AI复习平台开发 0805

## 环节三：知识点题目匹配与AI Tutor对话

### 功能实现顺序

#### 步骤一：知识点标签提取与匹配

**目标**: 从Summary中提取知识点并匹配题库标签

**具体任务**:

1. **解析Summary中的知识点标记**
   1. 识别[KP:...]格式的知识点标记
   2. 提取知识点描述文本
2. **LLM智能标签分类**
   1. 使用LLM理解知识点内容
   2. 从mapping.csv中的标签库选择最匹配的topic\_code
   3. 生成知识点到标签的映射关系

**注意事项**:

* 需要设计精确的prompt让LLM理解mapping.csv中的分类体系
* 要处理一个知识点可能对应多个标签的情况
* 建立评估机制验证标签匹配的准确性

#### 步骤二：题目匹配系统

**目标**: 根据知识点标签从AMC题库中筛选相关题目

**具体任务**:

1. