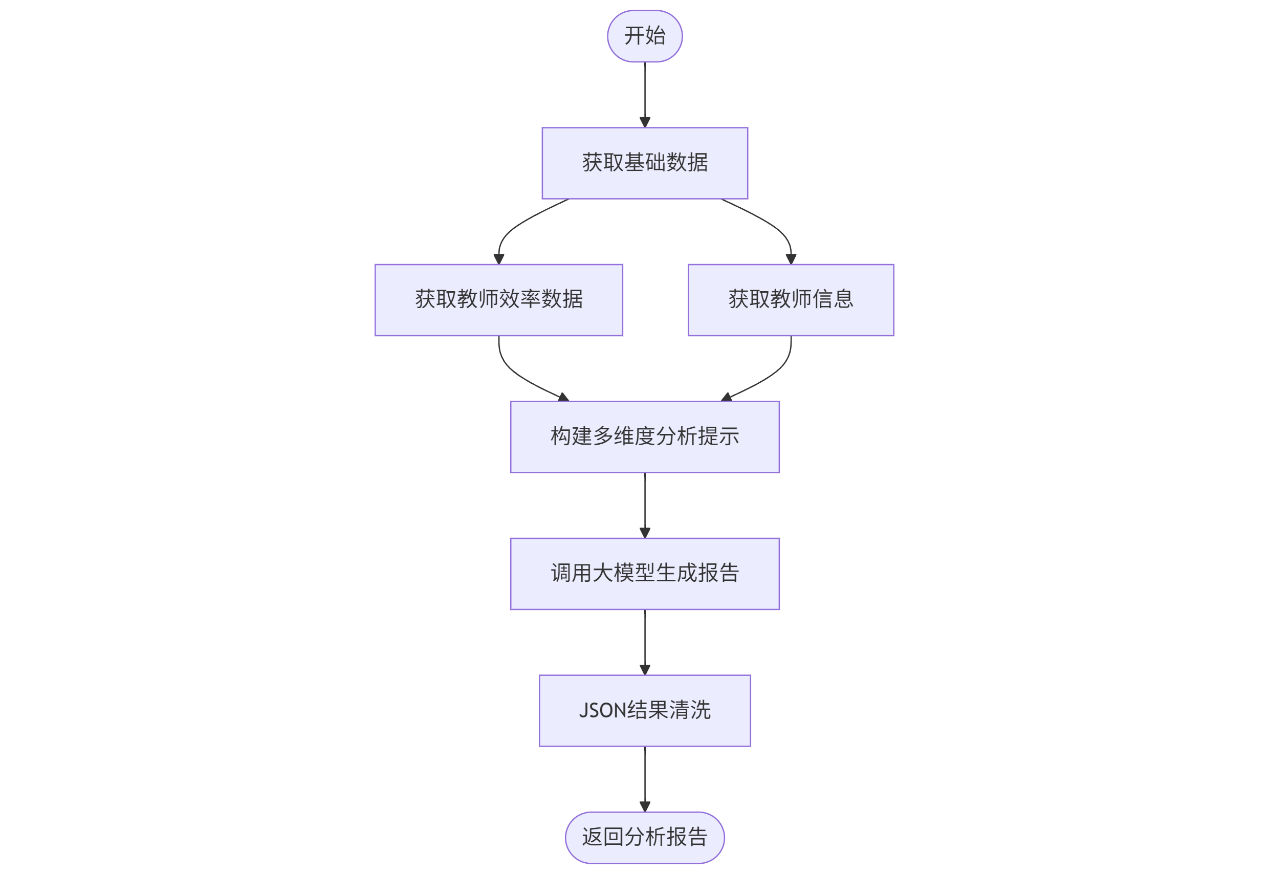


图 11教学效率分析

关键技术亮点说明：多源数据融合，结合结构化指标与非结构化教师信息；领域定制提示词工程；防御性清洗机制，敏感词过滤清单，JSON schema验证，异常降级处理（生成失败时返回基准报告）。

### 2.3后端系统功能设计

#### 2.3.1核心功能模块



#### 2.3.2典型使用流程

系统典型使用流程如图 12典型使用流程所示。

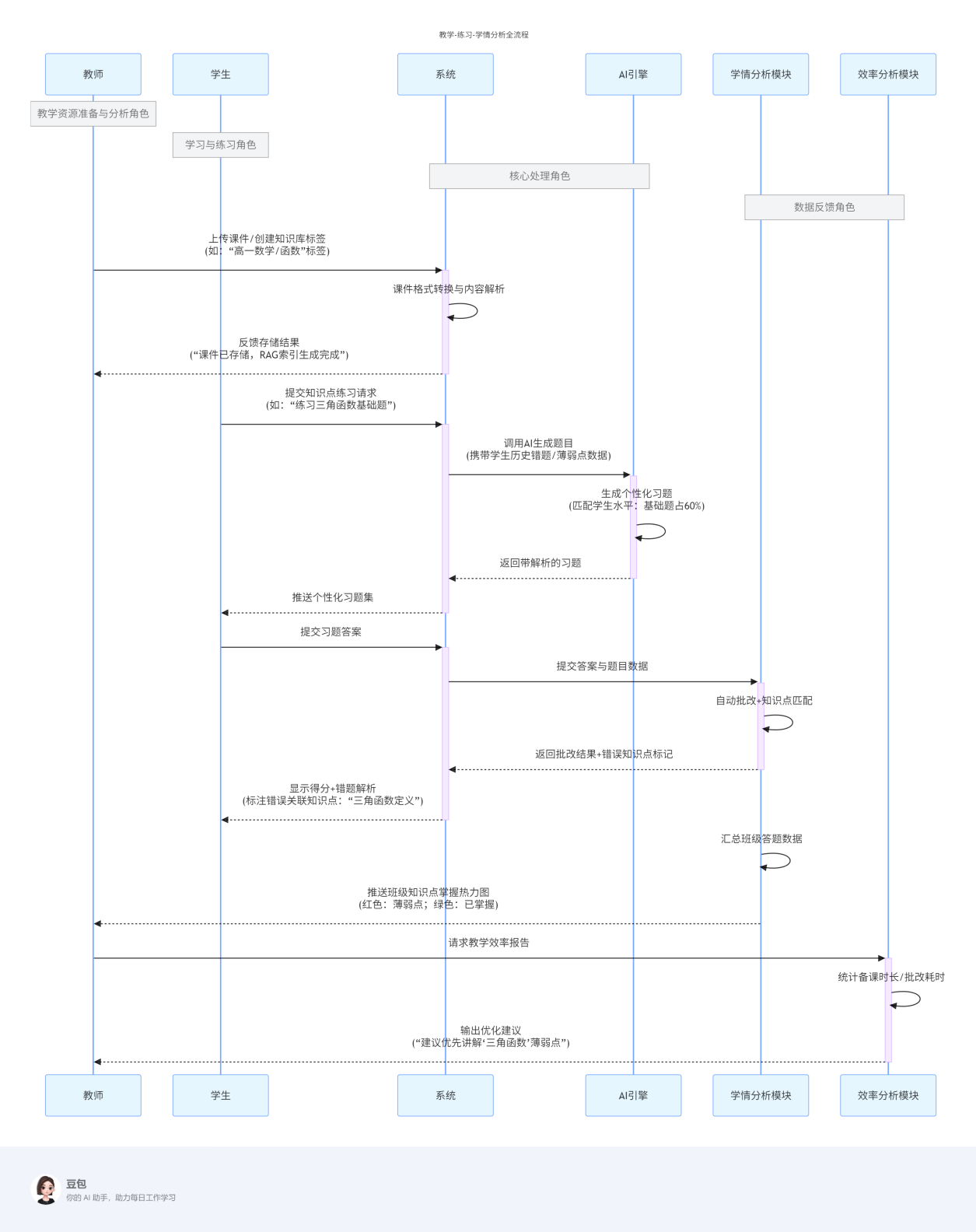


图 12典型使用流程

### 2.4后端技术难点解决方案

#### 2.4.1大模型输出控制

创新点：双层清洗机制（正则过滤+JSON提取）

实现：AISmartAnalysisService.cleanAndReturnJson()如下所示

private String cleanAndReturnJson(String response) {  
 try {  
 // 1. 清洗响应  
 String cleaned = response  
 .replaceAll("<think>.\*?</think>", "") // 移除思考标签  
 .replaceAll("([^\\\\])\"([^\"])", "$1\\\\\"$2") // 修复未闭合引号  
 .replaceAll("'([^']\*)'", "\"$1\""); // 单引号转双引号  
 // 2. 提取JSON部分  
 Pattern pattern = Pattern.*compile*("\\{(?:[^{}]|\\{(?:[^{}]|\\{[^{}]\*\\})\*\\})\*\\}");  
 Matcher matcher = pattern.matcher(cleaned);  
 if (matcher.find()) {  
 String jsonContent = matcher.group(0);  
 // 3. 验证并格式化JSON  
 Object parsed = objectMapper.readValue(jsonContent, Object.class);  
 return objectMapper.writeValueAsString(parsed);  
 }  
 throw new RuntimeException("未找到有效JSON内容");  
 } catch (Exception e) {  
 *log*.error("JSON处理失败", e);  
 return buildErrorJson("JSON处理失败: " + e.getMessage());  
 }  
}

#### 2.4.2跨场景知识融合

方案：RAG检索时融合历史记录，伪码如下所示

// AiServiceStudentimpl中融合向量库与历史记录

List<Message> messages = new ArrayList<>();

messages.add(new UserMessage(question));

messages.add(ragMessage); // 向量库文档

messages.add(historyMessage); // 历史练习数据

### 2.5 模型集成设计

#### 2.5.1 多角色模型配置

本系统为不同用户角色配置了独立的模型实例，优化资源分配：

// 管理员模型 (32B大模型)

@Bean(name = "AdminChatModel")  
 public OllamaChatModel adminChatModel(@Value("${spring.ai.ollama.base-url}") String baseUrl){  
 OllamaApi ollamaApi = OllamaApi.*builder*()  
 .baseUrl(baseUrl)  
 .build();  
 return OllamaChatModel.*builder*()  
 .ollamaApi(ollamaApi)  
 .defaultOptions(  
 OllamaOptions.*builder*()  
 .topK(40)//控制采样个数  
 .topP(0.9)//采样概率  
 .frequencyPenalty(0.1)//重复词惩罚  
 .presencePenalty(0.2)//新词奖励  
 .model("deepseek-r1:32b")  
 .temperature(0.5)//温度控制  
 .build())  
 .build();  
 }

// 学生模型 (7B轻量模型)

@Bean(name = "studentChatModel")

public OllamaChatModel studentChatModel() {

return OllamaChatModel.builder()

.model("deepseek-r1:7b") // 高效响应模型

.temperature(0.3) // 低随机性

.build();

}

场景适配：

教师/管理员端：使用32B大模型处理复杂任务（课程设计、试卷生成学情分析等）

学生端：使用7B轻量模型实现低延迟交互（题目生成与解答）

#### 2.5.2流式响应集成

核心业务采用流式响应实现如下，提升用户体验：

// 流式RAG问答

@GetMapping("/generate\_stream\_rag")

public Flux<ServerSentEvent<ChatResponse>> generateStreamRag(

@RequestParam String ragTag,

@RequestParam String message) {

return aiService.generateStreamRag(ragTag, message)

.map(response -> ServerSentEvent.builder(response).build());

}

应用场景：

实时课程内容生成 (OllamaControllerForTeacher)

动态题目生成 (OllamaStudentController)

渐进式答案批改 (CheckAnswerController)

#### 2.5.3 RAG知识动态增强

多模块集成RAG实现知识定制化，实现如下：

// RAG文档上传

@PostMapping("/upload\_file")

public Response<String> uploadFile(

@RequestParam String ragTag,

@RequestParam List<MultipartFile> files) {

return ragService.uploadFile(ragTag, files);

}

**知识库应用**：

|  |  |  |
| --- | --- | --- |
| **模块** | **功能** | **知识库用途** |
| **教师课程设计** | 课程大纲生成 | 学科知识深度整合 |
| **智能组卷** | 试题生成 | 考点精准匹配 |
| **学生问答** | 知识点答疑 | 个性化学习资料支持 |
| **答案批改** | 答案验证 | 标准答案库参照 |

#### 2.5.4.参数精细化控制

通过OllamaOptions实现生成控制，实现如下：

OllamaOptions.builder()

.topK(40) // 限制采样候选集

.topP(0.9) // 控制结果多样性

.frequencyPenalty(0.1) // 抑制重复内容

.presencePenalty(0.2) // 鼓励新概念引入

.temperature(0.5) // 平衡创意与准确性

.build()

（1）业务映射：

高频惩罚：试卷生成时避免相似题目重复

新词奖励：课程设计中鼓励引入新教学案例

（2）温度控制：

高值(0.8+)：创意课程设计

低值(0.2-)：标准化试题生成

#### 2.5.5.模型路由机制

通过@Qualifier实现服务级模型路由：

// 教师/管理员服务使用32B模型

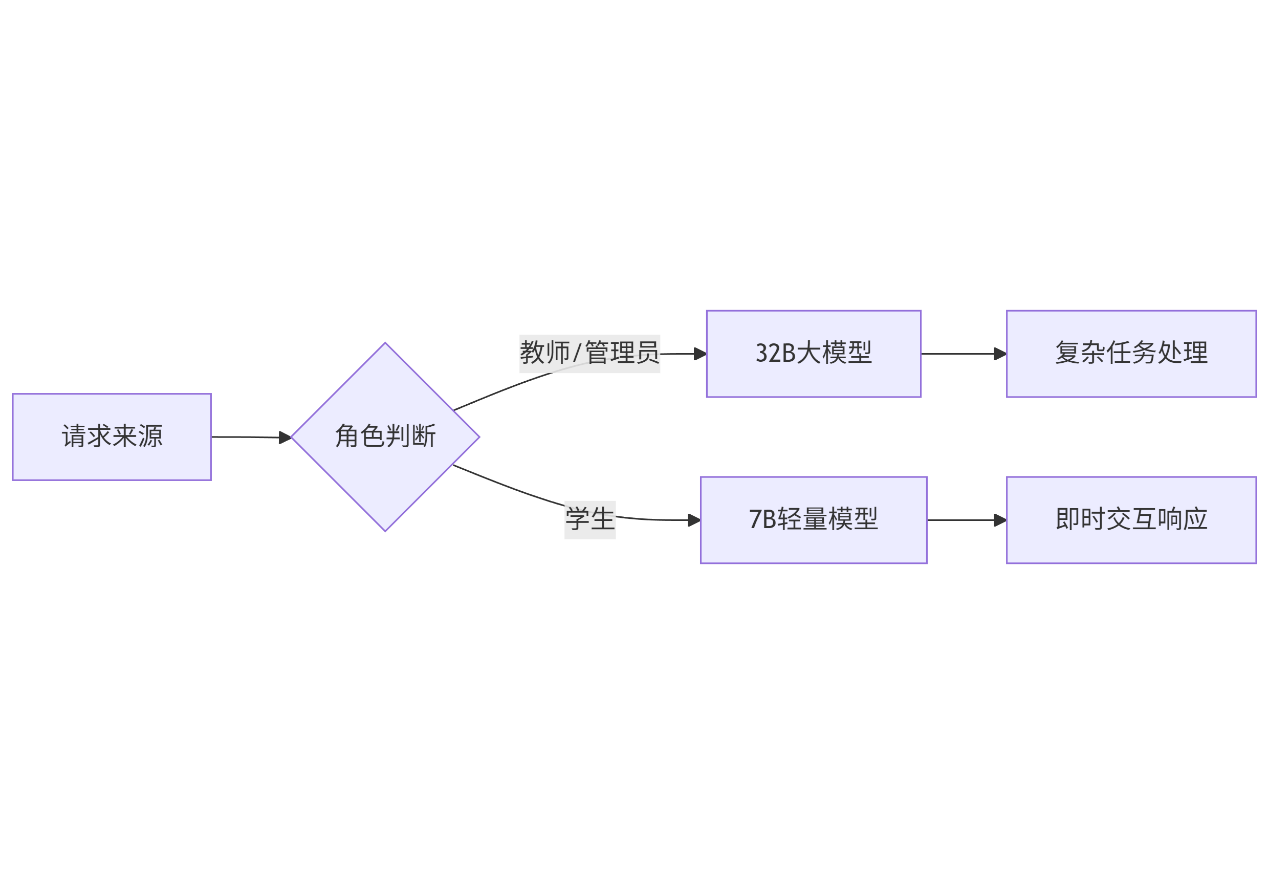
@Autowired @Qualifier("AdminChatModel")

private OllamaChatModel teacherModel;

// 学生服务使用7B模型

@Autowired @Qualifier("studentChatModel")

private OllamaChatModel studentModel;



**典型集成场景**

|  |  |  |
| --- | --- | --- |
| **场景** | **技术栈组合** | **关键优势** |
| **智能课程设计** | RAG+32B模型+流式响应 | 结合教材生成结构化大纲 |
| **动态试卷生成** | 参数控制+多题型约束 | 精准匹配难度与知识点 |
| **学生答案批改** | 7B模型+历史记录分析 | 实时反馈+学习轨迹跟踪 |
| **个性化题目推荐** | RAG+向量检索+7B模型 | 基于薄弱点定向强化 |

#### 2.5.6未来扩展设计

##### 2.5.6.1模型热切换：

通过application.properties动态调整模型版本

##### 2.5.6.2分级部署：

教学管理端：GPU服务器部署32B模型

学生交互端：CPU服务器部署7B模型

##### 2.5.6.3混合推理：

// 根据内容复杂度自动路由

if (contentComplexity > THRESHOLD) {

return adminModel.generate(content);

} else {

return studentModel.generate(content);

}

### 2.6前端设计

#### 2.6.1 前端ER关系

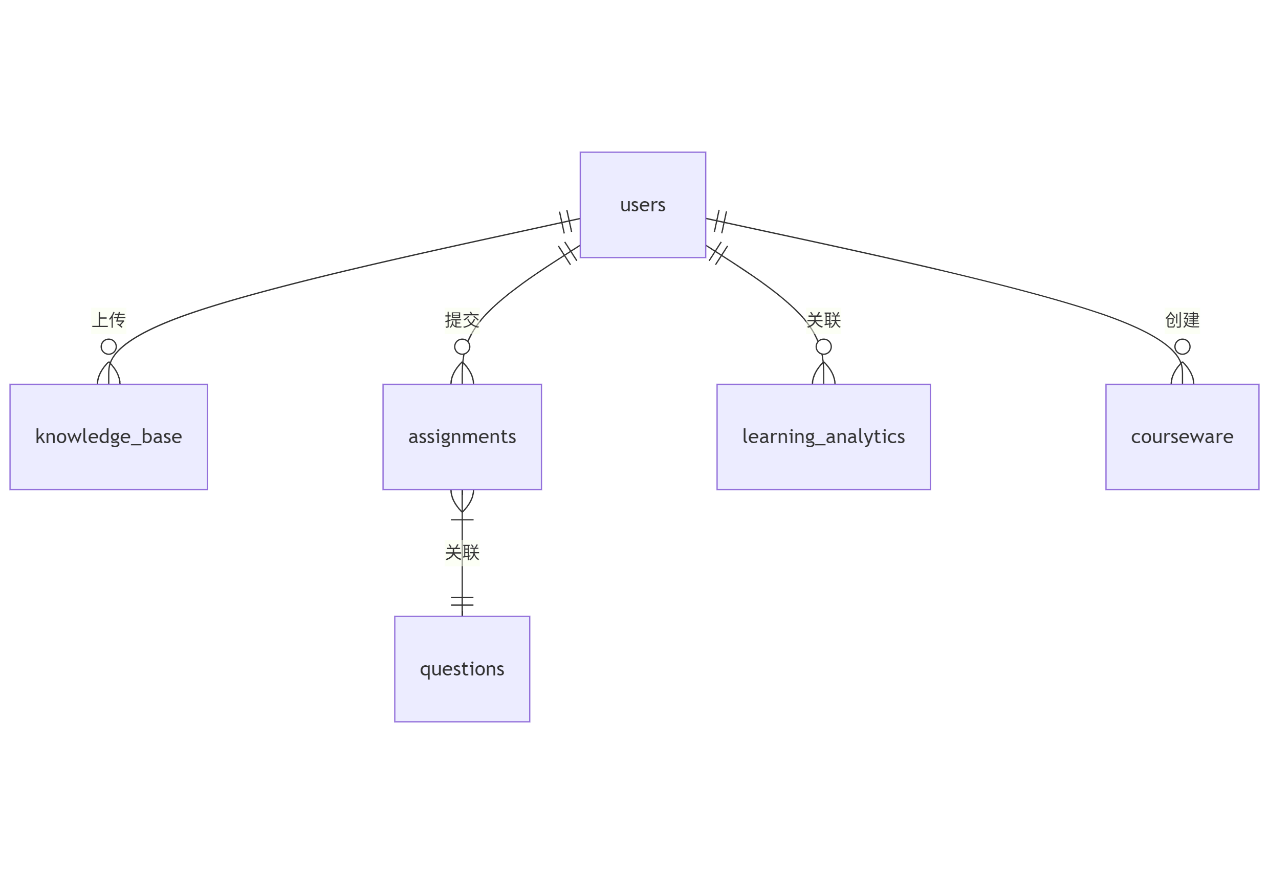


图 13前端ER关系

#### 2.6.2关键前端算法设计

##### 2.6.2.1. 流式响应处理

// SSE实时数据流处理

const eventSource = new EventSource('/api/stream');

eventSource.onmessage = (event) => {

const data = JSON.parse(event.data);

renderMarkdownContent(data.text); // 实时渲染

};

##### 2.6.2.2. 智能代码分析

def analyze\_code(student\_code, reference\_code):

# 语法树差异分析

ast\_diff = compare\_ast(student\_code, reference\_code)

# 逻辑错误检测

logic\_errors = detect\_logic\_errors(student\_code)

# 代码风格评分

style\_score = evaluate\_coding\_style(student\_code)

return {

'ast\_diff': ast\_diff,

'logic\_errors': logic\_errors,

'style\_score': style\_score

}

##### 2.6.2.3. 知识点关联推荐

def recommend\_resources(current\_topic):

# 知识图谱查询

related\_nodes = knowledge\_graph.query(

f"MATCH (c:Concept)-[:RELATED]->(r)

WHERE c.name='{current\_topic}'

RETURN r"

)

# 资源相关性排序

return sorted(related\_nodes,

key=lambda x: x.relevance\_score,

reverse=True)[:5]

##### 2.6.2.4 跨平台文档生成

function generatePDF(content) {

// 分页智能分割算法

const pages = adaptive\_pagination(content);

// 教育主题模板

apply\_education\_template();

// 交互式元素保留

preserve\_interactive\_elements();

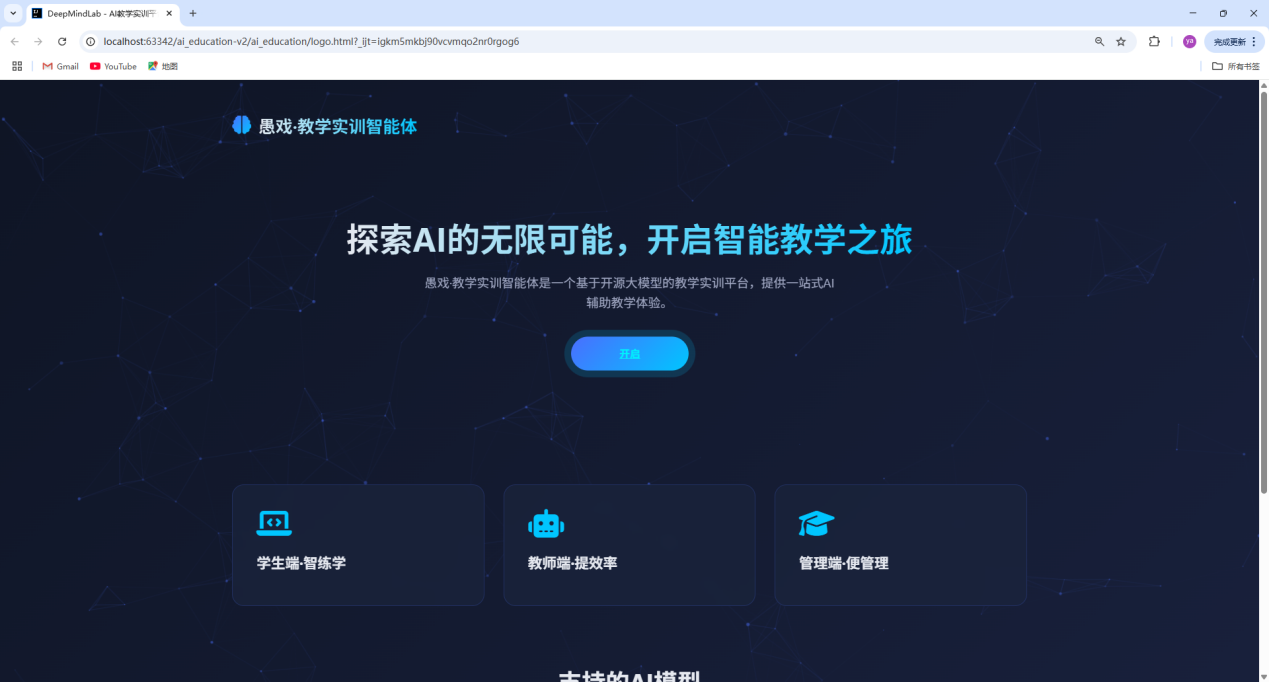
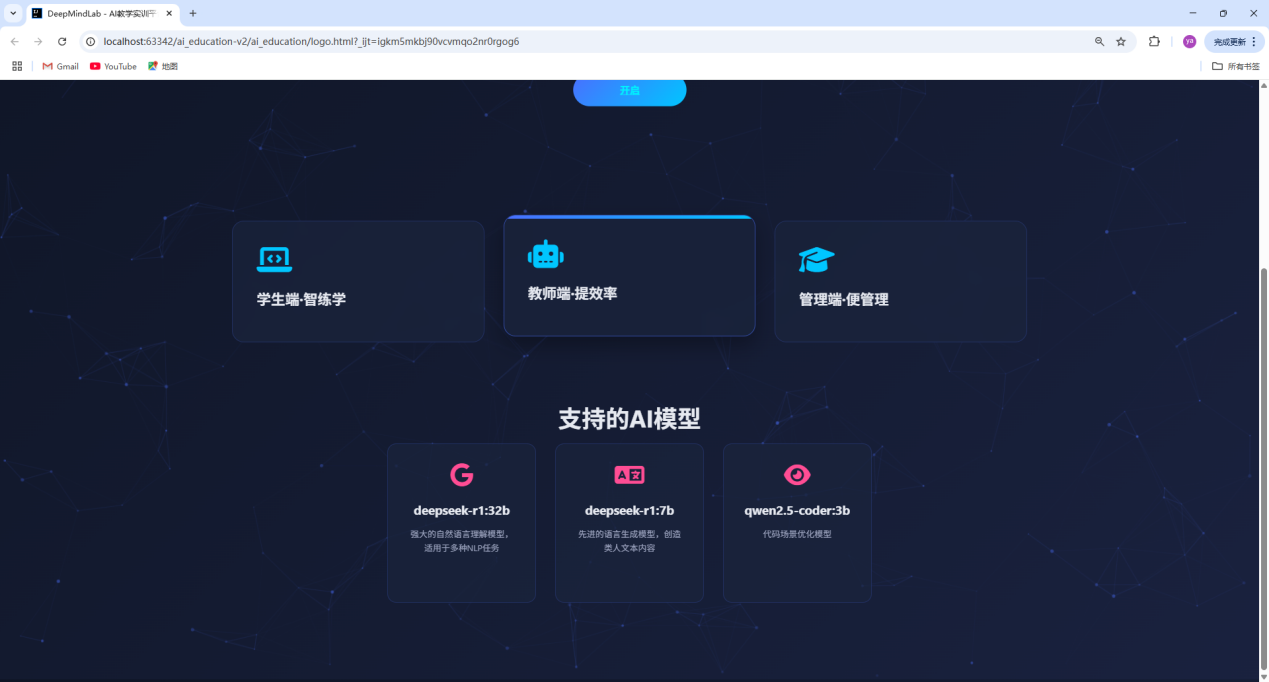
}

优势：

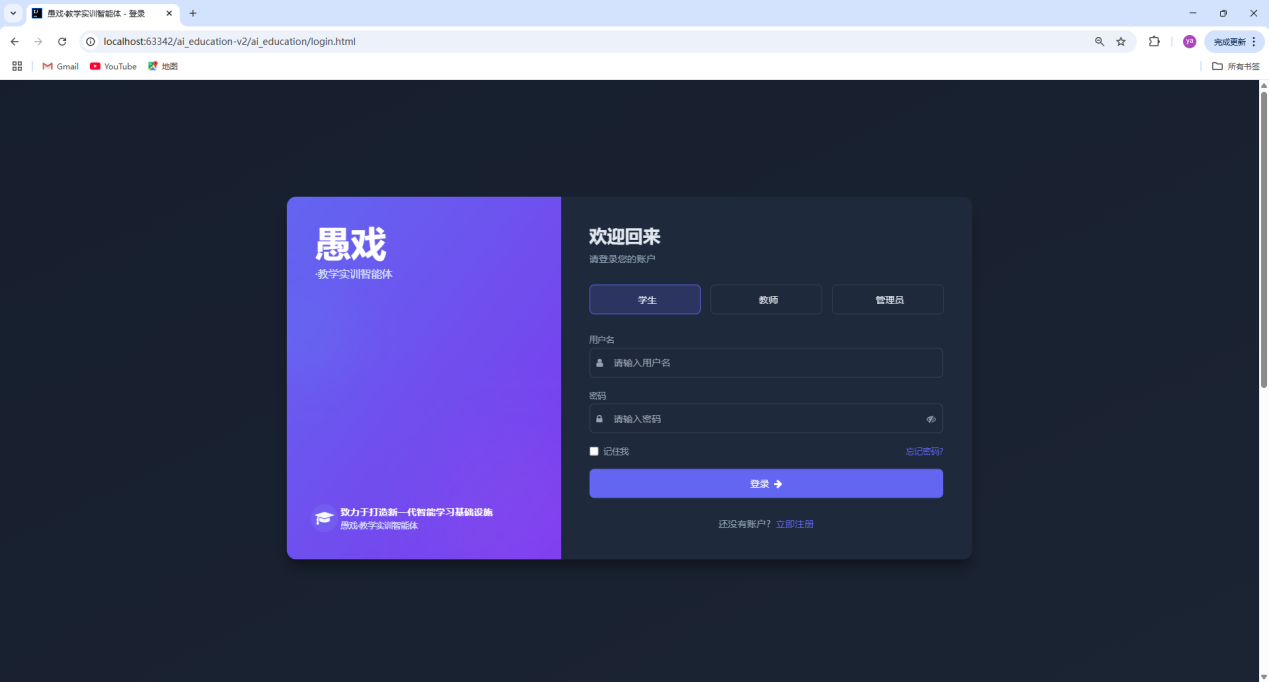
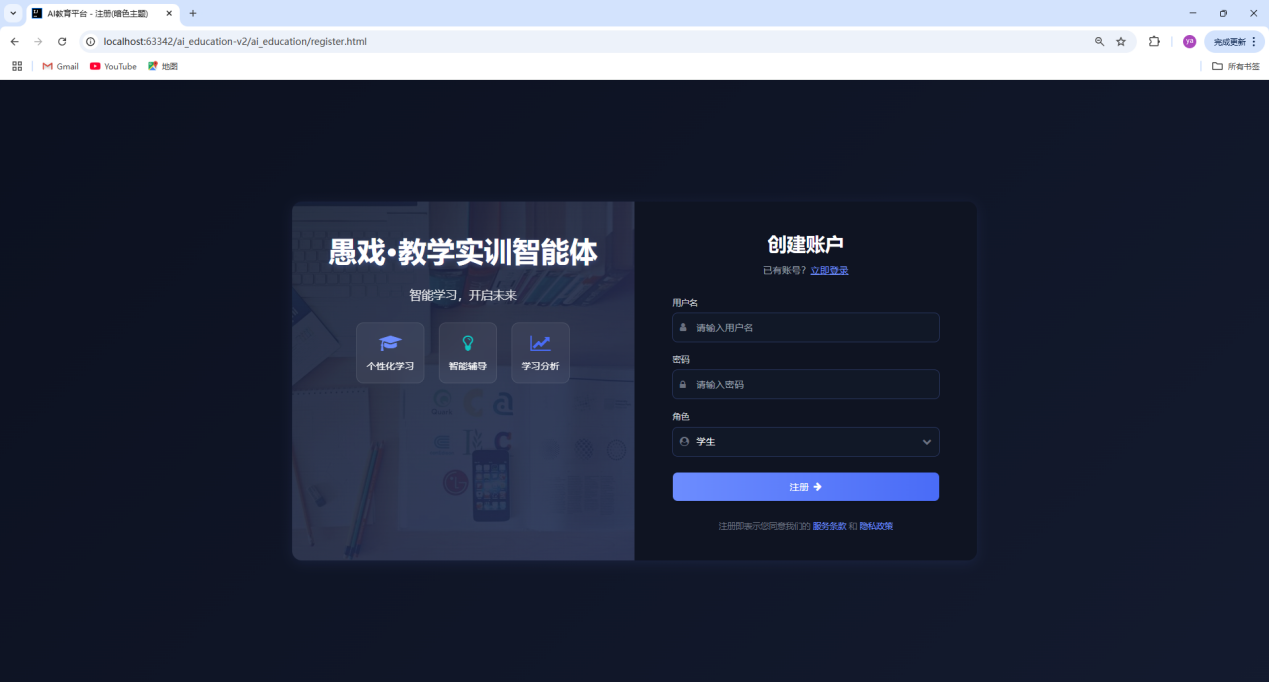
保持Markdown原始结构（代码块/表格/公式）、自动添加教学元数据（课时/教学目标）和响应式布局（PC/移动双适配）

### 2.7前端界面设计

#### 2.7.1 初始界面

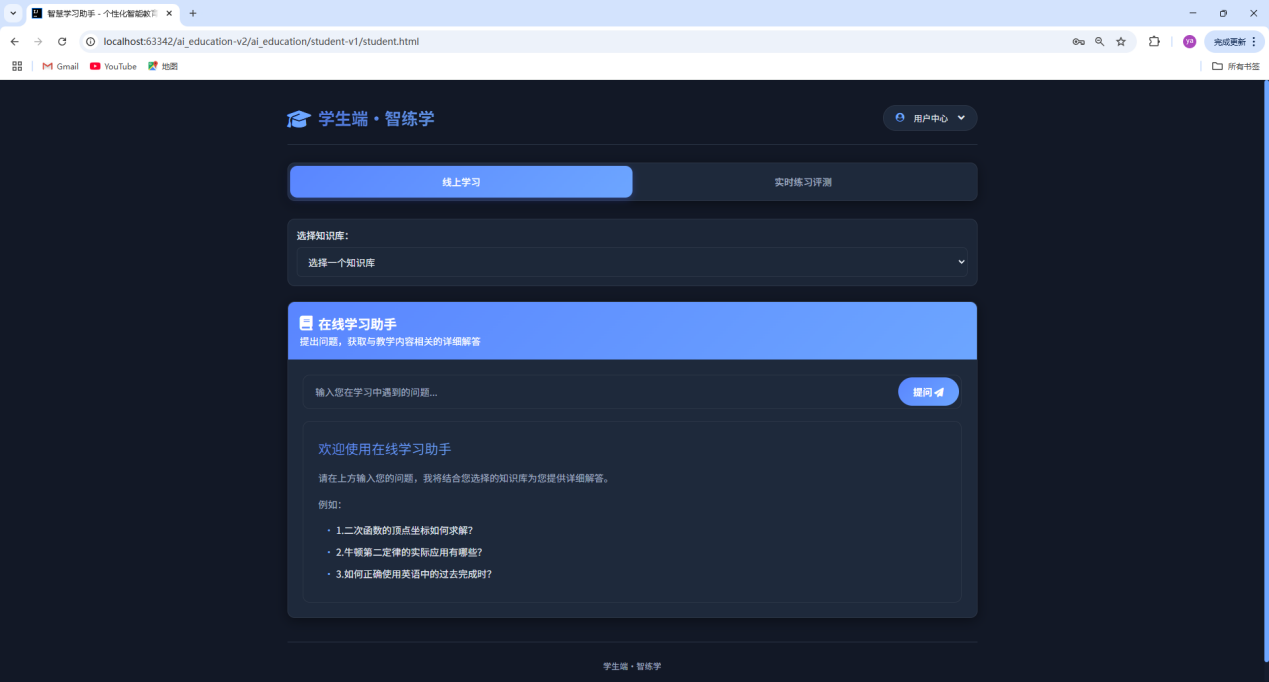
 

#### 2.7.2登录与注册用户界面

#### 2.7.3学生端界面

##### 2.7.3.1线上学习界面



##### 2.7.3.2实时练习评测界面

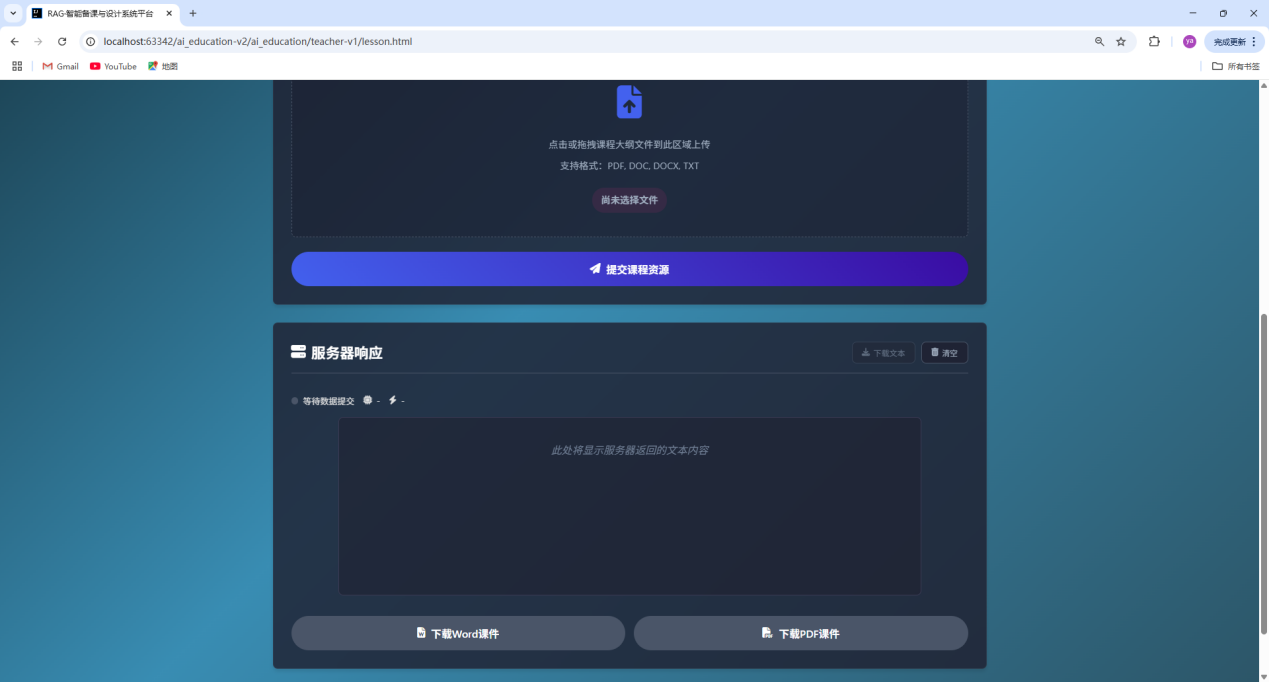
#### 2.7.4教师端界面

##### 2.7.4.1 教师端导航界面

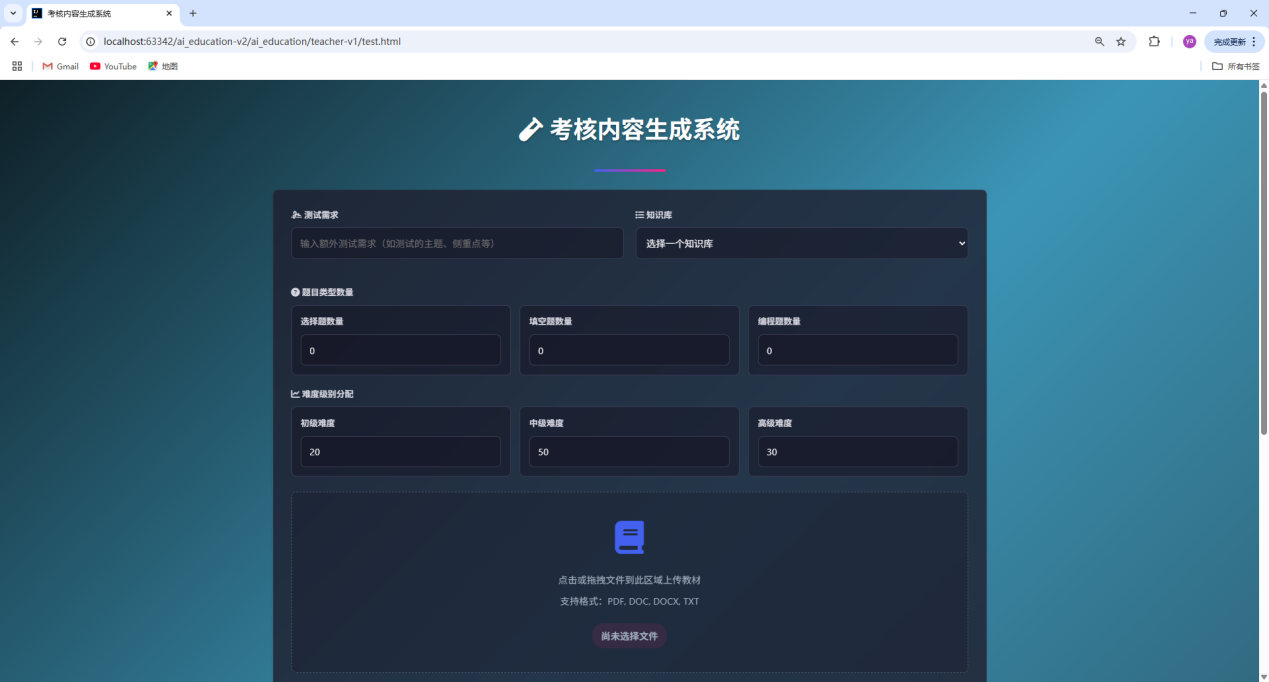
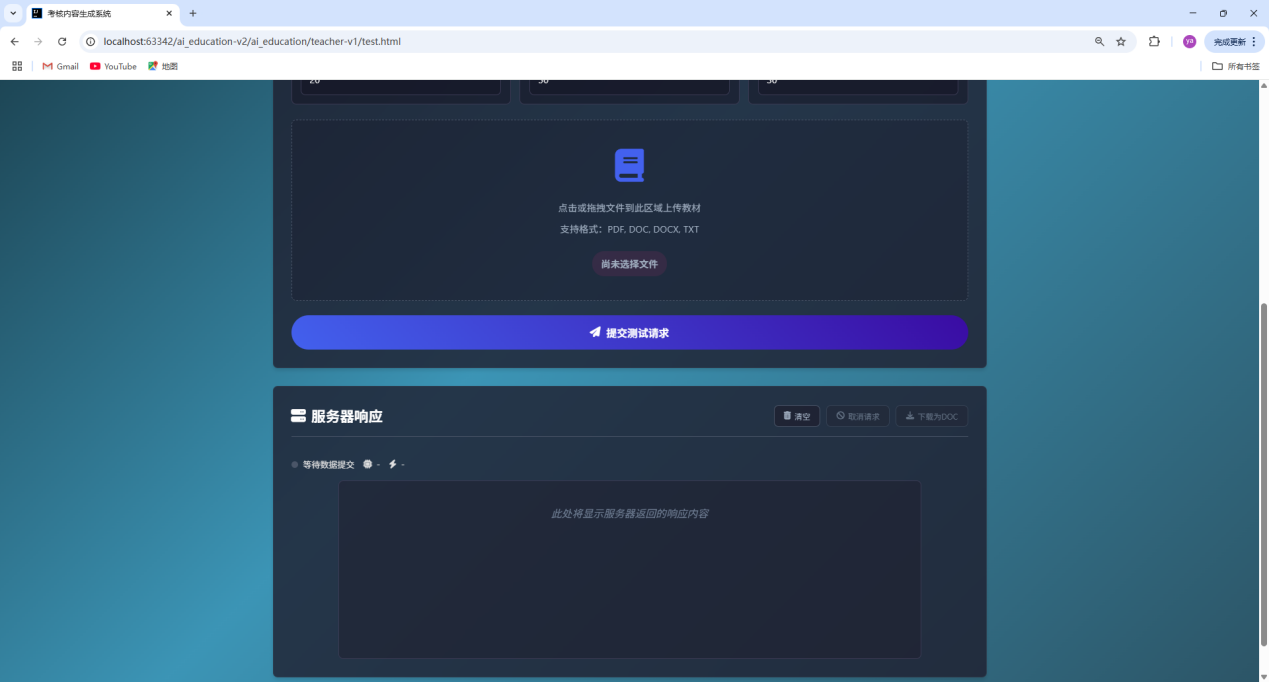


##### 2.7.4.2智能备课系统界面

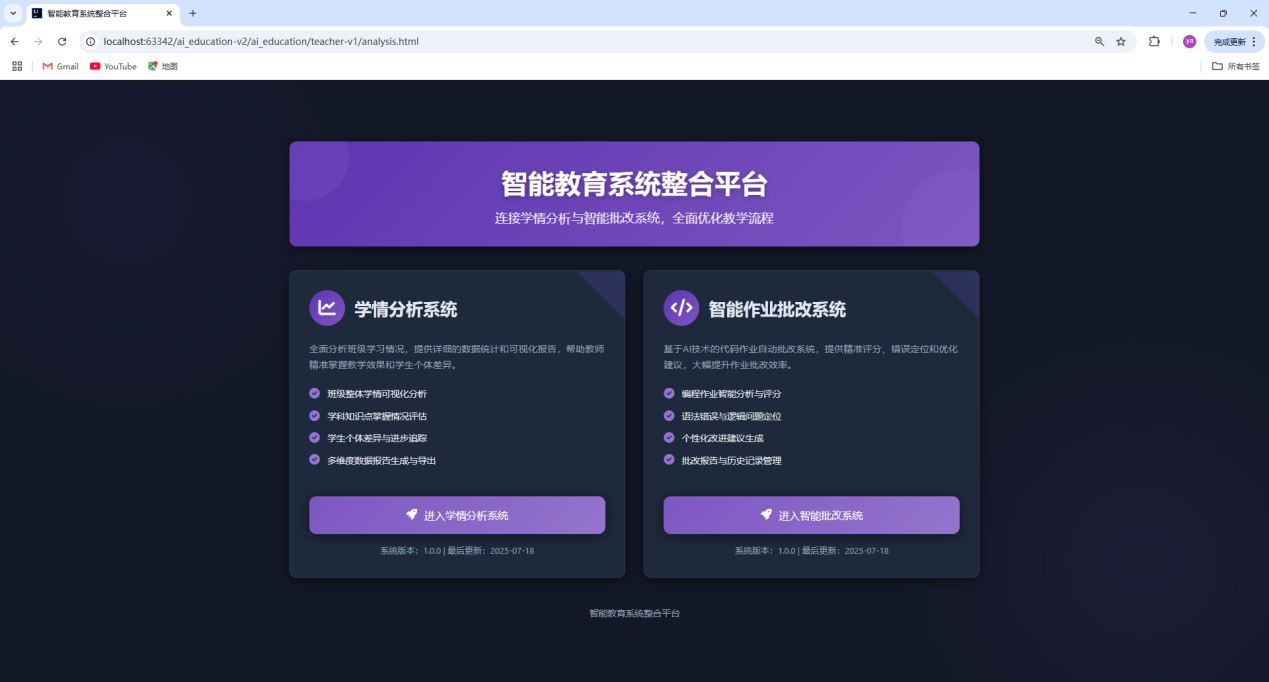




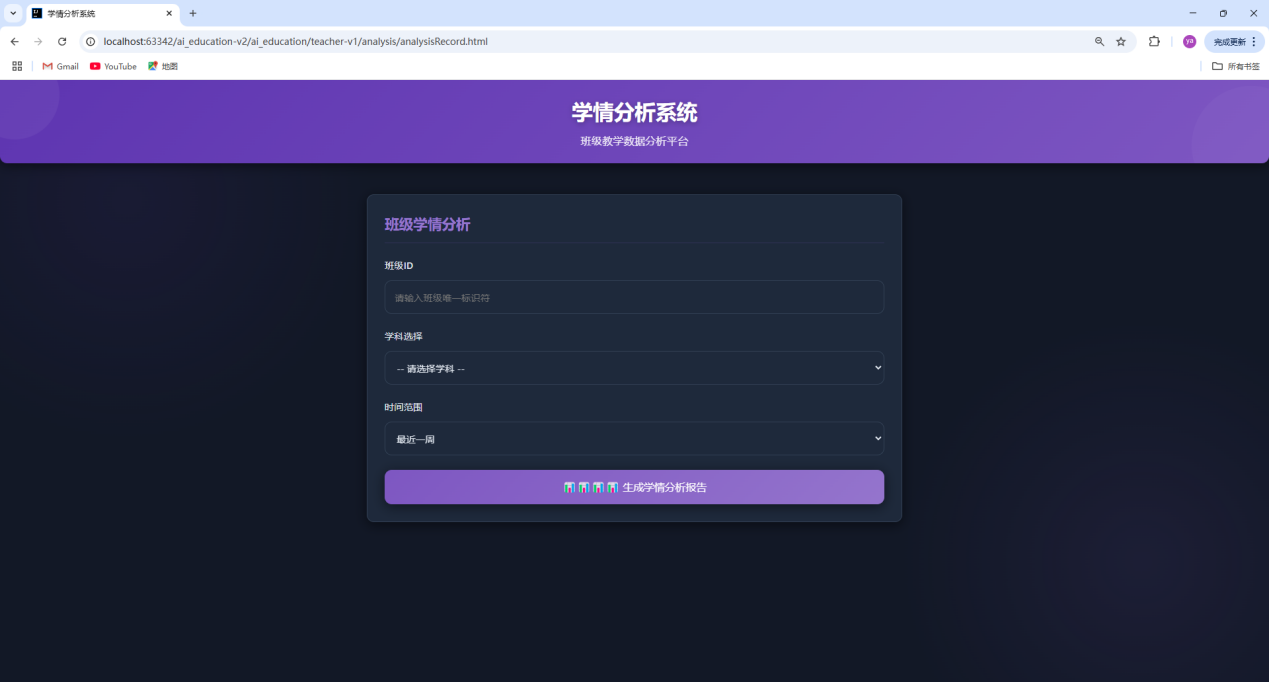
##### 2.7.4.3 考核生成系统界面

##### 2.7.4.4学情分析与作业批改系统导航界面



##### 2.7.4.5学情分析界面



##### 2.7.4.6作业批改系统界面

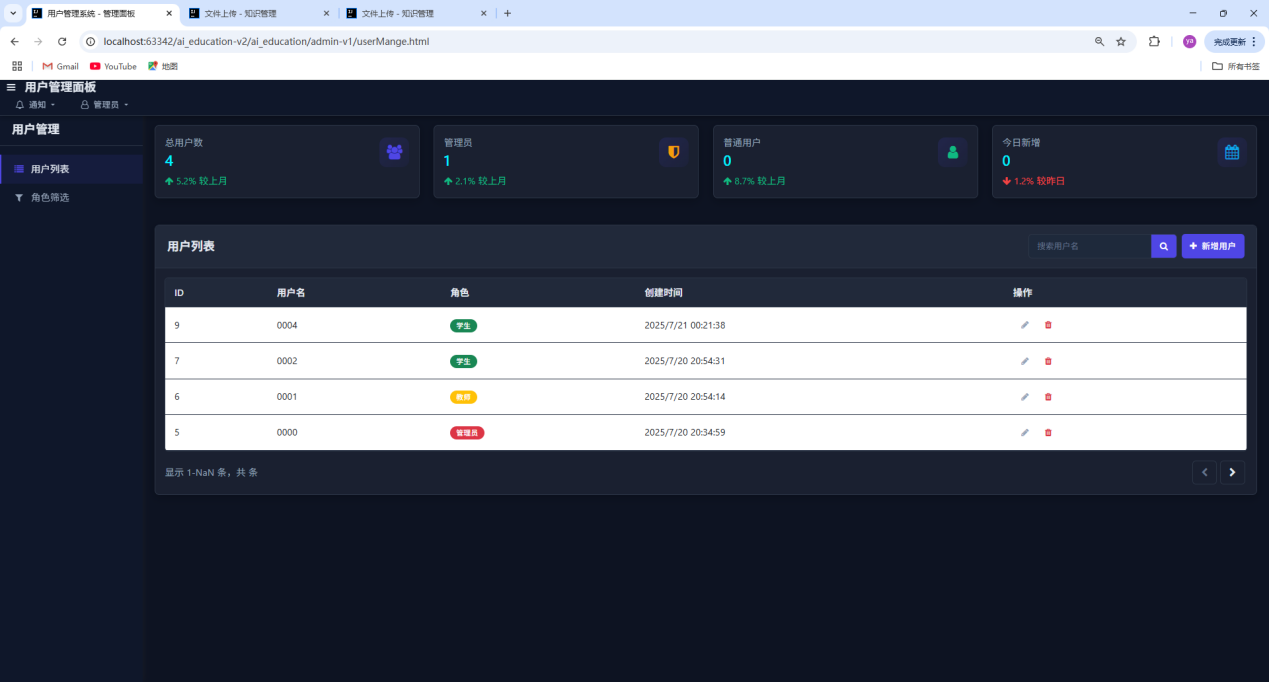


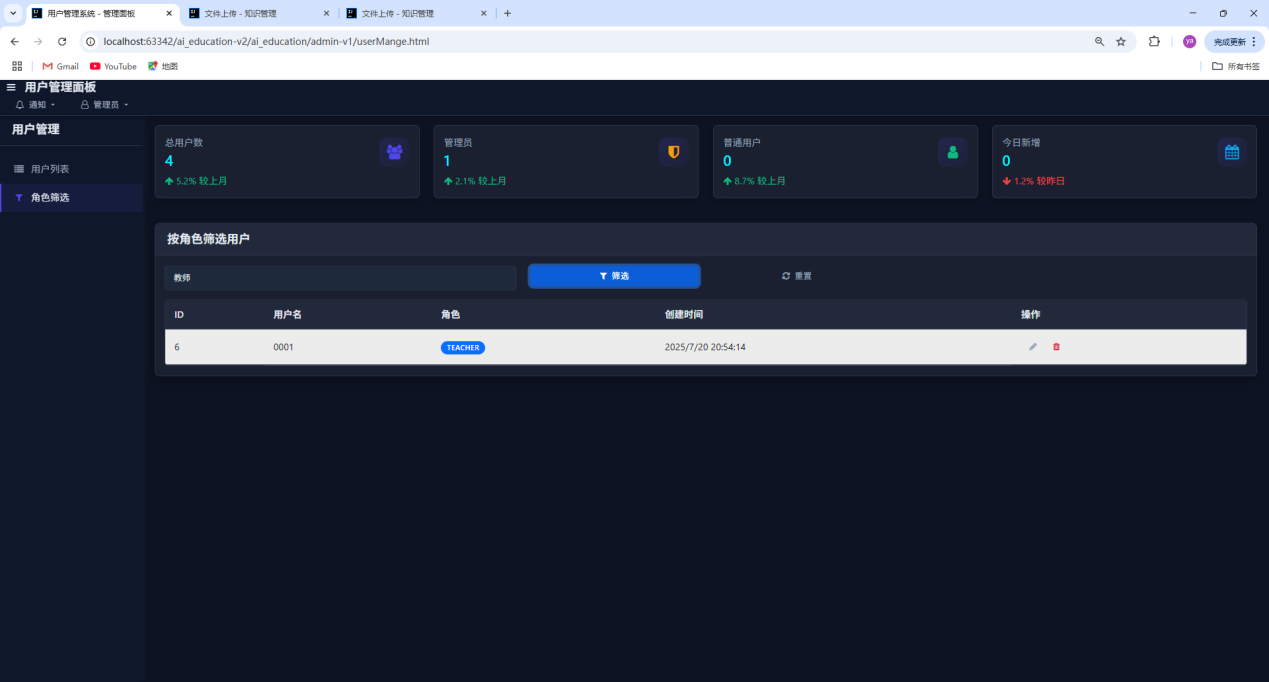


#### 2.7.5 管理端界面

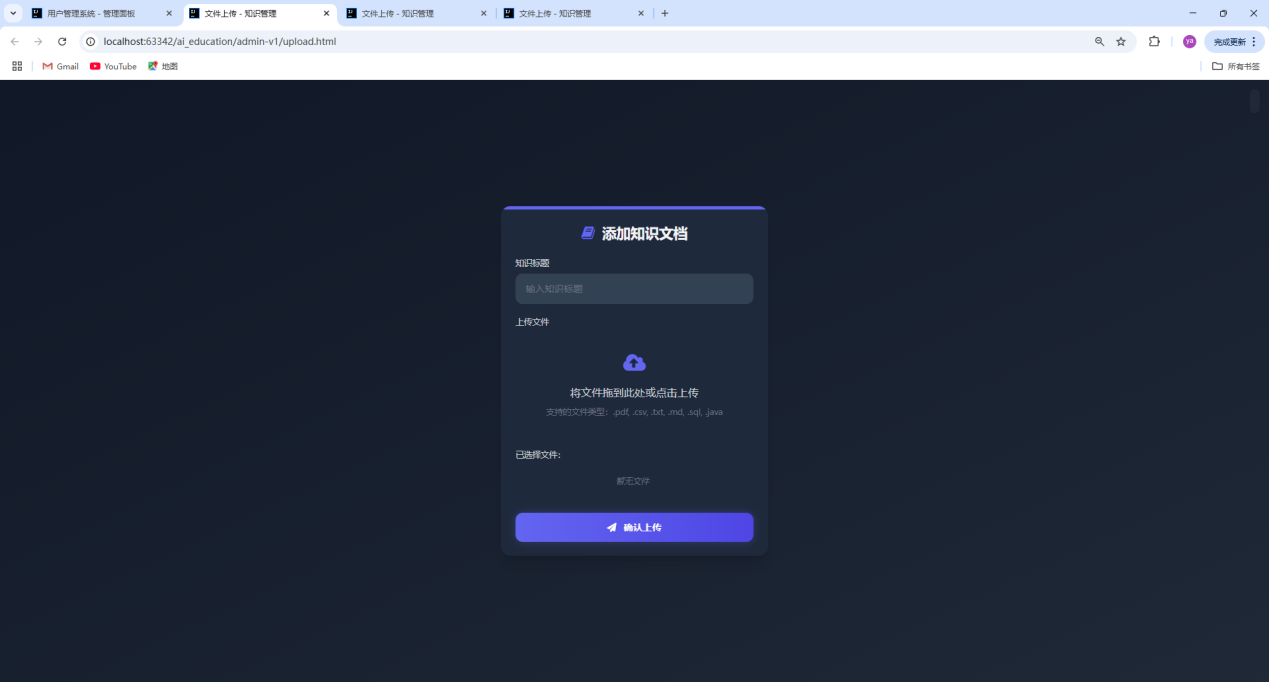


##### 2.7.5.1用户管理系统界面

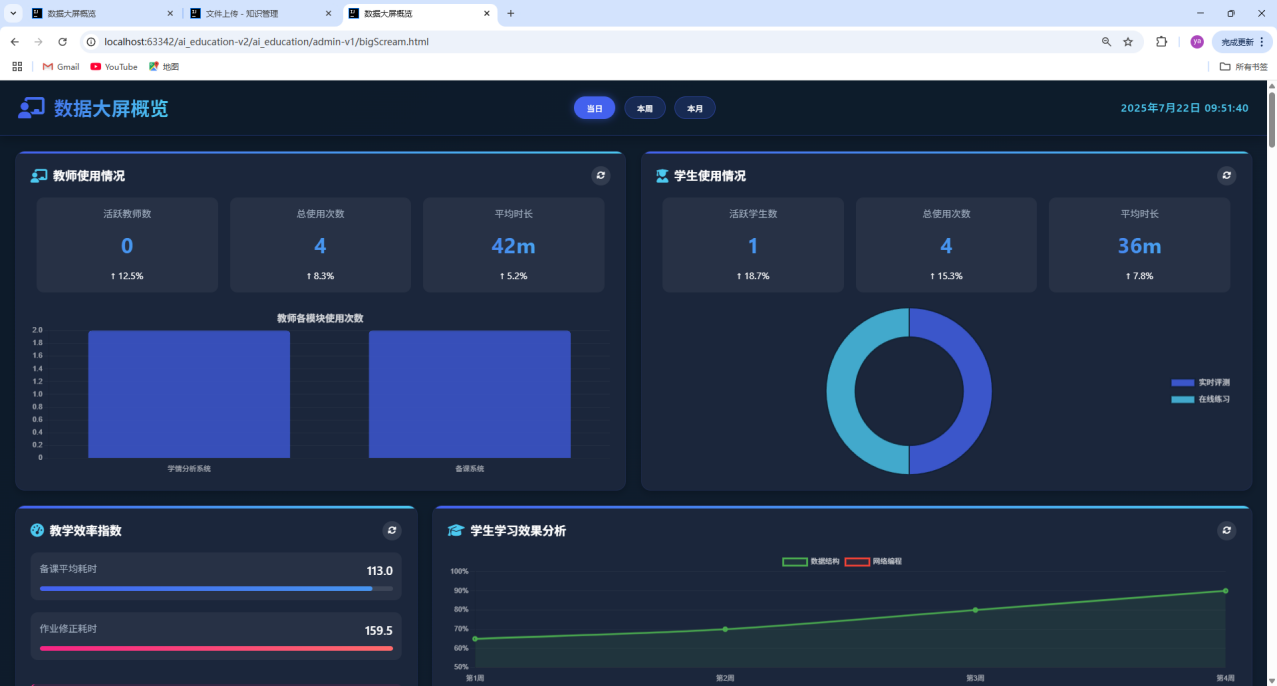


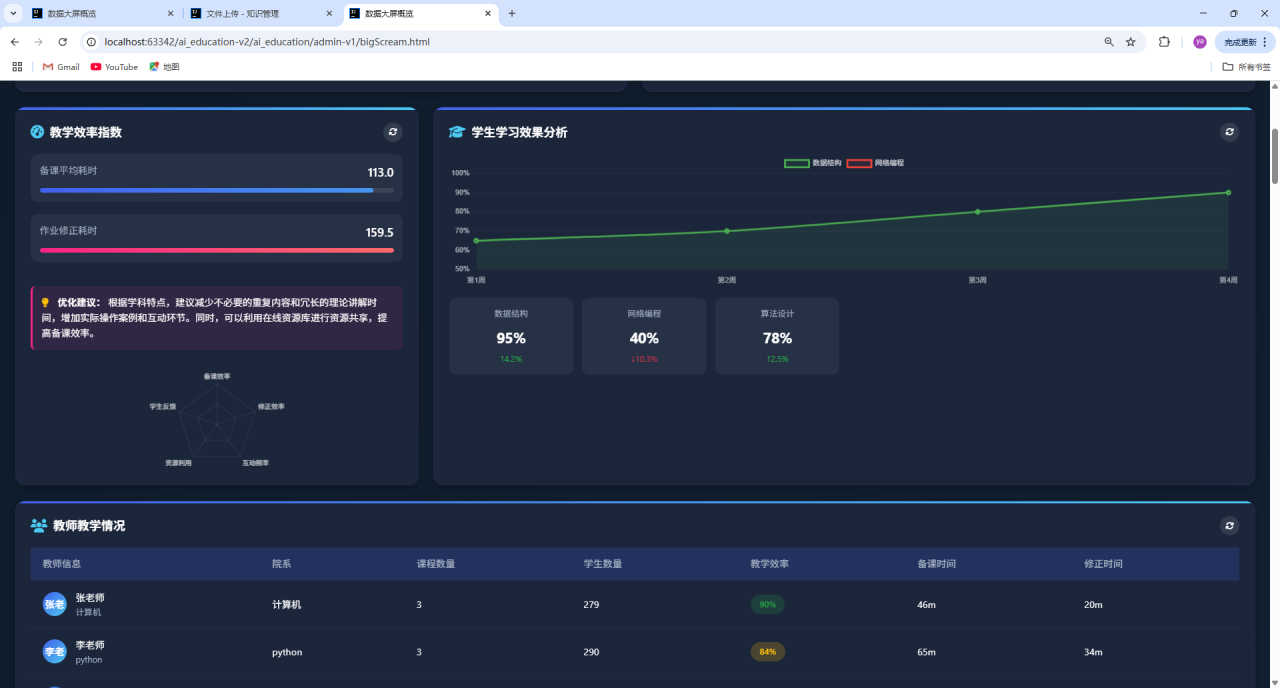


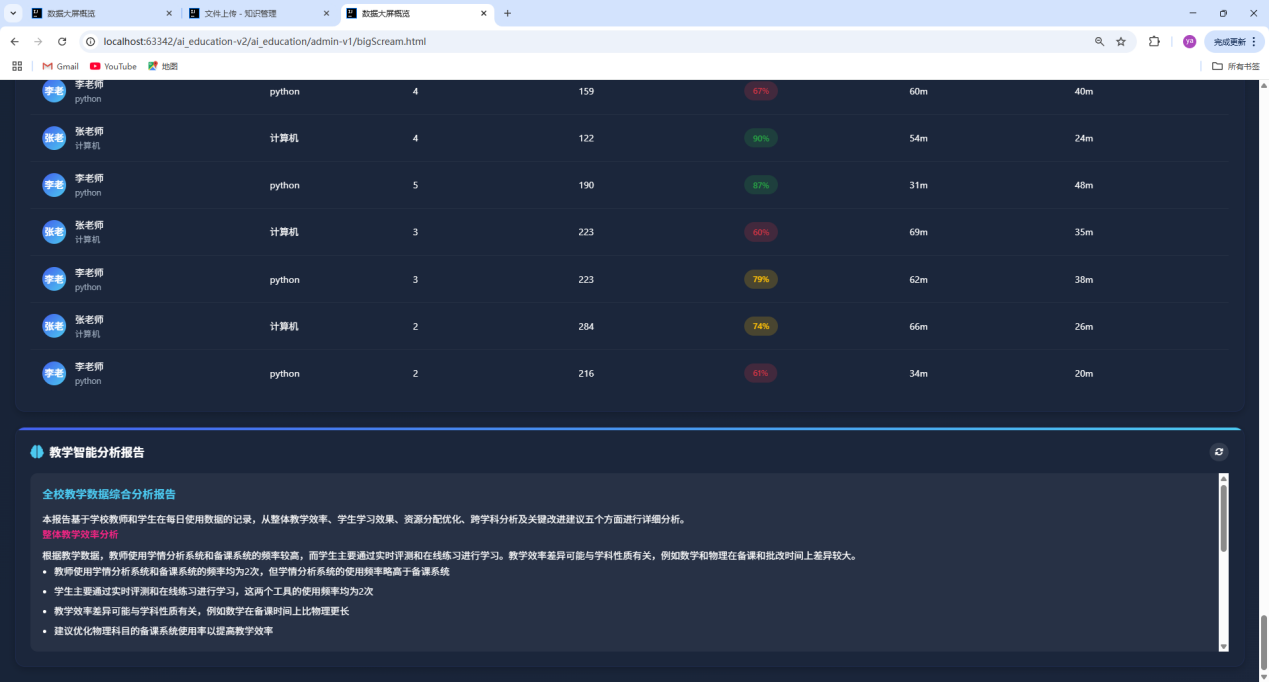
##### 2.7.5.2知识库动态更新系统界面



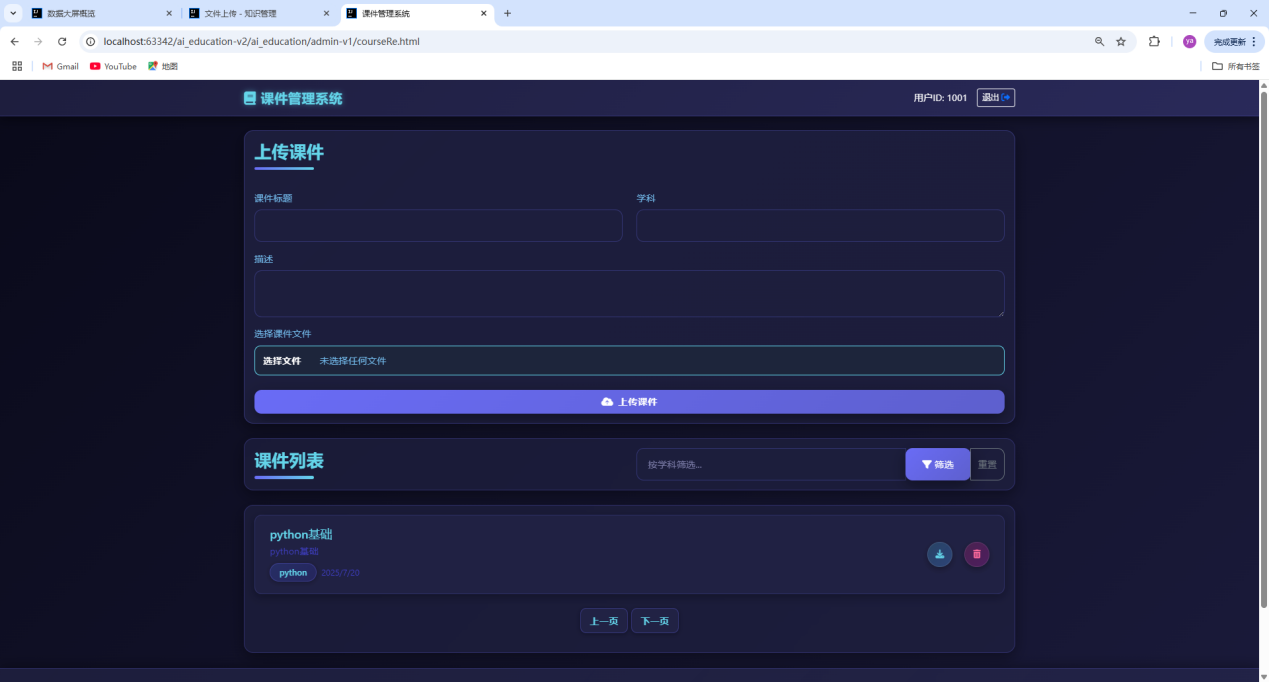
##### 2.7.5.3数据大屏界面



****



##### 2.7.5.4课件管理系统界面



# **产品设计严格满足赛题需求说明**

### 3.1 非功能性需求满足说明

#### 3.1.1需明确使用至少1个开源大模型作为核心技术组件(开源模型使用说明)

使用的开源模型：



Ollama 集成的 DeepSeek、Qwen模型在系统中扮演以下核心角色：

1. ​​智能内容生成，​​括号内为对应实现类

​​课件生成​​（OllamaServiceTeacherImpl）：根据课程大纲、知识库内容和教师需求，动态生成结构化教学课件（教学目标、知识点讲解、实训设计等）。

​​试题生成​​（OllamaServiceTeacherImpl）：基于知识点库、难度分布和题型要求，生成选择题、编程题等试题（含答案和评分标准）。

​​②自动化评估与反馈​​

​​作业批改​​（CheckAnswerServiceImpl）：评估学生代码质量，输出分数、知识点掌握情况和改进建议。示例：SYSTEM\_PROMPT\_CHECK\_ANSWER要求模型分析代码并给出结构化反馈。

​​答案判定​​（AnswerCorrectionService）：对比学生答案与参考答案，识别错误类型（概念错误、逻辑错误等）并提供纠正建议。

​​③智能问答与检索增强（RAG）​​

​​RAG 问答​​（AiServiceImpl）：结合知识库检索结果生成准确回答，支持翻译、查询重写等预处理。示例：generateStreamRag()使用 RetrievalAugmentationAdvisor集成检索与生成。

​​题目生成​​（AiServiceStudentimpl）：基于历史练习记录和知识点库，动态生成适配学生水平的题目。

​​④数据分析与报告生成​​

​​教学效率分析​​（AISmartAnalysisService）：基于备课/批改耗时数据生成优化建议报告（JSON 格式）。

​​学情报告​​（LearningAnalysisService）：分析学生答题数据，生成知识点掌握率、错误分布等可视化报告。

#### 3.1.2.需采用本地知识库作为输入，知识库资料总量不大于100M（本地知识库构建说明）

本地知识库实现架构如图 14本地知识库实现架构所示。

##### 3.1.2.1轻量化嵌入模型选择

OllamaEmbeddingModel.builder()

.ollamaApi(ollamaApi)

.defaultOptions(OllamaOptions.builder().model("nomic-embed-text").build())

技术实现：采用开源的nomic-embed-text轻量级嵌入模型

空间优化：相比百亿级大模型（通常>10GB），该模型体积缩小100倍以上

##### 3.1.2.2动态语义分块压缩

TokenTextSplitter tokenTextSplitter = new TokenTextSplitter();

List<Document> documentSplitterList = tokenTextSplitter.apply(documents);

按语义边界而非固定长度分块（如每块512 tokens）；避免传统分块导致的冗余存储（如标题重复存储）拓展点：可支持中英文混合场景

##### 3.1.2.3向量存储空间优化

使用PgVector的vector存储嵌入向量；相比文本存储节省50%空间（二进制编码对比UTF-8编码）支持压缩索引。

##### 3.1.2.4容量实时监控系统

@Scheduled(fixedRate = 60000) // 每分钟容量检查

public void checkCapacity() {

double currentSize = getCurrentSizeMB();

if (currentSize > maxSizeMB) triggerAlert(...);

}

控制机制：实时计算库大小：pg\_total\_relation\_size()；双阈值预警（默认warning=80M, critical=95M）；Redis分布式告警通道。

##### 3.1.2.5数据准入控制流程

// 文档解析时执行预压缩

TikaDocumentReader documentReader = new TikaDocumentReader(file.getResource());

预处理优化：Apache Tika解析时移除格式冗余（如PDF页眉/页脚）；本规范化（全角转半角、合并连续空格）；过滤非文本元素（图片/表格元数据）

##### 3.1.2.6双库隔离设计

@Bean("pgVectorStore") // 教学知识库

@Bean("pgVectorStoreQuestion") // 学生问题库

空间隔离：核心知识库独立存储（不含问答数据）；按学科标签分区存储（元数据过滤）

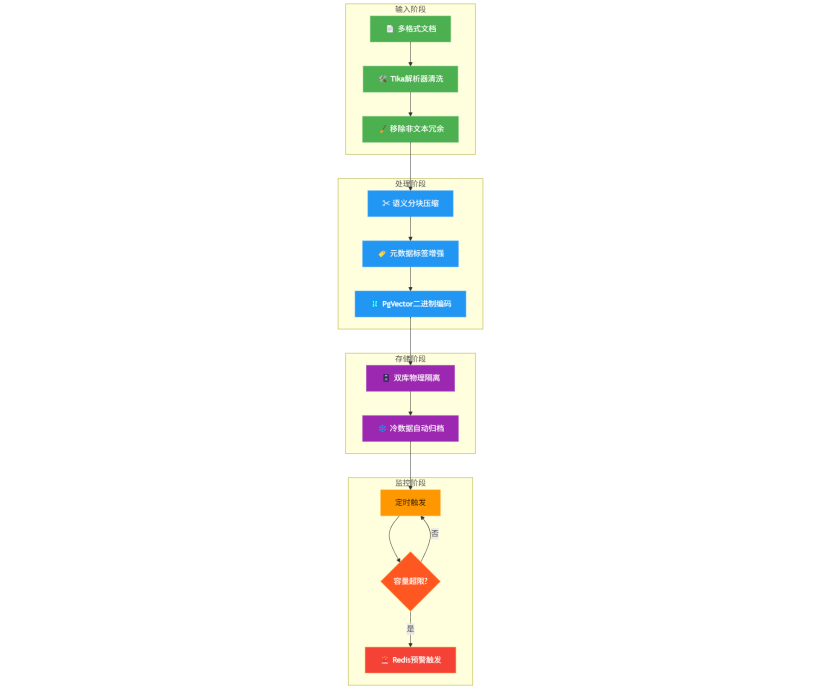


图 14本地知识库实现架构

该方案通过模型轻量化（Ollama）、存储优化（PgVector）、动态分块（TokenTextSplitter）三重技术保障，在保持语义完整性的前提下，将知识库严格控制在100M以内。

#### 3.1.3. 生成的内容、练习与答案与本地知识库的关联性和准确性说明

##### 3.1.3.1RAG（检索增强生成）架构流程保证关联性和准确性

​​核心流程​​如下：

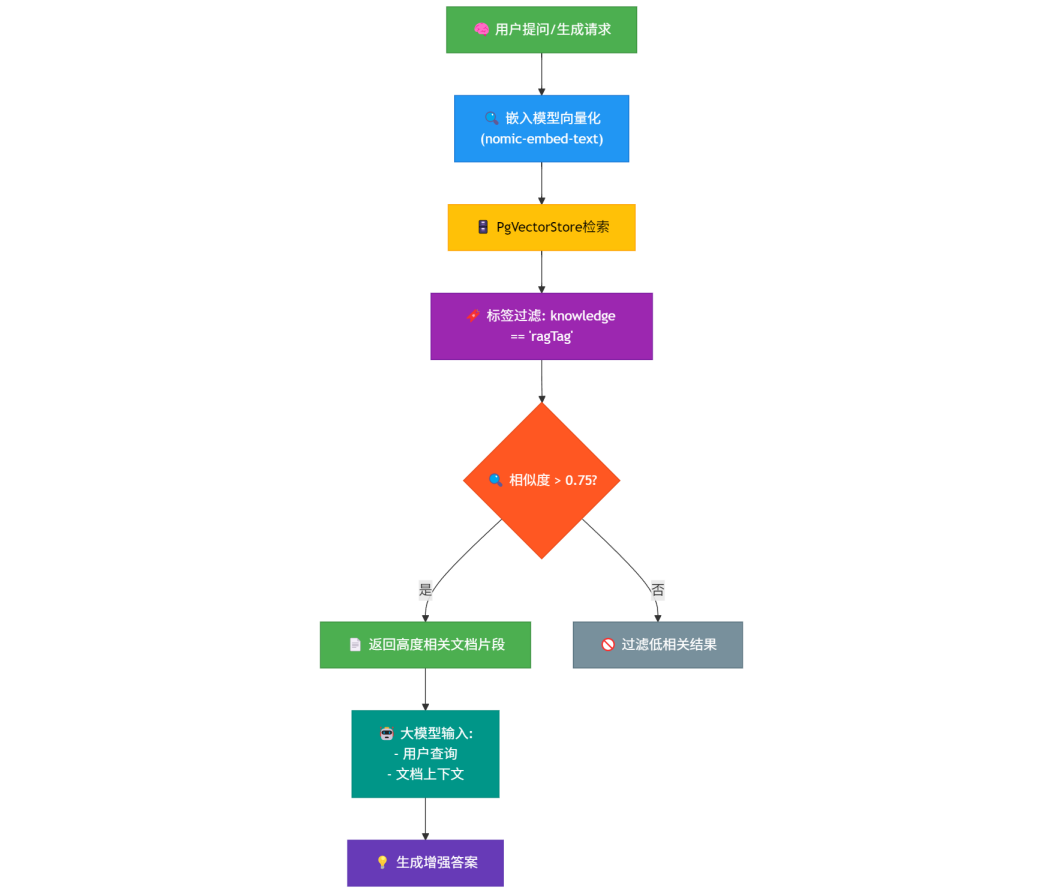
​​ 

图 15 RAG（检索增强生成）架构流程

当用户提问或请求生成内容（如课件、题目）时，系统首先从知识库中检索与输入相关的文档片段。

检索步骤：

使用嵌入模型（如nomic-embed-text）将用户查询向量化。

在向量数据库（PgVectorStore）中执行相似度搜索，并过滤指定标签（filterExpression("knowledge == '" + ragTag + "'")）。

设置相似度阈值（如0.75），仅返回高度相关的文档。

将检索到的文档片段（作为上下文）与用户查询一起输入大模型，要求模型基于这些文档生成答案。

核心代码实现如下：

//AiServiceImpl

public Flux<ChatResponse> generateStreamRag(String ragTag, String message) {

// 1. 检索：从知识库获取相关文档

List<Message> messages = ListMessage(message, ragTag);

// 2. 生成：将检索结果注入提示词

return studentChatModel.stream(new Prompt(messages));

}

​​检索阶段​​：使用向量数据库（PgVectorStore）按知识库标签（ragTag）检索相关文档：

SearchRequest request = SearchRequest.builder()

.query(message)

.filterExpression("knowledge == '" + ragTag + "'") // 按标签过滤

.build();

​​生成阶段​​：将检索结果注入系统提示词，强制模型基于知识库内容生成：

String SYSTEM\_PROMPT = "Use the information from DOCUMENTS: {documents}";

知识库关联性和准确性保障​​



##### 3.1.3.2提示词工程（精准内容生成控制）

结构化提示词约束​​：

// OllamaServiceTeacherImpl

String SYSTEM\_PROMPT\_TEST = """

题目必须融合三类源信息：

1. 知识库内容：{ragDocumentsText}

2. 用户要求：{assessmentName}

输出格式（Markdown）：

一、选择题...

答案：B ◆ // ◆标注正确答案

知识点：[知识点]

""";

​​强制内容来源​​：明确要求融合知识库内容（ragDocumentsText）

​​格式约束​​：通过Markdown模板控制输出结构，确保答案位置标准化（如◆标注正确答案）。

动态内容锚定​​：

//AiServiceStudentimpl

String SYSTEM\_PROMPT = """

题目生成要求：

1. 所有题目必须严格基于DOCUMENTS内容生成

2. 标注出处：<基于文档X>

DOCUMENTS: {documents}

HISTORY: {history} // 学生历史练习记录

""";

​​出处追踪​​：要求模型标注题目来源（<基于文档X>），实现内容可追溯。

​​历史记录融合​​：结合学生历史练习数据（HISTORY），避免重复生成已掌握知识点。

##### ​​3.1.3.3准确性增强机制

​​预处理优化​​



​​后处理校验​​

// postProcess

public List<Document> process(Query query, List<Document> documents) {

// 1. 去重

List<Document> uniqueDocs = documents.stream().distinct().collect(Collectors.toList());

// 2. 按关键词匹配数排序

return uniqueDocs.stream()

.sorted((d1, d2) -> countKeywordMatches(d2, query) - countKeywordMatches(d1, query))

.collect(Collectors.toList());

}

​​去重过滤​​：移除重复文档片段。

​​相关性重排序​​：按查询关键词匹配数量对文档重新排序，确保最相关内容优先。

##### ​​3.1.3.4知识库管理闭环​​

知识库构建规范

// RagServiceImpl

public Response<String> uploadFile(String ragTag, List<MultipartFile> files) {

documents.forEach(doc -> doc.getMetadata().put("knowledge", ragTag)); // 打标签

pgVectorStore.accept(documentSplitterList); // 存储到向量数据库

redissonClient.getList("ragTag").add(ragTag); // 更新标签列表

}

​​标签化存储​​：每个文档片段绑定知识库标签（如Java基础），确保检索范围精确。

​​统一维护​​：通过Redis管理标签列表，避免无效标签查询。

内容更新联动​​，当知识库新增文档时：

文本分割器（TokenTextSplitter）将文档拆解为语义片段

嵌入模型（OllamaEmbeddingModel）生成向量

存储到PgVectorStore，标签与原有知识库一致

##### ​​3.1.3.5特殊场景处理​​

编程题验证

// AnswerCorrectionService

public CorrectionResult correctAnswer(String question, String reference, String studentAnswer) {

String prompt = """

对比参考答案：{reference}

和学生答案：{studentAnswer}

返回JSON格式：{correct:boolean, errors:[{location, errorType, suggestion}]}

""";

// 调用模型校验

return parseCorrectionResponse(codeTeacherChatModel.call(prompt));

}

​​参考答案锚定​​：强制模型以知识库中的参考答案为基准进行比对。

​​结构化输出​​：通过JSON Schema约束输出，确保错误定位精准（如代码行号）。

开放性题目处理

// OllamaServiceTeacherImpl

String SYSTEM\_PROMPT\_LESSON = """

若知识库存在缺口 → 标注“建议补充知识点：XXX”

若用户需求与规范冲突 → 标注“已按规范调整：XXX”

""";

​​知识缺口标注​​：当知识库内容不足时，明确提示需补充的知识点，而非编造内容。

​​冲突解决机制​​：优先遵守预设规范（如课件格式），并说明调整逻辑。

##### ​​3.1.3.6小结与设计原则说明



设计原则说明：

所有生成内容必须可追溯至知识库（通过标签过滤、来源标注），并通过预处理-检索-生成-后处理的闭环流程，确保内容既符合知识库事实，又满足用户个性化需求。

### 3.2 基础功能满足说明

#### 3.2.1学生端

##### 3.2.1.1对学生的提出的问题，结合教学内容进行解答功能实现说明

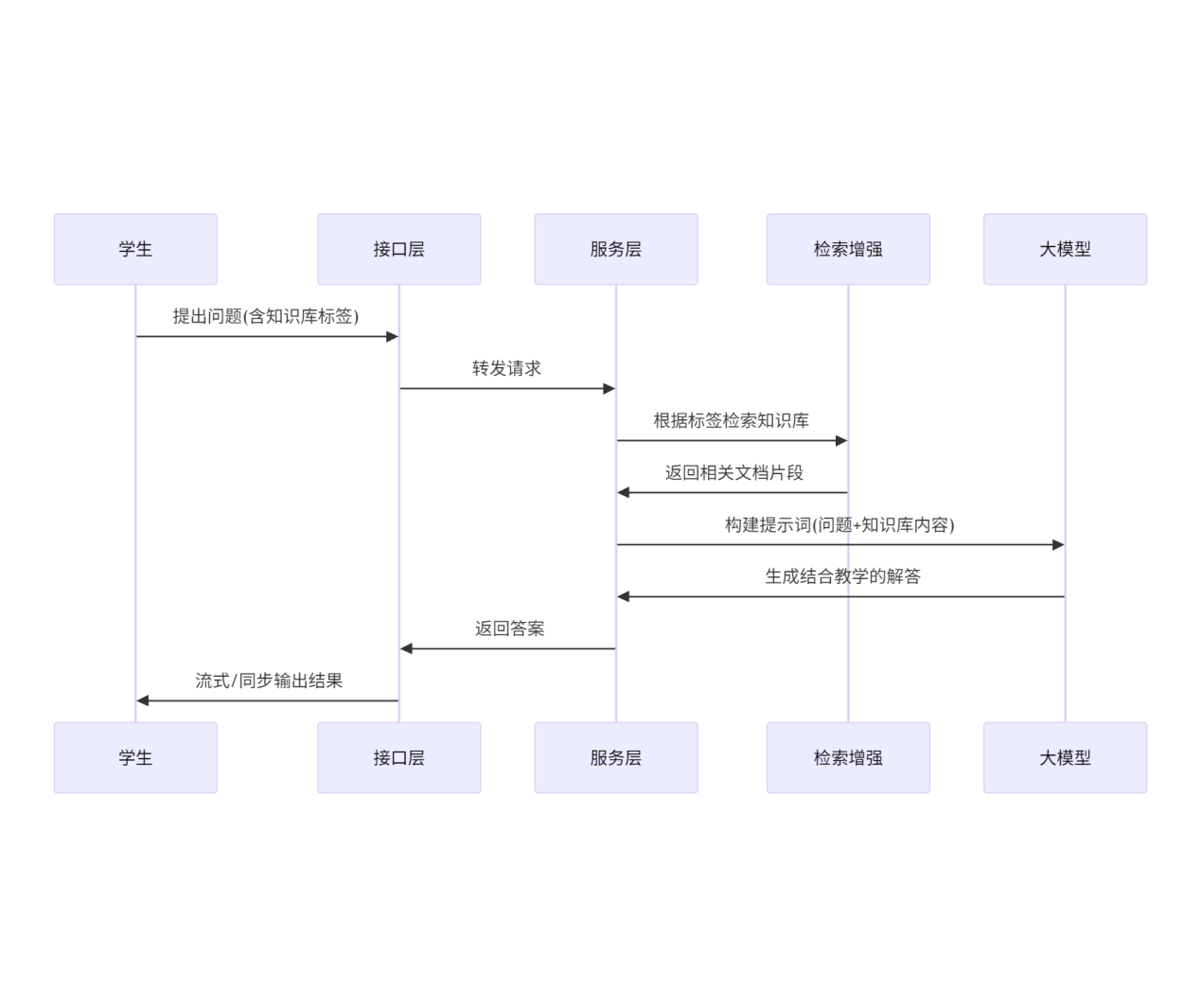


图 16在线学习助手流程架构

1、​​问题接收与路由​​

// OllamaController

@RequestMapping(value = "/generate\_stream\_rag", method = RequestMethod.GET)

public Flux<ServerSentEvent<ChatResponse>> generateStreamRag(

@RequestParam("ragTag") String ragTag, // 知识库标签

@RequestParam("message") String question // 学生问题

) {

return aiService.generateStreamRag(ragTag, question)

.map(response -> ServerSentEvent.builder(response).build());

}

​​知识库标签​​：强制学生指定问题关联的知识库

​​流式输出​​：通过Flux实现答案的逐步返回

2、​​知识检索增强（RAG）​​

// AiServiceImpl

public Flux<ChatResponse> generateStreamRag(String ragTag, String message) {

// 1. 知识检索

List<Message> messages = ListMessage(message, ragTag);

// 2. 模型生成

return studentChatModel.stream(new Prompt(messages));

}

private List<Message> ListMessage(String message, String ragTag) {

// 构建带标签过滤的检索请求

SearchRequest request = SearchRequest.builder()

.query(message)

.filterExpression("knowledge == '" + ragTag + "'")

.build();

// 执行检索

List<Document> documents = retriever.retrieve(new Query(request.getQuery()));

// 合并文档内容

String docText = documents.stream()

.map(Document::getText)

.collect(Collectors.joining());

// 构造提示词

Message ragMessage = new SystemPromptTemplate(SYSTEM\_PROMPT)

.createMessage(Map.of("documents", docText));

return List.of(new UserMessage(message), ragMessage);

}

3、​​精准提示词工程​​

// SYSTEM\_PROMPT

String SYSTEM\_PROMPT = """

Use the information from the DOCUMENTS section to provide accurate answers.

If unsure, state "I don't know".

Reply in Chinese!

DOCUMENTS:

{documents}

""";

​​双重约束​​：强制使用DOCUMENTS中的知识；限定中文输出和固定格式。

​​防幻觉机制​​：明确要求对不确定内容回答"不知道"。

4、​​查询优化处理​​

//generateStreamRagAdvisor

public Flux<ChatResponse> generateStreamRagAdvisor(String message, String ragTag) {

// 1. 查询翻译

TranslationQueryTransformer translator = ...

Query translated = translator.transform(new Query(message));

// 2. 查询重写

RewriteQueryTransformer rewriter = ...

Query rewritten = rewriter.transform(translated);

// 3. 多重扩展

MultiQueryExpander expander = ...

List<Query> expandedQueries = expander.expand(rewritten);

// 4. 检索并生成

return chatClient.prompt()

.advisors(new RetrievalAugmentationAdvisor()

.withQueryTransformers(translator, rewriter)

.withDocumentRetriever(retriever))

.messages(new UserMessage(message))

.stream();

}

5、教学内容结合机制​​

​​知识点锚定​​

//AiServiceStudentimpl

public Flux<ChatResponse> generateRagQuestion(String ragTag, String message) {

// 提取知识点标签

String knowledgePoint = getKnowlagePoint(message);

// 获取相关历史练习

List<LearningRecord> records = learningRecordService

.findSimilarKnowledgeWithThreshold(knowledgePoint, 0.5f, 5);

// 构建含历史数据的提示词

Message prompt = new SystemPromptTemplate(SYSTEM\_PROMPT)

.createMessage(Map.of("documents", docText, "history", history));

}

教学场景适配​​

// OllamaServiceTeacherImpl

String SYSTEM\_PROMPT = """

根据教学场景调整回答深度：

- 初学者：用简单类比解释概念

- 进阶者：提供代码示例

- 高级者：讨论设计模式应用

当前场景：{userLevel}

""";

多模态支持​​

// CheckAnswerServiceImpl

TikaDocumentReader documentReader = new TikaDocumentReader(studentAnswer.getResource());

List<Document> documents = documentReader.get(); // 解析PDF/Word等格式

String content = documents.stream().map(Document::getText).collect(joining());

​​支持格式​​：代码文件、PDF作业、手写图片（需OCR）

​​统一处理​​：转换为文本后注入提示词

6、保障机制总结



##### 3.2.1.2  根据学生历史练习情况，以及学生的练习要求，生成随练题目，并对练习纠错功能实现说明

###### 3.2.1.2.1个性化题目生成机制​​

历史练习数据分析​​

// LearningRecordService

public List<LearningRecord> findSimilarKnowledgeWithThreshold(

String keyword, float minSimilarity, int limit) {

// 1. 根据知识点关键词检索历史记录

List<LearningRecord> results = learningRecordMapper

.findSimilarKnowledge(keyword, limit \* 2);

// 2. 过滤低相似度记录

return results.stream()

.filter(record -> record.getSimilarityScore() >= minSimilarity)

.limit(limit)

.collect(Collectors.toList());

}

​​核心参数​​：keyword：当前知识点；minSimilarity：相似度阈值（过滤无关历史记录）；limit：最多返回记录数

​​题目生成流程​​

//AiServiceStudentimpl

public Flux<ChatResponse> generateRagQuestion(String ragTag, String message) {

// 1. 解析知识点和题目要求

String knowledgePoint = getKnowlagePoint(message);

String questionReq = getQuestion(message);

// 2. 获取历史练习数据

List<LearningRecord> records = learningRecordService

.findSimilarKnowledgeWithThreshold(knowledgePoint, 0.5f, 5);

// 3. 构建历史数据字符串

StringBuilder history = new StringBuilder();

for (LearningRecord record : records) {

history.append("题目:")

.append(record.getQuestion())

.append(" 学生回答:")

.append(record.getUserAnswer())

.append(" 得分:")

.append(record.getScore());

}

// 4. 检索知识库文档

List<Document> ragDocs = searchService.ListDocument(knowledgePoint, ragTag);

String ragText = ragDocs.stream().map(Document::getText).collect(joining());

// 5. 构建提示词

Message prompt = new SystemPromptTemplate(SYSTEM\_PROMPT)

.createMessage(Map.of(

"documents", ragText,

"history", history.toString(),

"question", questionReq

));

// 6. 调用模型生成题目

return ollamaChatModel.stream(new Prompt(prompt));

}

智能题目生成策略​​

// SYSTEM\_PROMPT

String SYSTEM\_PROMPT = """

根据历史练习记录(HISTORY)和知识库(DOCUMENTS)生成题目：

1. 选题逻辑：

- 优先选择学生错误率最高的知识点

- 难度分级：

\* 正确率<50% → 基础题

\* 50%-80% → 进阶题

\* >80% → 综合应用题

2. 避免重复近期题目形式

3. 输出格式：MARKDOWN

示例：题目：用Python实现快速排序

DOCUMENTS: {documents}

HISTORY: {history}

""";

历史数据驱动决策​​



###### 3.2.1.2.2智能纠错机制​​

​​纠错流程​​

// AnswerCorrectionService

public CorrectionResult correctAnswer(String question, String reference, String studentAnswer) {

// 1. 构建提示词

String prompt = """

对比参考答案：{reference}

和学生答案：{studentAnswer}

返回JSON：{correct:boolean, errors:[{location,errorType,suggestion}]}

""";

// 2. 调用专用模型针对代码评价优化

String response = codeTeacherChatModel.call(prompt);

// 3. 解析为结构化结果

return parseCorrectionResponse(response);

}

纠错策略​​

// SYSTEM\_PROMPT

String prompt = """

你是一个专业教师，请对比学生答案和参考答案：

问题：%s

参考答案：%s

学生答案：%s

要求：

1. 错误类型包括：

- 概念错误

- 计算错误

- 逻辑错误

- 表述不清

2. 错误定位精确到行号/段落

3. 输出JSON格式

""";

历史感知纠错​​

// generateRagJudgement

public Flux<ChatResponse> generateRagJudgement(String question, String answer) {

// 1. 获取历史纠错记录

List<LearningRecord> records = learningRecordService

.findSimilarKnowledgeWithThreshold(question, 0.7f, 3);

// 2. 构建历史上下文

String history = records.stream()

.map(r -> "问题:" + r.getQuestion() + " 错误:" + r.getCorrection().getErrors())

.collect(joining("\n"));

// 3. 构建提示词

Message prompt = new SystemPromptTemplate(SYSTEM\_PROMPT)

.createMessage(Map.of(

"question", question,

"answer", answer,

"history", history

));

// 4. 生成纠错建议

return ollamaChatModel.stream(new Prompt(prompt));

}

多维度评分体系​​

//SYSTEM\_PROMPT\_Judgment

String SYSTEM\_PROMPT\_Judgment = """

【结果判定】：正确/错误

【评分】：

- 正确：85-100分（按完整度）

- 错误：60-79分（按错误程度）

【评价】：

- 正确：说明亮点

- 错误：

a) 指出具体错误点

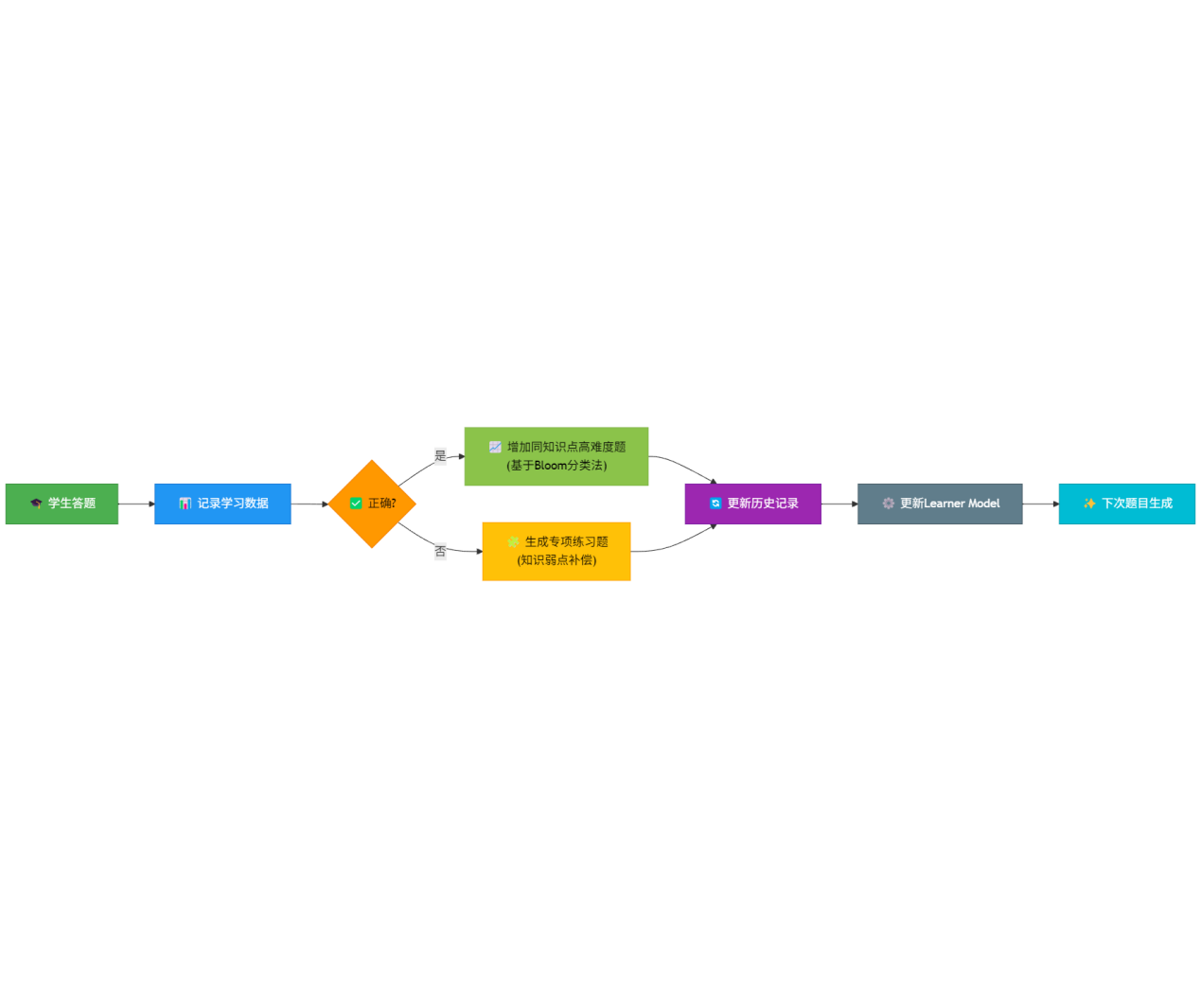
b) 1句话解释正确知识

c) 鼓励建议（如"建议复习XX"）

""";

###### 3.2.1.2.3闭环学习系统​​

​​数据闭环​​

历史记录结构​​

//：LearningRecord

public class LearningRecord {

private Long id;

private String questionId; // 题目ID

private String studentId; // 学生ID

private String knowledge; // 知识点

private String question; // 问题内容

private String userAnswer; // 学生答案

private CorrectionResult correction; // 批改结果(含错误详情)

private Double score; // 得分

}

###### 3.2.1.2.4技术亮点​​

​​历史感知生成​​通过findSimilarKnowledgeWithThreshold精准定位薄弱点，避免"无的放矢"

​​渐进式难度调整​​基于历史正确率动态调整题目难度（基础→进阶→综合）

​​纠错溯源能力​​错误定位精确到代码行/文本段落，直接关联知识库文档

​​防刷题机制​​避免重复近期题目形式防止机械刷题

​​闭环优化​​每次练习更新历史记录，驱动后续题目个性化生成

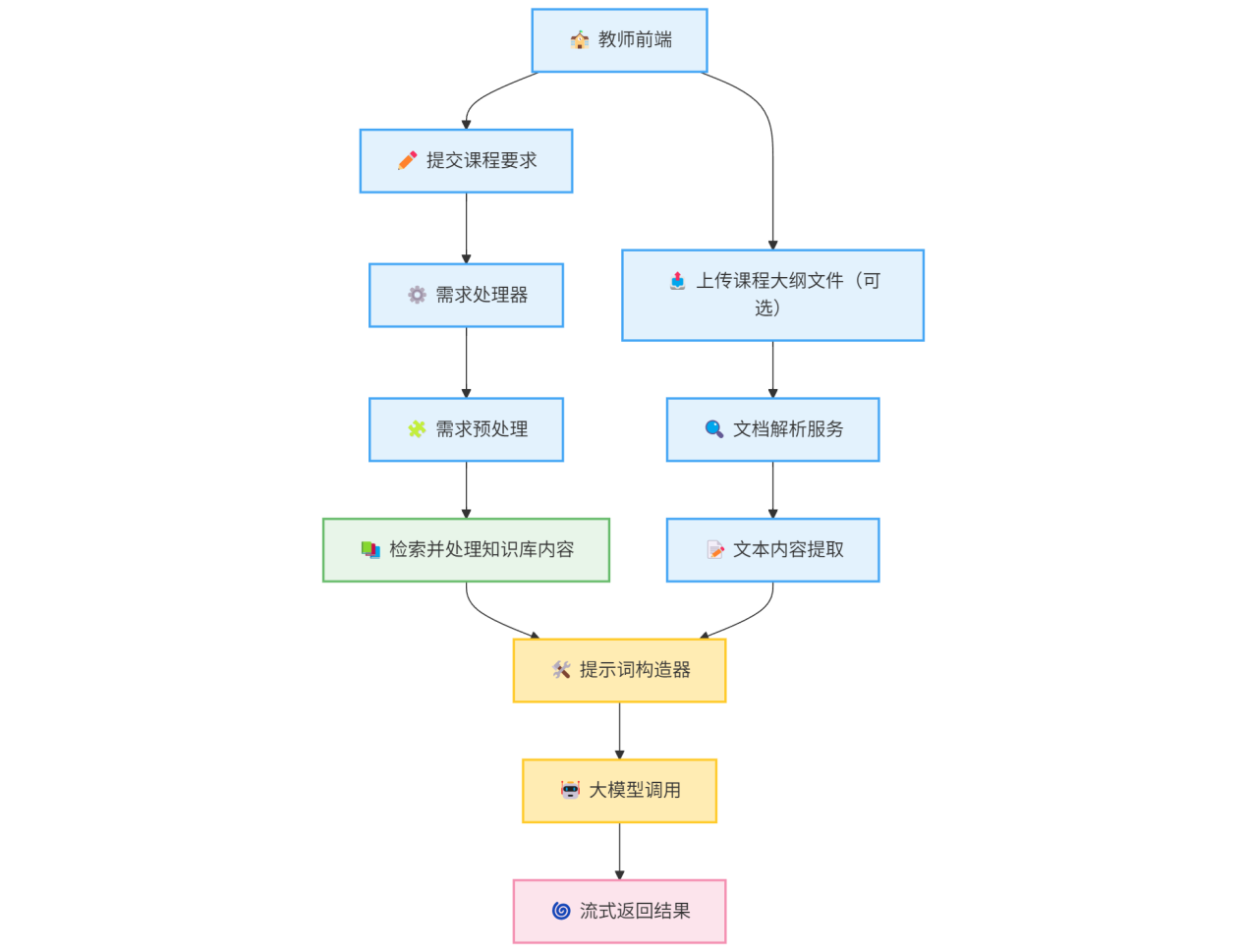
系统将历史练习数据作为核心燃料，通过题目生成→练习→纠错→报告的闭环流程，实现"越练越准"的自适应学习体验。每次练习都精准打击薄弱点，避免无效重复。

#### 3.2.2教师端

##### 3.2.2.1备课与设计

（根据所提供的本地课程大纲、课程知识库文档等自动设计教学内容，包括知识讲解、实训练习与指导、时间分布）

###### 3.2.2.1.1实现流程



拓展文件上传接口，实现教师与系统的多轮有效交互（如教师不满意生成的课件可下载生成的课件自行修改后上传系统，经过处理后作为上下文注入模型或直接在教学目标处说明修改的方向再次生成课件，直到满意为止）



###### 3.2.2.1.2接口触发​​

教师端调用 OllamaServiceTeacherImpl.lessonCreatStream()方法

传入参数：课程要求（LessonRequest）和课程大纲文件（MultipartFile）

###### 3.2.2.1.3文档解析​​

// 解析课程文件

TikaDocumentReader documentReader = new TikaDocumentReader(syllabusFile.getResource());

List<Document> documents = documentReader.get();

String documentCollectors = documents.stream().map(Document::getText).collect(Collectors.joining());

###### 3.2.2.1.4知识库检索​​

// 根据学科和知识库标签检索文档

List<Document> ragDocuments = searchService.ListDocument(

lessonRequest.getSubject(),

lessonRequest.getKnowledgeBase()

);

String ragDocumentsText = ragDocuments.stream().map(Document::getText).collect(Collectors.joining());

###### 3.2.2.1.5提示词工程​​使用多源输入的复杂模板构建系统提示词

String SYSTEM\_PROMPT\_LESSON = ... //详细规则见下文

Message lessonMessage = new SystemPromptTemplate(SYSTEM\_PROMPT\_LESSON)

.createMessage(Map.of(

"documentCollectors", documentCollectors, // 大纲内容

"userQuestion", userQuestion, // 教师需求

"coursewareRequirements", coursewareRequirements, // 课时/难度等

"ragDocumentsText", ragDocumentsText // 知识库内容

));

###### 3.2.2.1.6大模型生成​​

// 使用32B大模型生成内容

return teacherChatModel.stream(new Prompt(lessonMessage));

核心提示词设计（SYSTEM\_PROMPT\_LESSON）如下：

1. \*\*教学目标\*\*

- 结合知识库和课程大纲

- 呼应教师个性化需求

2. \*\*知识讲解\*\*

- 按大纲逻辑组织知识点

- 标注重点(★)/难点(▲)

- 融入案例/演示等需求

3. \*\*实训指导\*\*

- 设计基于知识库的实操任务

- 提供参考答案及错误提示

- 包含分步指导

4. \*\*时间规划\*\*

- 按总课时分配模块时间

- 重点/难点增加时间配比

- 标注各环节时长（如"★重点知识：15分钟"）

5. \*\*补充说明\*\*

- 知识缺口标注建议

- 规范冲突时的调整说明

###### 3.2.2.1.7关键技术点

​​多源输入融合​​整合四种输入源：课程大纲文件 + 教师需求 + 知识库文档 + 教学规范要求

​​约束驱动生成​​通过提示词实现强约束：

所有知识必须源自知识库

严格遵循格式规范

明确标注个性化需求响应

​​大模型选型​​使用 deepseek-r1:32b模型确保复杂逻辑处理能力：

@Qualifier("customTeacherChatModel") // 配置的32B大模型

OllamaChatModel teacherChatModel;

​​实时流式输出​​采用响应式流返回结果：

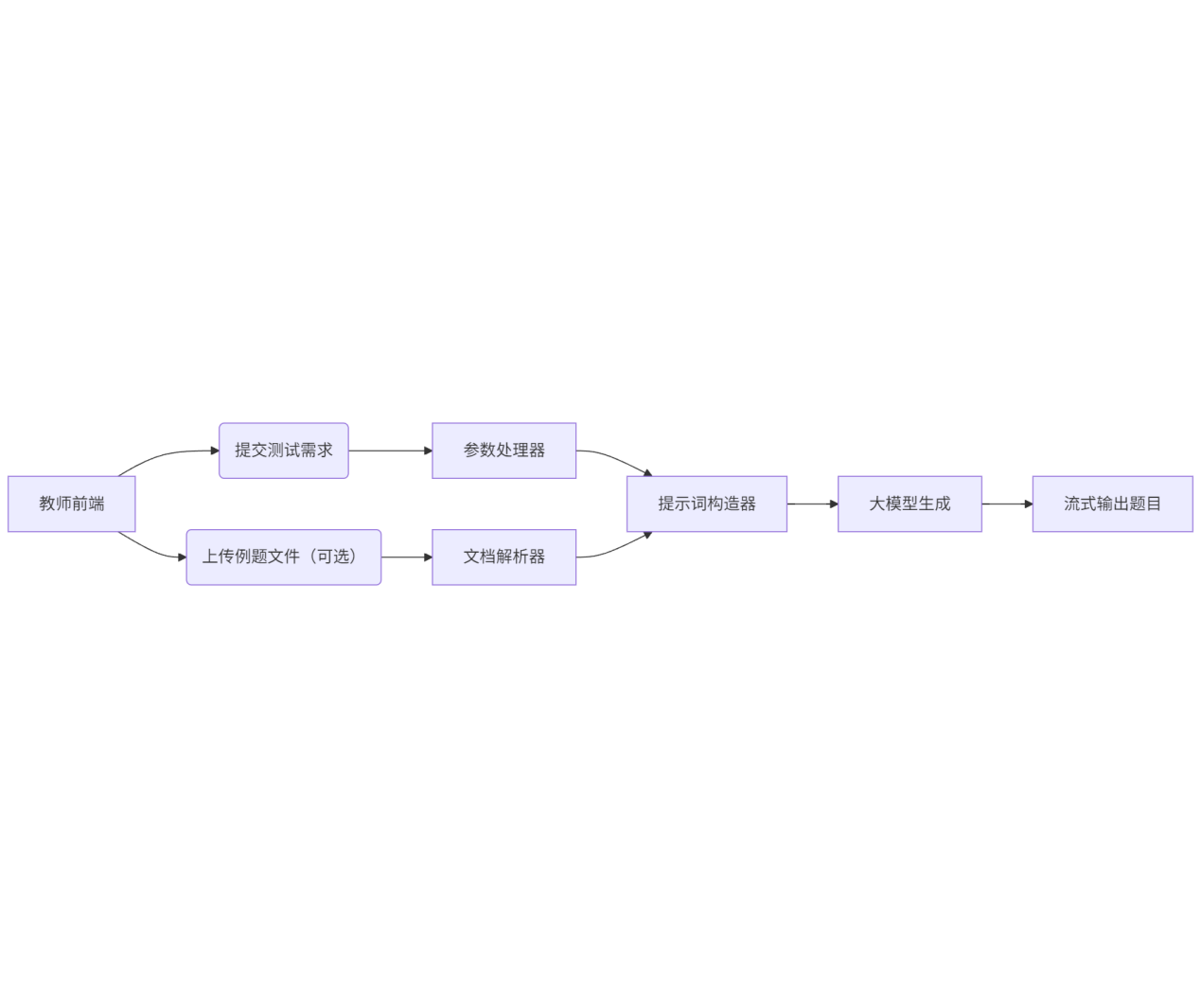
return teacherChatModel.stream(new Prompt(lessonMessage));

该实现通过严格的提示词工程和多源数据整合，确保生成的课件既符合教学规范要求，又能满足教师的个性化需求，同时保持所有知识内容的准确性和可追溯性。

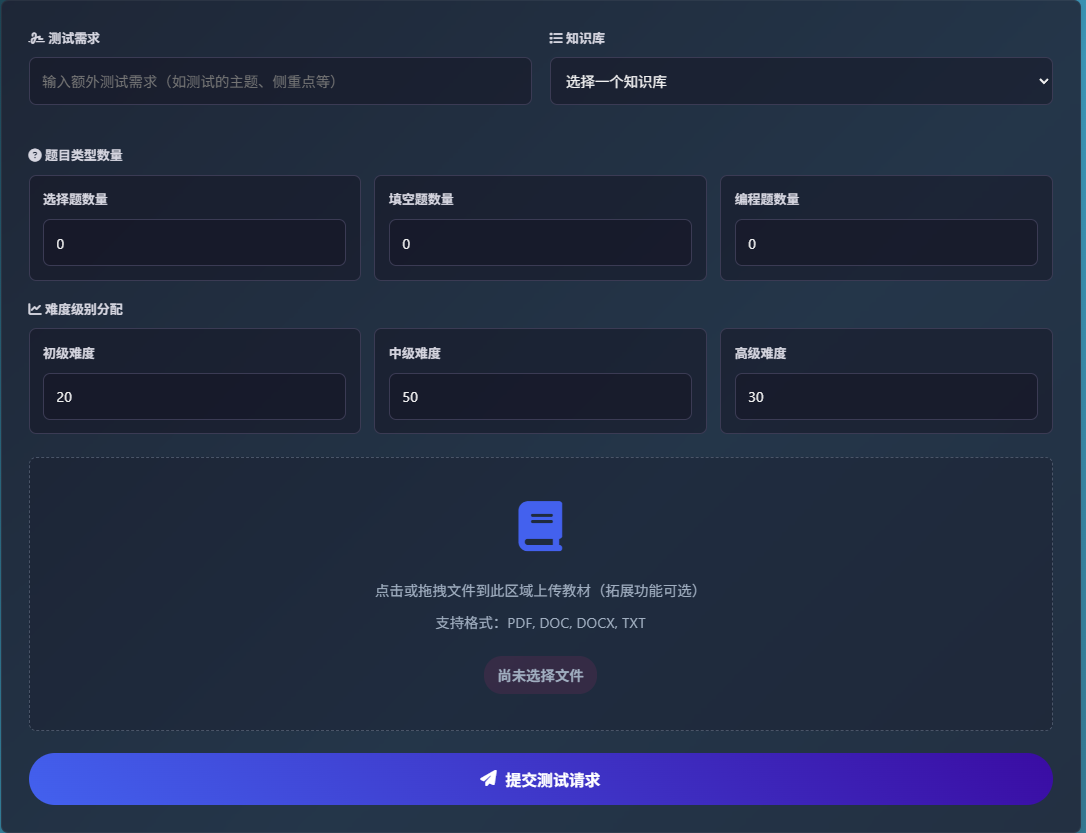
##### 3.2.2.2考核内容生成

（根据教学内容自动生成考核题目及参考答案，考核题目种类可多样化，根据学科设计，如计算机类可设计相关编程题和答案）

###### 3.2.2.2.1实现流程



拓展文件上传接口，实现教师与系统的多轮有效交互（如教师不满意生成的试题可下载生成的试题自行修改后上传系统再次生成或者在测试需求部分说明需求方向再次生成，直到满意为止）



###### 3.2.2.2.2接口触发​​

教师端调用lessonCreatStreamTest()方法

传入参数：

TestRequest testRequest // 包含题目要求的结构体

MultipartFile syllabusFile // 包含往年例题的课程大纲文件（拓展功能，可选）

###### 3.2.2.2.3文档解析​​

// 解析上传文件

TikaDocumentReader documentReader = new TikaDocumentReader(syllabusFile.getResource());

List<Document> documents = documentReader.get();

String documentCollectors = documents.stream().map(Document::getText).collect(Collectors.joining());

###### 3.2.2.2.4需求参数提取​​

从TestRequest对象中提取：

String assessmentName = "用户需求：" + testRequest.getAssessmentName(); // 测试主题

String assessmentType = "知识库：" + testRequest.getAssessmentType(); // 知识点分类

String questionTypes = "【测试题型】选择题数量：" + testRequest.getMultipleChoiceCount() + ...; // 题型分布

String difficulty = "【测试难度】初级：" + testRequest.getBeginnerLevel() + ...; // 难度分布

###### 3.2.2.2.5知识库检索​​

List<Document> ragDocuments = searchService.ListDocument(

testRequest.getAssessmentName(),

testRequest.getAssessmentType()

);

String ragDocumentsText = ragDocuments.stream().map(Document::getText).collect(Collectors.joining());

###### 3.2.2.2.6提示词工程​​

使用多约束的提示词模板：

String SYSTEM\_PROMPT\_TEST = """

角色：高级试题命题专家

任务：基于输入源生成结构化试题

输入源：

1. 参考例题：{documentCollectors}

2. 知识库：{ragDocumentsText}

3. 用户要求：{assessmentName}

4. 题型：{questionTypes}

5. 难度：{difficulty}

核心规则：

1. 题目必须融合三类源信息

2. 难度比例误差 ≤ ±5%

3. 题型要求：

- 选择题：正确答案用 ◆ 标注

- 填空题：用「 」标注答案区

- 编程题：包含测试用例

4. 在题目后给出参考答案

输出格式示例（Markdown）：

一、选择题（每题[分值]分）

1. 题干...

A. 选项1

B. 选项2

...

答案：B ◆

知识点：[知识点]

...""";

###### 3.2.2.2.7大模型生成​​

Message testMessage = new SystemPromptTemplate(SYSTEM\_PROMPT\_TEST)

.createMessage(Map.of(

"ragDocumentsText", ragDocumentsText,

"assessmentName", assessmentName,

"questionTypes", questionTypes,

"difficulty", difficulty,

"documentCollectors", documentCollectors

));

return teacherChatModel.stream(new Prompt(testMessage)); // 流式输出

###### 3.2.2.2.8关键技术设计

题型规范约束,通过提示词强制规范输出格式：

"""

二、简答题（每题[分值]分）

1. 问题...

评分要点：

1) 要点1

2) 要点2

知识点：[知识点]

三、编程题（每题[分值]分）

1. 题目...

输入：[格式]

输出：[格式]

样例：

输入：[示例]

输出：[示例]

评分标准：

- 功能（X分）：...

- 代码（X分）：...

"""

难度控制机制

// 难度分布参数

String difficulty = "初级占比：" + testRequest.getBeginnerLevel() + "%" +

"中级占比：" + testRequest.getIntermediateLevel() + "%" +

"高级占比：" + testRequest.getAdvancedLevel() + "%";

// 在提示词中要求

"难度比例误差 ≤ ±5%"

参考答案生成,通过两种方式确保参考答案质量：自动标注​​,要求选择题用◆标注正确答案;结构化输出​​。

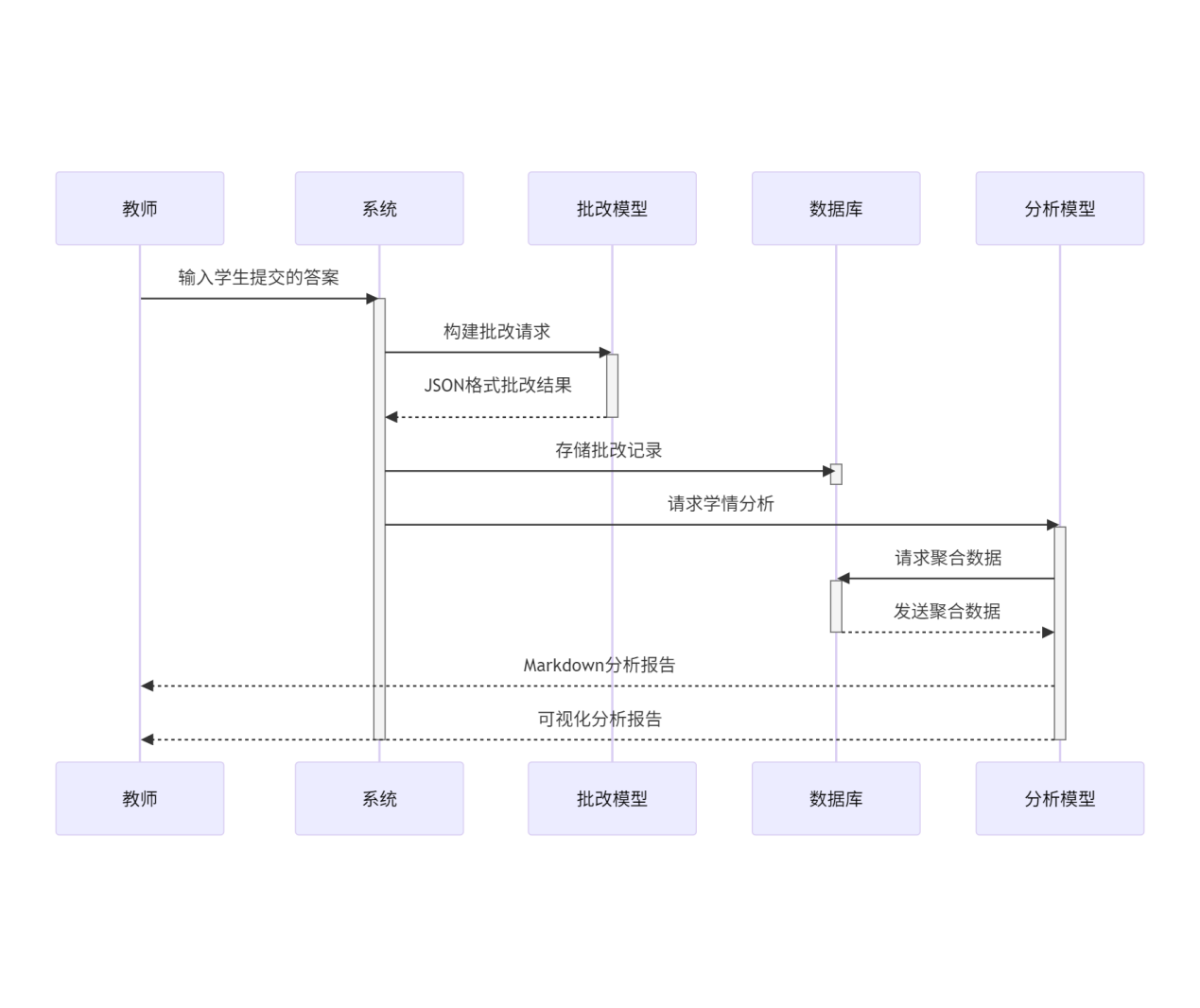
多源验证，强制融合二/三类信息源：往年例题（可选）（解析自文件）；知识库文档（向量检索）；教师定制要求（题型/难度）

该实现通过严格的提示词约束和多源数据融合，确保生成的考核题目既符合教学要求，又能适应不同难度层级，同时提供可直接使用的参考答案和评分标准。

##### 3.2.2.3学情数据分析

（对学生提交的答案进行自动化检测，提供错误定位与修正建议。对学生整体数据进行分析，总结知识掌握情况与教学建议。）

###### 3.2.2.3.1实现流程



###### 3.2.2.3.2答案批改与错误定位（AnswerCorrectionService类）

​**​**a. 提示词工程​​

String prompt = """

你是一个专业教师，请对比学生答案和参考答案：

问题：%s

参考答案：%s

学生答案：%s

要求：

1. 用JSON格式返回：{correct:boolean, errors:[{location:string, errorType:string, suggestion:string}]}

2. 错误定位精确到句子位置

3. 错误类型包括：概念错误、计算错误、逻辑错误、表述不清

4. 一般表述用中文

""".formatted(question, reference, studentAnswer);

​​b. JSON结果解析​​

private CorrectionResult parseCorrectionResponse(String json) {

// 自定义反序列化器处理复杂结构

SimpleModule module = new SimpleModule();

module.addDeserializer(CorrectionResult.class, new CorrectionResultDeserializer());

objectMapper.registerModule(module);

return objectMapper.readValue(json, CorrectionResult.class);

}

// 反序列化器实现

private static class CorrectionResultDeserializer extends StdDeserializer<CorrectionResult> {

public CorrectionResult deserialize(JsonParser p, DeserializationContext ctxt) {

JsonNode node = p.getCodec().readTree(p);

CorrectionResult result = new CorrectionResult();

result.setCorrect(node.get("correct").asBoolean());

// 解析错误详情

List<ErrorDetail> errors = new ArrayList<>();

Iterator<JsonNode> errorNodes = node.get("errors").elements();

while (errorNodes.hasNext()) {

JsonNode errorNode = errorNodes.next();

ErrorDetail error = new ErrorDetail();

error.setLocation(errorNode.get("location").asText());

error.setErrorType(errorNode.get("errorType").asText());

error.setSuggestion(errorNode.get("suggestion").asText());

errors.add(error);

}

result.setErrors(errors);

return result;

}

}

​​c. 数据结构​​

public class CorrectionResult {

private boolean correct;

private List<ErrorDetail> errors;

// getters & setters

}

public class ErrorDetail {

private String location; // 错误位置

private String errorType; // 错误类型

private String suggestion; // 修正建议

// getters & setters

}

d. 批改结果示例

{

"correct": false,

"errors": [

{

"location": "第3行",

"errorType": "概念错误",

"suggestion": "变量作用域理解错误，建议复习局部变量定义"

},

{

"location": "第7-8行",

"errorType": "逻辑错误",

"suggestion": "循环终止条件不正确，应改为i<=10"

}

]

}

###### 3.2.2.3.3整体学情分析与报告生成（LearningAnalysisService类）

​​a. 数据预处理​​

public String prepareAnalysisData(List<AnswerRecord> records) {

// 按问题分组的学生答题情况

Map<String, List<Map<String, Object>>> analysisData = records.stream()

.collect(Collectors.groupingBy(AnswerRecord::getQuestionId))

.entrySet().stream()

.collect(Collectors.toMap(...));

// 错误类型统计

Map<String, Long> errorDistribution = records.stream()

.filter(record -> record.getCorrection() != null)

.flatMap(record -> record.getCorrection().getErrors().stream())

.collect(Collectors.groupingBy(ErrorDetail::getErrorType, Collectors.counting()));

// 构造JSON数据

Map<String, Object> finalData = new HashMap<>();

finalData.put("questions", analysisData);

finalData.put("errorDistribution", errorDistribution);

finalData.put("totalRecords", records.size());

return objectMapper.writeValueAsString(finalData);

}

​​b. 提示词工程​​

String prompt = """

基于以下学生答题数据（JSON格式），生成学情分析报告：

%s

输出要求

1.输出格式:MARKDOWN

报告要求：

1. 知识掌握情况（按知识点分类统计用表格表示）

2. 常见错误类型分布（使用表格展示）

3. 针对性教学建议（分学生和班级两个维度）

""".formatted(analysisData);

​​c. 知识点掌握分析​​

public Map<String, Double> getKnowledgeMastery(String classId) {

List<AnswerRecord> records = recordRepository.findByClassId(classId);

return records.stream()

.collect(Collectors.groupingBy(

AnswerRecord::getQuestionId,

Collectors.collectingAndThen(

Collectors.toList(),

list -> {

long correctCount = list.stream()

.filter(r -> r.getCorrection() != null && r.getCorrection().isCorrect())

.count();

return (double) correctCount / list.size() \* 100;

}

));

}

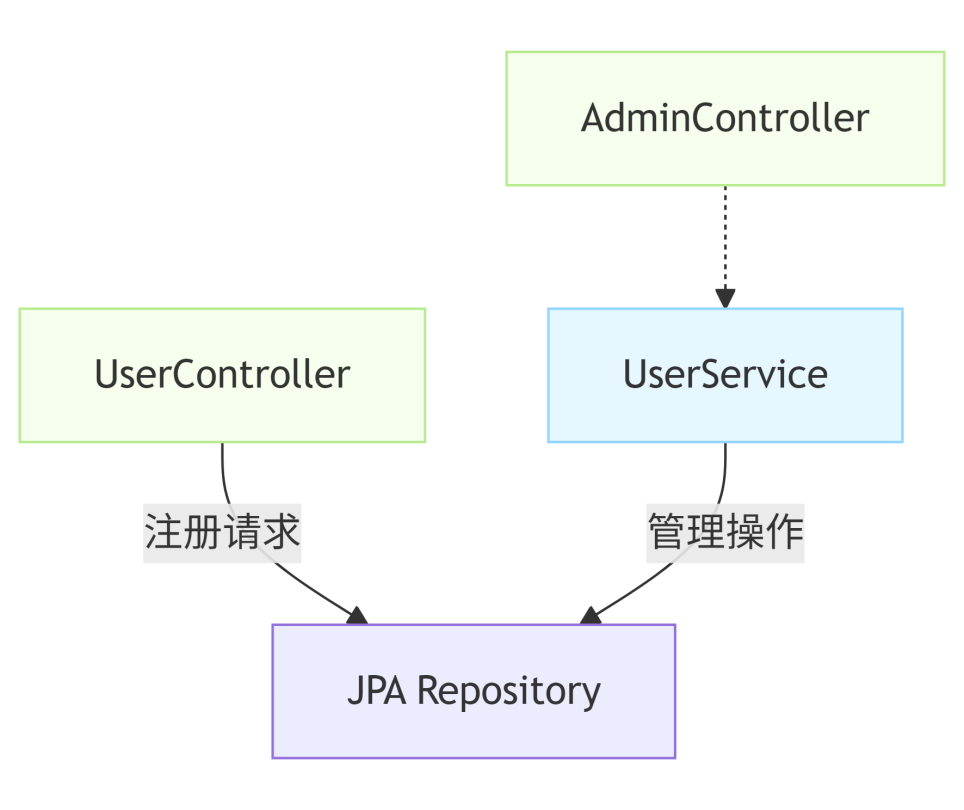
该实现通过结构化提示词工程、专门优化的JSON解析机制和多维度数据分析，实现了从答案的精细化批改到整体学情的深度分析全流程覆盖。

#### 3.2.3管理员端

##### 3.2.3.1 用户管理

（用户管理：管理员/教师/学生等用户的基本管理）

###### 3.2.3.1.1实现流程



###### 3.2.3.1.2​​用户实体与存储​​

​​用户实体类​​： User类表示用户对象

​​数据存储​​：使用Spring Data JPA的Repository接口

###### 3.2.3.1.3 ​​核心功能实现​​

​​注册功能​​（UserController类）：

@PostMapping("/register")

public Result register(@RequestBody myUsers user){

userRe.save(user); // 直接调用JPA保存

return Result.success();

}

​​管理员操作​​（UserService类）：

​​创建用户​​：检查用户名唯一性后保存

public User createUser(User user) {

if (userRepo.existsByUsername(user.getUsername())) {

throw new IllegalArgumentException("用户名已存在");

}

return userRepo.save(user);

}

​​用户管理​​：

分页查询：getAllUsers(Pageable pageable)

ID查询：getUserById(Long id)

更新信息：updateUser()包含用户名冲突校验

删除用户：deleteUser()先校验存在性

按角色过滤：getUsersByRole(String role)

###### 3.2.3.1.4 ​​安全设计​​

​​唯一性校验​​：注册和管理操作均检查用户名唯一性

​​数据隔离​​：管理员服务位于com.wxx.service.admin包，与普通用户功能分离

​​密码管理​​（预留了加密接口）：

user.setPassword(passwordEncoder.encode(user.getPassword()));

​​分层实现​​：Controller处理请求，Service处理业务逻辑

​​双重校验​​：注册和管理操作都有用户名唯一性检查

##### 3.2.3.2 课件资源管理

（课件资源管理：按学科列表教师备课产生的课件、练习等资源，可以导出。）

###### 3.2.3.2.1实现流程

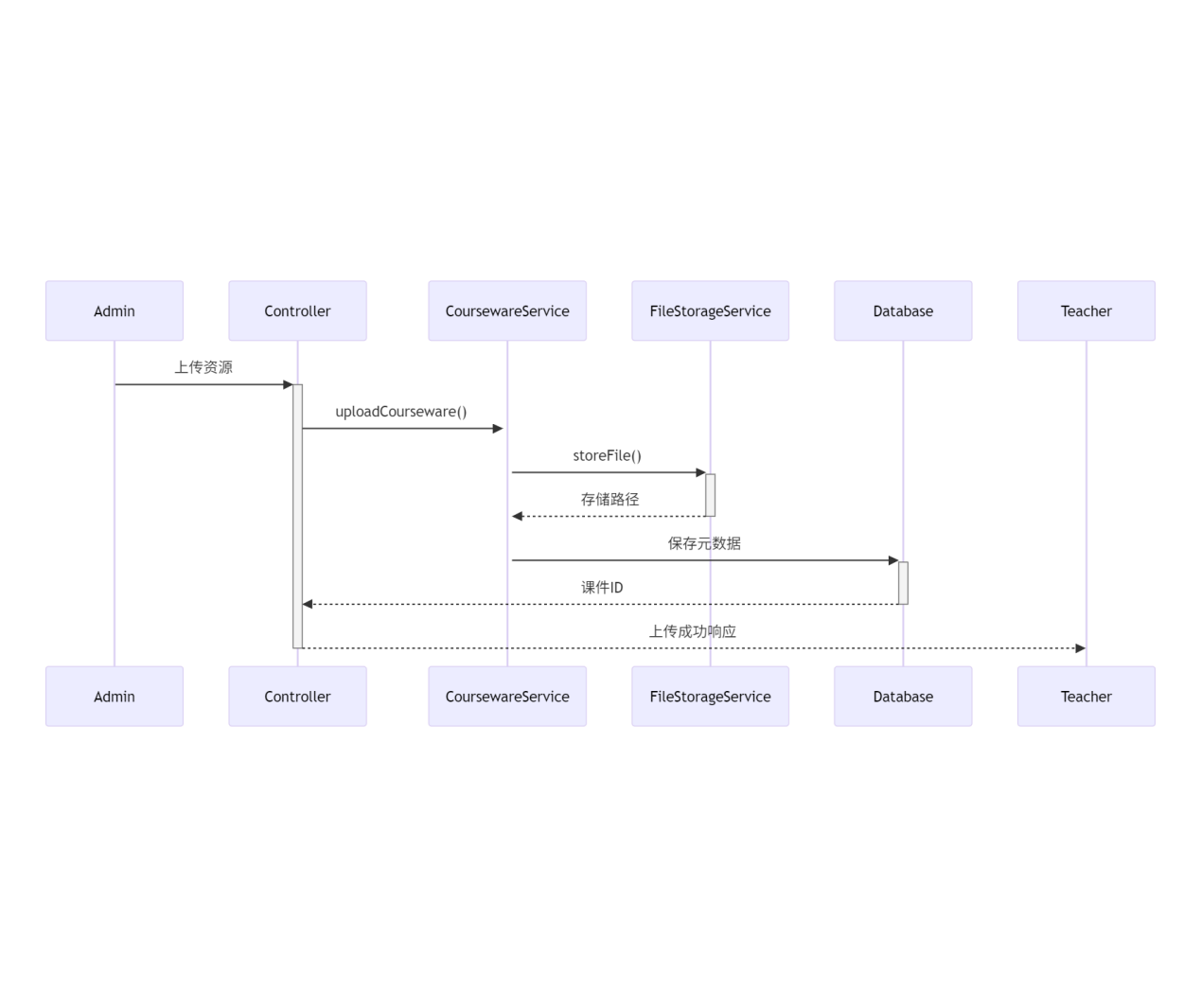


图 17管理员上传审核后课件资源流程图

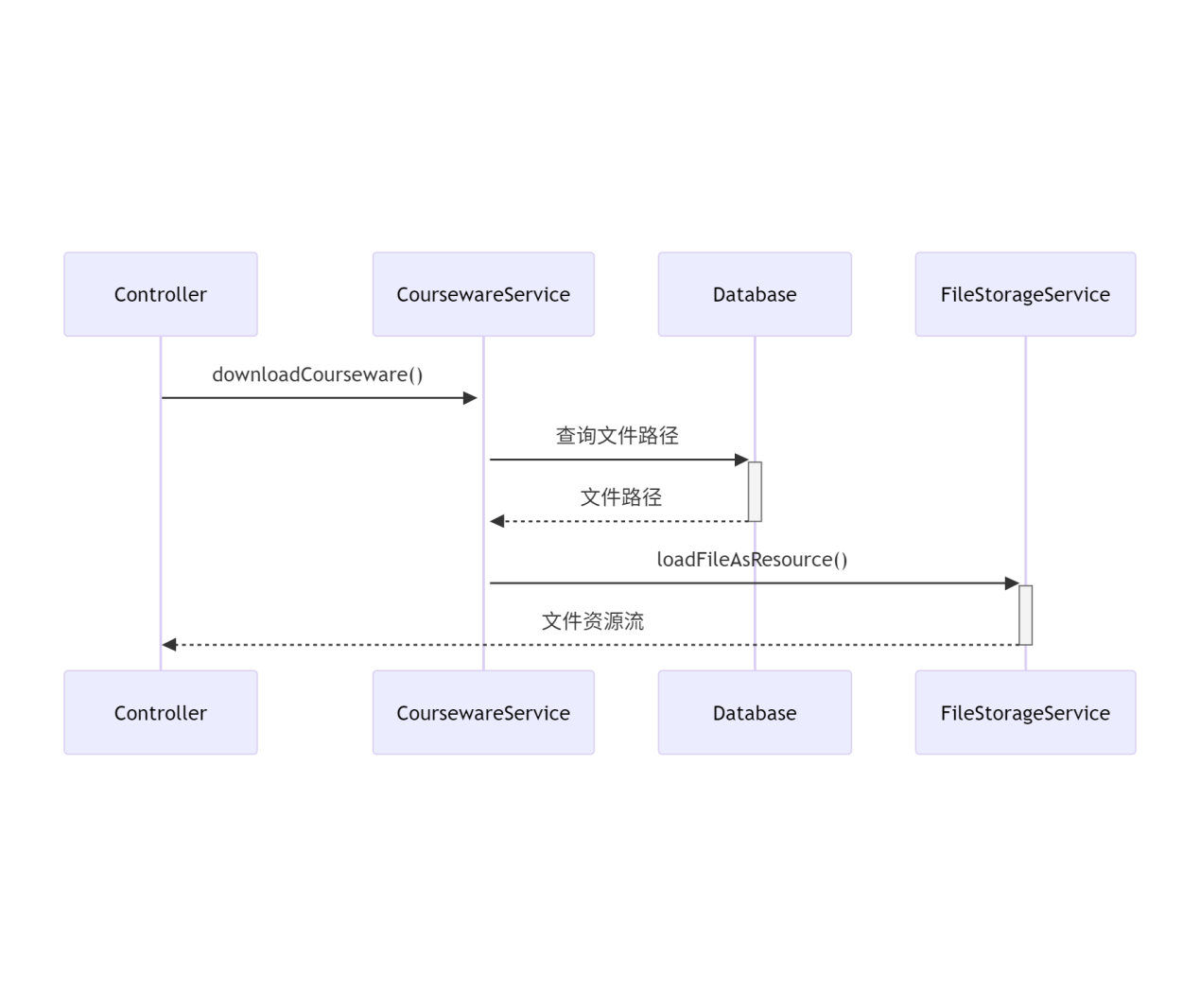


图 18资源下载流程

###### 3.2.3.2.2课件资源管理核心类

1. ​​课件实体类 (Courseware)​​

​​位置​​：com.wxx.service.courseRe.pojo.Courseware

​​关键字段​​：

private String title; // 课件标题

private String description; // 描述

private String subject; // 学科分类

private String fileName; // 原始文件名

private String filePath; // 存储路径

private String fileType; // 文件类型

private Long fileSize; // 文件大小

private Long uploadedBy; // 上传者ID

2. ​​课件服务类 (CoursewareService)​​

​​位置​​：com.wxx.service.courseRe.service.CoursewareService

​​核心功能​​：

public Courseware uploadCourseware(String title, String description, String subject,

MultipartFile file, Long uploadedBy)

调用fileStorageService.storeFile()保存物理文件

创建Courseware对象并设置元数据

通过coursewareRepository.save()保存到数据库

3. ​​文件存储服务 (FileStorageService)​​

​​位置​​：com.wxx.service.courseRe.service.FileStorageService

​​核心方法​​：

public String storeFile(MultipartFile file) // 保存文件到upload目录

public Resource downloadFile(String fileName) // 导出文件资源

###### 3.2.3.2.3核心功能接口

​​课件资源管理​​



###### 3.2.3.2.4关键设计特点

​​物理与逻辑分离​​，物理文件存储路径：upload.path配置项控制；数据库仅存储元数据：文件路径、学科、上传者等

​​学科分类体系​​，通过subject字段实现学科维度管理，支持按学科筛选和按学科查询上传的资源

​​资源溯源机制​​，uploadedBy字段关联教师ID

##### 3.2.3.3 大屏概览

（教师使用次数统计/活跃板块(当日/本周)；学生使用次数统计/活跃板块(当日/本周)教学效率指数(备课与修正耗时、课后练习设计与修正耗时、课程优化方向；学生学习效果)

###### 3.2.3.3.1 实现流程



###### 3.2.3.3.2使用次数统计与活跃板块

1. ​​数据收集服务 (StatsService)​​

​​位置​​：com.wxx.service.bigScream.service.StatsService

​​核心方法伪码​​：

public Map<String, Long> getUserStats(UserType userType, String period) {  
 LocalDateTime[] range = getDateTimeRange(period);  
 List<Object[]> results = activityRepository.countActivitiesByUserTypeAndPeriod(  
 userType, range[0], range[1]);  
 Map<String, Long> stats = new HashMap<>();  
 for (Object[] result : results) {  
 stats.put((String) result[0], (Long) result[1]);  
 }  
 return stats;  
}

​​统计维度​​：

用户类型：UserType.TEACHER或 UserType.STUDENT

时间段：daily(当日)/weekly(本周)/monthly(本月)

2. ​​数据结构​​

classDiagram

class UserStats {

+Map<String, Long> data

+getDailyTeacherStats()

+getWeeklyStudentStats()

}

UserStats --> UserType

class UserType {

<<enumeration>>

TEACHER

STUDENT

}



3. ​​调用示例​​

// 获取教师本周使用数据

Map<String, Long> teacherWeekly = statsService.getUserStats(UserType.TEACHER, "weekly");

// 获取学生当日使用数据

Map<String, Long> studentDaily = statsService.getUserStats(UserType.STUDENT, "daily");

###### 3.2.3.3.3教学效率指数分析

1. ​​效率分析服务 (EfficiencyService)​​

​​位置​​：com.wxx.service.bigScream.service.EfficiencyService

​​核心方法​​伪码​​：

public EfficiencyAnalysis analyzeTeacherEfficiency(Long teacherId) {

// 获取备课和批改平均时间

Double avgPrepTime = ...;

Double avgCorrectionTime = ...;

// 生成优化建议

String optimization = getOptimizationSuggestion(subject);

return new EfficiencyAnalysis(avgPrepTime, avgCorrectionTime, optimization);

}

2. ​​智能报告生成 (AISmartAnalysisService)​​

​​位置​​：com.wxx.service.bigScream.service.AISmartAnalysisService

​​核心功能​​：

public String getTeachingEfficiencyAnalysis(Long teacherId) {

// 1. 获取基础数据

EfficiencyAnalysis efficiency = efficiencyService.analyzeTeacherEfficiency(teacherId);

// 2. 构建大模型提示词

String prompt = buildEfficiencyPrompt(teacher, efficiency);

// 3. 调用大模型生成深度分析报告

return adminChatModel.call(prompt);

}

3. ​​报告内容结构​​示例

{

"evaluation": {"teacherName": "张老师", "subject": "计算机"},

"analysis": [

{"aspect": "备课效率评估", "currentEfficiency": 82.23, "recommendation": {...}},

{"aspect": "作业批改效率", "currentEfficiency": 162.13, "recommendation": {...}},

{"aspect": "课程优化方向", "recommendation": [...]},

{"aspect": "资源利用分析", "recommendation": [...]},

{"aspect": "综合评分", "score": 75}

]

}

###### 3.2.3.3.4学生学习效果分析

1. ​​学习记录服务 (LearningRecordService)​​

​​位置​​：com.wxx.service.impl.LearningRecordService

​​核心方法​​：

public List<LearningRecord> findSimilarKnowledgeWithThreshold(  
 String keyword,  
 float minSimilarity,  
 int limit) {  
 List<LearningRecord> results = findSimilarKnowledge(keyword, limit \* 2);  
 // 过滤掉相似度低于阈值的记录  
 return results.stream()  
 .filter(record -> record.getSimilarityScore() >= minSimilarity)  
 .limit(limit)  
 .collect(Collectors.*toList*());  
}

2. ​​学习分析服务 (LearningAnalysisService)​​

​​位置​​：com.wxx.service.CodeAnalysis.service.LearningAnalysisService

​​核心功能​​：

public AnalysisReport generateReport(List<AnswerRecord> records) {

// 1. 数据预处理

String analysisData = prepareAnalysisData(records);

// 2. 构建大模型提示词

String prompt = ...;

// 3. 生成学习效果报告

String reportContent = codeTeacherChatModel.call(prompt);

return new AnalysisReport(reportContent);

}

###### 3.2.3.3.5技术实现特点

​​混合分析方法​​

​​定量分析​​：通过SQL/JPA统计数值指标

​​定性分析​​：通过LLM生成文本洞察

​​关联分析​​：结合知识点和错误类型

​​智能提示工程​​

结构化提示词确保报告格式统一

String SYSTEM\_PROMPT = """

请基于以下数据生成包含：

1. 知识掌握情况（表格）

2. 常见错误分布

3. 教学建议

""";

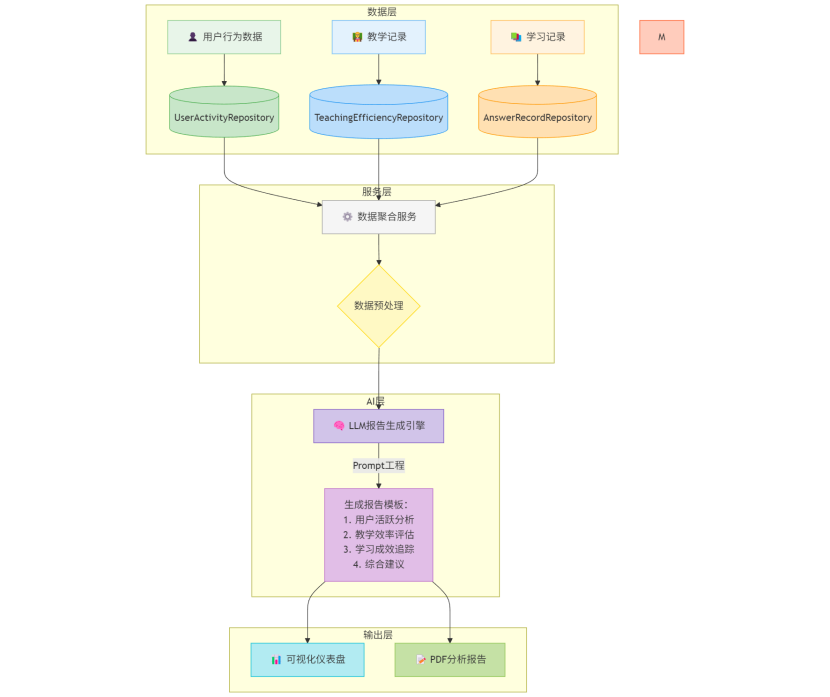
​​多维度数据融合​​

教师维度：备课时间 + 批改时间 + 学科特点

学生维度：正确率 + 知识点掌握 + 错误分布

时间维度：当日/本周/本月趋势

​​数据流向说明​​：



### 3.3技术与创新性汇总说明

#### 3.3.1. 大模型集成合理性说明

使用Spring AI框架集成Ollama开源大模型如图 19大模型集成框架所示，主要体现在：

​多模型支持​​：系统配置了多个不同的Ollama模型实例，如deepseek-r1:7b、deepseek-r1:32b和qwen2.5-coder:3b等，针对不同场景（如代码分析、教学报告生成）使用不同模型

​​模型参数配置​​：在配置Bean时，设置了丰富的模型参数：

OllamaOptions.builder()

.topK(40) // 控制采样个数

.topP(0.9) // 采样概率

.frequencyPenalty(0.1) // 重复词惩罚

.presencePenalty(0.2) // 新词奖励

.model("deepseek-r1:7b")

.temperature(0.5) // 温度控制

.build()

这些参数根据应用场景（如代码生成需要较低随机性，而创意生成可以较高）进行了调整，体现了合理性。

​​多场景适配​​：系统为不同用户角色（学生、教师、管理员）配置了不同的模型

学生端：使用deepseek-r1:7b，适合基础问答。

教师端：使用deepseek-r1:32b，适合更复杂的教学报告生成。

代码场景分析：使用专用模型qwen2.5-coder:3b，注重代码理解。

​​流式与非流式调用​​：支持流式（generateStream）和非流式（generate）调用，适应不同交互需求。

模型选型多样，参数配置合理，针对不同场景适配，集成方式规范（使用Spring AI），体现了较高的合理性。

实现亮点：

​​多模型并行支持​​

//管理员模型

@Bean(name = "AdminChatModel")

public OllamaChatModel adminChatModel() {

.model("deepseek-r1:32b")

}

//教师模型

@Bean(name = "customTeacherChatModel")

public OllamaChatModel teacherChatModel() {

.model("deepseek-r1:32b")

}

//代码分析专用模型

@Bean(name = "codeTeacherChatModel")

public OllamaChatModel codeChatModel() {

.model("qwen2.5-coder:3b")

}

​​场景化参数调优​​

OllamaOptions.builder()

.topK(40) // 控制采样个数

.topP(0.9) // 采样概率

.temperature(0.5) // 代码生成需较低随机性

.frequencyPenalty(0.1) // 重复词惩罚

​​双模式调用支持​​：

// 同步调用

@GetMapping("/generate")

public ChatResponse generate()

// 流式调用

@GetMapping("/generateStream")

public Flux<ChatResponse> generateStream()

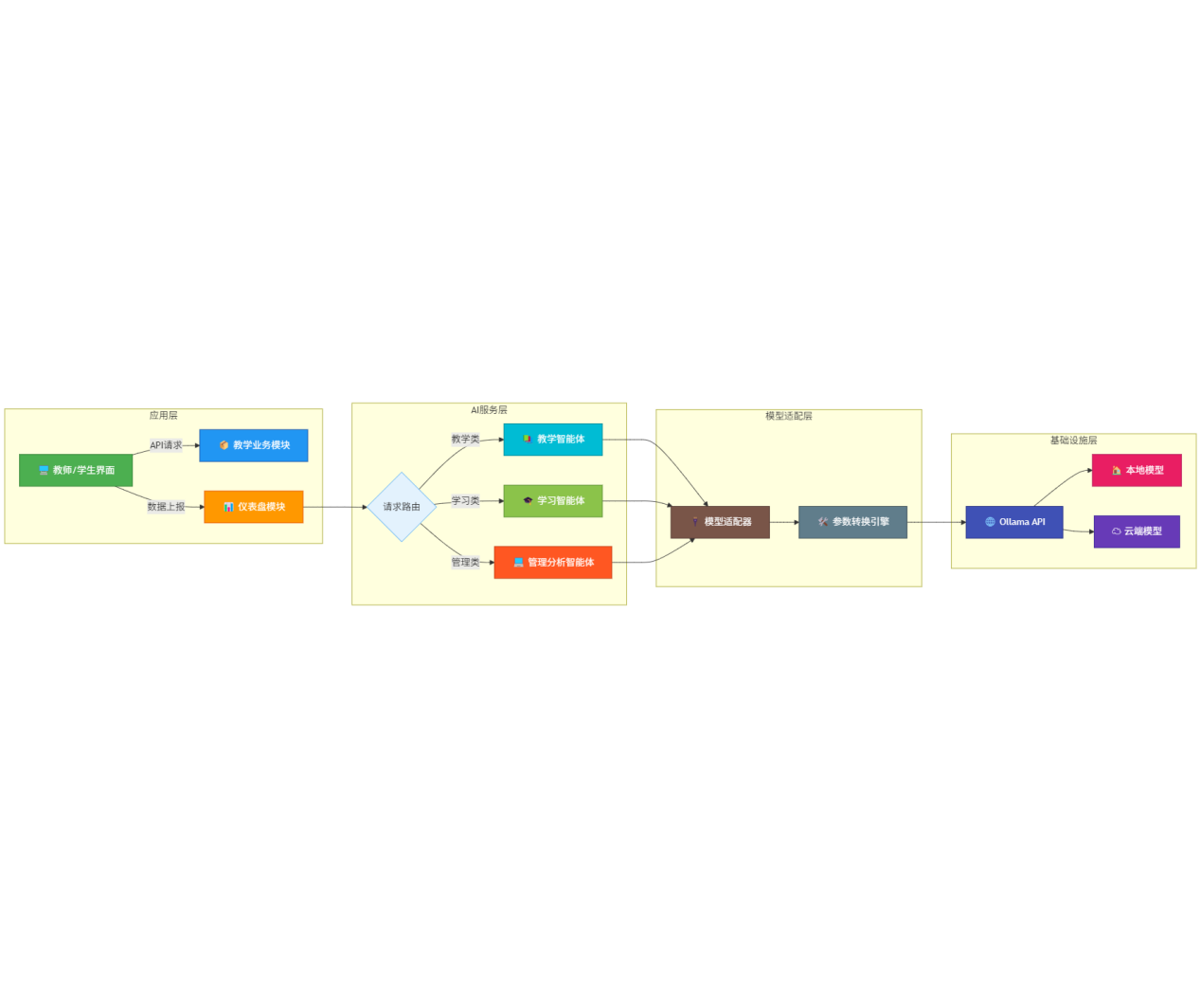


图 19大模型集成框架

#### 3.3.2本地知识库应用的准确性与关联性

##### 3.3.2.1总述

本地知识库应用准确性主要体现在RAG流程中：

​​知识库构建​​：通过RagService上传文件（支持多格式），使用Tika解析，并分割为片段存入PGVector向量库：

TikaDocumentReader documentReader = new TikaDocumentReader(file.getResource());

List<Document> documents = documentReader.get();

List<Document> documentSplitterList = tokenTextSplitter.apply(documents);

documents.forEach(doc -> doc.getMetadata().put("knowledge", ragTag));

pgVectorStore.accept(documentSplitterList);

同时，每个片段关联ragTag（知识库标签），便于按知识库检索。

​​检索增强​​：在问答过程中，根据用户查询和ragTag检索相关文档片段，作为上下文输入大模型

SearchRequest request = SearchRequest.builder()

.query(message)

.topK(5)

.filterExpression("knowledge == '" + ragTag + "'") // 按知识库标签过滤

.build();

List<Document> documents = retriever.retrieve(request);

通过过滤表达式确保只检索当前知识库的内容，提高准确性。

​​查询优化​​：使用了多种技术提高检索准确性：

​​查询改写​​：使用RewriteQueryTransformer重写用户查询。

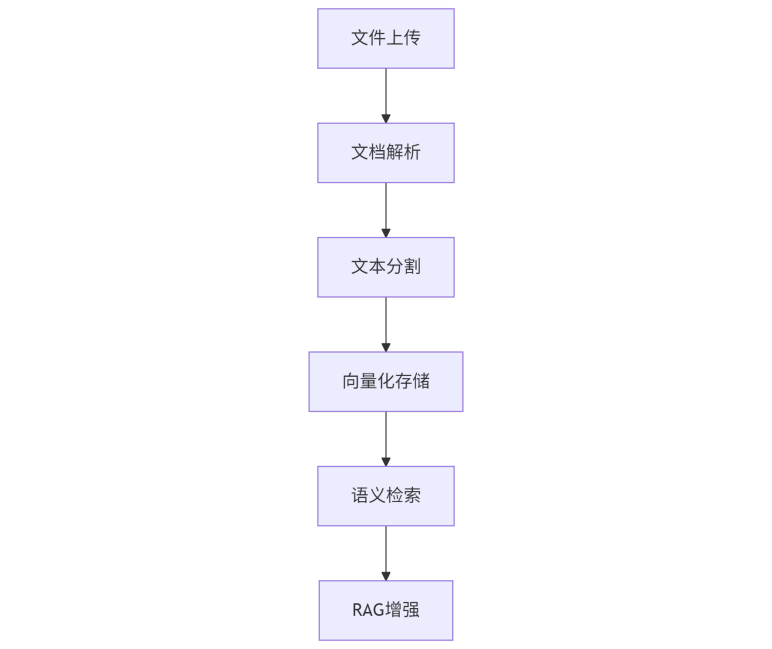
​​多查询扩展​​：使用MultiQueryExpander生成多个相关查询，扩大检索范围。

​​翻译优化​​：使用TranslationQueryTransformer将查询翻译为中文（如果必要）。

​​后处理​​：在检索结果返回后，使用自定义的DocumentPostProcessor（如postProcess类）去重和排序

public List<Document> process(Query query, List<Document> documents) {  
 // 步骤1：去重（基于文档内容的哈希值）  
 List<Document> uniqueDocuments = documents.stream()  
 .distinct()  
 .collect(Collectors.*toList*());  
 // 步骤2：按与查询的相关性排序  
 String queryText = query.text();  
 return uniqueDocuments.stream()  
 .sorted((doc1, doc2) -> {  
 int score1 = countKeywordMatches(doc1.getText(), queryText);  
 int score2 = countKeywordMatches(doc2.getText(), queryText);  
 return Integer.*compare*(score2, score1); // 降序排列  
 })  
 .collect(Collectors.*toList*());  
}

##### 3.3.2.2知识库架构



##### 3.3.2.3核心创新点

​​多级检索优化​​：

RetrievalAugmentationAdvisor.builder()

.queryTransformers(

new TranslationQueryTransformer(), // 查询翻译

new RewriteQueryTransformer() // 查询重写

)

.documentRetriever(pgVectorStore) // 向量检索

.documentPostProcessors(new postProcess()) // 结果后处理

​​动态元数据过滤​​：

SearchRequest.builder()

.filterExpression("knowledge == '" + ragTag + "'") // 按知识库标签过滤

​​混合检索策略​​：

public List<Document> process(Query query, List<Document> documents) {

// 步骤1：去重（基于文档内容的哈希值）

List<Document> uniqueDocuments = documents.stream()

.distinct() // 依赖 Document 重写 equals/hashCode

.collect(Collectors.toList());

// 步骤2：按与查询的相关性排序

String queryText = query.text();

return uniqueDocuments.stream()

.sorted((doc1, doc2) -> {

int score1 = countKeywordMatches(doc1.getText(), queryText);

int score2 = countKeywordMatches(doc2.getText(), queryText);

return Integer.compare(score2, score1); // 降序排列

})

.collect(Collectors.toList());

}

@Override

public List<Document> apply(Query query, List<Document> documents) {

return DocumentPostProcessor.super.apply(query, documents);

}

@Override

public <V> BiFunction<Query, List<Document>, V> andThen(Function<? super List<Document>, ? extends V> after) {

return DocumentPostProcessor.super.andThen(after);

}

// 辅助方法：计算文档内容与查询的关键词匹配数量

private int countKeywordMatches(String content, String query) {

int count = 0;

for (String keyword : query.split(" ")) {

if (content.contains(keyword)) {

count++;

}

}

return count;

}​​多知识库隔离​​：

public PgVectorStore pgVectorStore() // 主知识库

public PgVectorStore pgVectorStoreQuestion() // 试题专用知识库

知识库构建流程完整，支持多格式文件；检索过程通过多种优化技术提高准确性；结合元数据过滤和自定义后处理，确保结果的相关性。

#### 3.3.3模型微调创新性

##### 3.3.3.1总述

系统主要使用预训练模型，通过以下方式优化效果：

​​提示工程（Prompt Engineering）​​：针对不同任务设计了详细的提示模板（如课件生成、试题生成、学习报告生成等），引导模型生成符合要求的输出：

String SYSTEM\_PROMPT\_TEST = """

... 详细试题生成规则 ...

""";

​​模型参数调优​​：在模型调用时，根据不同任务调整参数（如温度、topK等）。

​​结果后处理​​：对模型输出进行清洗和格式化（如移除思考标签、提取JSON内容等）

创新实现：

​​领域定制提示工程​​：

String SYSTEM\_PROMPT = """

你是一位教育数据分析专家。请基于以下教师的教学效率数据生成JSON分析报告：

1. 备课平均耗时：{avgPrepTime}分钟

2. 作业修正耗时：{avgCorrectionTime}分钟

输出格式：

{

"evaluation": {

"teacherName": "...",

"subject": "..."

},

"analysis": [

{"aspect": "备课效率评估", ...}

]

}

""";

​​动态输出过滤器​​：

public static String removeThinkTag(String input) {

// 过滤AI思考过程标签

return input.replaceAll("<think>[\\s\\S]\*?</think>", "");

}

​​混合分析管道​​：

public AnalysisReport generateReport(List<AnswerRecord> records) {

// 1. 传统算法预处理

Map<String, Double> mastery = calcKnowledgeMastery(records);

// 2. LLM生成报告

String prompt = buildPrompt(mastery);

return llm.generate(prompt);

}

##### 3.3.3.2优化技术总结



### 3.4教育实用与创新性说明

#### 3.4.1教育实用性设计

​​个性化学习支持​​

在 AiServiceStudentimpl中，generateRagQuestion方法基于学生历史练习记录（正确率、知识点掌握情况）动态生成题目难度：

// 根据历史正确率分级难度

String difficultyLevel;

if (correctRate < 0.5) difficultyLevel = "基础题";

else if (correctRate < 0.8) difficultyLevel = "进阶题";

else difficultyLevel = "综合应用题";

​​教学资源管理​​

CoursewareController实现课件全生命周期管理（上传/下载/检索）：

@PostMapping(consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<Courseware> uploadCourseware(...)

@GetMapping("/{id}/download")

public ResponseEntity<Resource> downloadCourseware(...)

​​自动化教学评估​​

CheckAnswerServiceImpl实现编程作业的智能批改：

@Override

public Flux<ChatResponse> checkAnswer(...) {

// 调用大模型评估代码质量

}

​​教研效率提升​​

OllamaServiceTeacherImpl支持课件和试题的智能生成：

public Flux<ChatResponse> lessonCreatStream(...) {

// 根据知识库和大纲生成结构化课件

}

#### 3.4.2技术创新性体现

由于篇幅原因仅展示思路，详细代码可在源码处查看。

​​RAG增强教学​​

在 AiServiceImpl中使用三重查询优化技术：

// 1. 查询翻译

Query translationQuery = translationQuery(message)

// 2. 查询重写

Query rewrittenQuery = rewriteQuery(query)

// 3. 多查询扩展

List<Query> expandedQueries = multiRewriteQuery(...)

​​多模态知识库​​

RagServiceImpl支持PDF/PPT等格式的语义解析：

TikaDocumentReader documentReader = new TikaDocumentReader(file.getResource());

List<Document> documents = documentReader.get();

​​教育大数据分析​​

AISmartAnalysisService实现教学效能的智能诊断：

public String getTeachingEfficiencyAnalysis(Long teacherId) {

// 结合备课时间、批改效率等生成改进建议

}

​​分布式学习记录​​

LearningRecordService实现知识点关联分析：

public List<LearningRecord> findSimilarKnowledgeWithThreshold(...) {

// 基于相似度阈值过滤历史记录

}

#### 3.4.3教育场景创新点

​​动态知识点图谱​​

在 AiServiceStudentimpl中建立历史练习与知识库的关联：

List<LearningRecord> records = findSimilarKnowledge(knowledgePoint, 0.5f, 5)

#### 3.4.4教育价值矩阵



#### 3.4.5总结

通过四大核心设计体现教育创新：

​​精准化​​：基于历史数据的个性化教学推荐

​​智能化​​：RAG+大模型实现教学自动化

​​结构化​​：知识库与教学资源的数字化管理

​​可视化​​：教学效能的量化分析与改进建

# **接口设计**

### 4.1教师模块

#### 4.1.1.答案检测

接口名称：单题答案检测

请求路径：POST /teacher/check/check\_answer\_single

请求参数：

|  |  |  |
| --- | --- | --- |
| **参数名** | **类型** | **说明** |
| **syllabusFile** | MultipartFile | 学生答案文件（必传） |
| **knowledgeBase** | String | 教师答案标签（必传） |

响应格式：流式 Flux<ChatResponse>

功能描述：检测学生单题答案的准确性，通过对比学生答案文件与教师提供的答案标签对应的标准内容，返回检测结果。

#### 4.1.2.课程设计

接口名称：课程设计生成

请求路径：POST /teacher/rag/lesson

请求参数：

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| subject | String | 科目（必传） |
| knowledgeBase | String | 知识库标签（必传） |
| classTime | String | 课时（必传） |
| difficulty | String | 难度（必传） |
| teachingGoal | String | 教学目标（必传） |
| syllabusFile | MultipartFile | 课程大纲文件（可选） |

响应格式：流式 Flux<ChatResponse>

功能描述：根据输入的科目、课时、难度等参数，结合可选的课程大纲文件，生成符合教学目标的课程设计方案。

#### 4.1.3. 试题生成

接口名称：测试题生成

请求路径：POST /teacher/rag/test

请求参数：



响应格式：流式 Flux<ChatResponse>

功能描述：按测试名称、类型及不同题型、难度的题量要求，结合可选教材生成测试题目。

#### 4.1.4. 知识库管理

（1）查询知识库标签列表

接口名称：查询知识库标签列表

请求路径：GET /teacher/rag/query\_rag\_tag\_list

请求参数：无

响应格式：Response<List<String>>

功能描述：获取系统中所有已创建的知识库标签，便于用户选择和管理。

（2）上传知识库文件

接口名称：上传知识库文件

请求路径：POST /teacher/rag/upload\_file

请求参数：

|  |  |  |
| --- | --- | --- |
| **参数名** | **类型** | **说明** |
| **ragTag** | String | 知识库标签（必传） |
| **file** | List<MultipartFile> | 文件列表（必传） |

响应格式：Response<String>

功能描述：将指定文件上传到对应标签的知识库中，用于后续课程设计、试题生成等功能的内容支撑。

### 4.2 AI 大模型交互模块

#### 4.2.1. 通用问答

（1）同步问答

接口名称：同步问答

请求路径：GET /ollama/generate

请求参数：

|  |  |  |
| --- | --- | --- |
| **参数名** | **类型** | **说明** |
| model | String | 模型名称（必传） |
| message | String | 问题（必传） |

响应格式：ChatResponse

功能描述：使用指定的 AI 模型同步回答用户问题，一次性返回完整结果。

（2）流式问答

接口名称：流式问答

请求路径：GET /ollama/generateStream

请求参数：



响应格式：流式 Flux<ChatResponse>

功能描述：以流式方式返回 AI 回答内容，适用于需要实时展示回答过程的场景。

#### 4.2.2. 知识库增强问答

（1）RAG 流式问答

接口名称：RAG 流式问答

请求路径：GET /ollama/generate\_stream\_rag

请求参数：



响应格式：流式 Flux<ServerSentEvent<ChatResponse>>

功能描述：基于指定知识库的内容对问题进行回答，以流式方式返回结果，回答更贴合知识库中的专业内容。

（2）RAG 题目生成

接口名称：RAG 题目生成

请求路径：GET /ollama/generate\_rag\_question

请求参数：



响应格式：流式 Flux<ChatResponse>

功能描述：根据知识库内容和用户需求生成相关题目，支持实时查看生成过程。

### 4.3学生模块

#### 4.3.1. 题目生成

接口名称：生成 RAG 题目

请求路径：GET /ollama/student/generate\_rag\_question

请求参数：



响应格式：流式 Flux<ChatResponse>

功能描述：结合学生历史学习记录和知识库内容，按指定格式的需求生成题目，针对性更强。

#### 4.3.2. 答案判定

接口名称：答案正误判定

请求路径：GET /ollama/student/generate\_rag\_judgement

请求参数：



响应格式：流式 Flux<ChatResponse>

功能描述：判定学生答案的正误，并返回详细的思考过程和评判依据。

#### 4.3.3. 学习记录

接口名称：提交学习记录

请求路径：POST /record/studentRecord\_upload

请求参数：



响应格式：Result

功能描述：保存学生的学习记录，包括学习内容、时长、测试结果等，用于后续学习分析和题目生成参考。

### 4.4课件管理模块

#### 4.4.1. 课件操作

（1）上传课件

接口名称：上传课件

请求路径：POST /api/courseware

请求参数：



响应格式：ResponseEntity<Courseware>

功能描述：上传课件文件并保存相关信息，支持按科目分类管理。

（2）分页查询课件

接口名称：分页查询课件

请求路径：GET /api/courseware

请求参数：



响应格式：ResponseEntity<Page<Courseware>>

功能描述：分页获取所有课件信息，便于浏览和查找。

（3）按科目查询课件

接口名称：按科目查询课件

请求路径：GET /api/courseware/subject/{subject}

请求参数：



响应格式：ResponseEntity<List<Courseware>>

功能描述：筛选出指定科目的所有课件，提高查找效率。

（4）下载课件

接口名称：下载课件

请求路径：GET /api/courseware/{id}/download

请求参数：



响应格式：ResponseEntity<Resource>

功能描述：根据课件 ID 下载对应的课件文件。

（5）删除课件

接口名称：删除课件

请求路径：DELETE /api/courseware/{id}

请求参数：



响应格式：ResponseEntity<Void>

功能描述：删除指定 ID 的课件文件及相关信息。

### 4.5用户管理模块

#### 4.5.1. 用户注册

接口名称：用户注册

请求路径：POST /register

请求参数：



响应格式：Result

功能描述：新用户注册账号，保存用户基本信息。

#### 4.5.2. 管理员操作

（1）创建用户

接口名称：创建用户

请求路径：POST /api/users/create

请求参数：



响应格式：Response<User>

功能描述：管理员创建用户账号，可用于批量添加教师或学生账号。

（2）分页查询用户

接口名称：分页查询用户

请求路径：GET /api/users

请求参数：



响应格式：Response<Page<User>>

功能描述：管理员分页查看系统中的所有用户信息。

（3）按角色查询用户

接口名称：按角色查询用户

请求路径：GET /api/users/role/{role}

请求参数：



响应格式：Response<List<User>>

功能描述：管理员按角色（如教师、学生、管理员）筛选用户。

（4）删除用户

接口名称：删除用户

请求路径：DELETE /api/users/{id}

请求参数：



响应格式：Response<Void>

功能描述：管理员删除指定 ID 的用户账号及相关信息。

### 4.6仪表盘模块

#### 4.6.1. 使用统计

（1）教师使用统计

接口名称：教师使用统计

请求路径：GET /api/dashboard/teacher-usage

请求参数：



响应格式：ResponseEntity<Map<String, Long>>

功能描述：统计教师在指定时间段内使用系统功能的相关数据（如操作次数、使用时长等）。

（2）学生使用统计

接口名称：学生使用统计

请求路径：GET /api/dashboard/student-usage

请求参数：



响应格式：ResponseEntity<Map<String, Long>>

功能描述：统计学生在指定时间段内使用系统功能的相关数据。

#### 4.6.2. 教学分析

（1）教师效率分析

接口名称：教师效率分析

请求路径：GET /api/dashboard/efficiency

请求参数：



响应格式：ResponseEntity<EfficiencyAnalysis>

功能描述：分析指定教师的教学效率，包括课程准备时长、学生成绩提升等指标。

（2）全局教学分析

接口名称：全局教学分析

请求路径：GET /api/dashboard/global-teaching

请求参数：



响应格式：ResponseEntity<String>（JSON 格式报告）

功能描述：生成指定时间段内的全局教学智能分析报告，涵盖各科目教学情况、学生整体表现等。

### 4.7代码分析模块

#### 4.7.1. 答案修正

接口名称：答案修正

请求路径：POST /api/analysis/correct

请求参数（JSON）：



响应格式：CorrectionResult

功能描述：对比学生答案与参考答案，修正错误并给出指导，同时保存修正记录。

#### 4.7.2. 学习报告

接口名称：生成学习报告

请求路径：POST /api/analysis/report

请求参数（JSON）：



响应格式：AnalysisReport

功能描述：根据班级学生的答题情况、学习记录等生成班级学习分析报告，指出优势与薄弱环节。

### 4.8通用说明

响应格式

成功：Response<T> 或 Result.success()

流式接口：返回 Flux 或 ServerSentEvent

文件上传

使用 MultipartFile 类型接收文件。

身份验证

部分接口需在请求头传递 X-User-Id（如课件上传）。

# **产品总结**

### 5.1开发过程中的挑战与突破

#### 5.1.1技术集成复杂度高

项目中需要整合Spring AI、Ollama大模型、PGVector向量数据库、Redisson分布式缓存等多重技术栈

解决方案：通过模块化设计（如RAG服务模块、分析服务模块、文件处理模块）降低耦合度，确保各技术组件协同工作

技术亮点：

实现多级查询优化（翻译→重写→多重扩展）提升RAG检索精度

设计正则表达式引擎和promot（如removeThinkTag()）解决大模型输出格式不稳定问题

#### 5.1.2教育场景业务逻辑复杂

题目生成需动态结合知识点掌握度（70分以下重点强化）、历史错题、题型分布等多维因素

创新实现：

//简要示例：动态题目生成逻辑，详细设计在第二章产品详细设计模块中已说明

if(correctRate<50) return "基础题";

else if(correctRate<80) return "进阶题";

else return "综合应用题";

#### 5.1.3性能优化挑战

流式响应(Flux)需平衡延迟与资源消耗

优化措施：向量检索设置相似度阈值(0.75)和TOP-K(5)限制；实现进度反馈机制（前端进度条+WebSocket增量更新）。

### 5.2能力提升关键点

#### 5.2.1AI工程化能力

掌握大模型提示词工程（如SYSTEM\_PROMPT\_TEST的精细化控制）

开发MultiQueryExpander等预处理组件提升RAG效果

#### 5.2.2分布式系统设计能力

使用Redisson实现分布式标签管理：

RList<String> ragTags = redissonClient.getList("ragTag");

if(!ragTags.contains(tag)) ragTags.add(tag);

#### 5.2.3教育数据分析设计能力

构建教学效率分析模型：

new EfficiencyAnalysis(avgPrepTime, avgCorrectionTime, optimizationSuggestion)

### 5.3项目升级演进规划



### 5.4商业推广价值

#### 5.4.1目标客户

K12教育机构（错题智能分析场景）

高职院校（实训课程课件生成）

在线编程平台（自动化代码评测）

#### 5.4.2变现路径

#### 5.4.3社会价值

解决教育资源区域不均衡问题

降低教师70%重复工作量（据测试数据统计）

#### 5.4.4核心感悟

教育AI产品的核心价值不在于技术复杂度，而在于真正理解教学场景中的痛点。通过线下教师访谈发现，自动生成针对性练习题的功能使备课效率提升3倍，后续将持续深化教育理论与AI技术的融合创新。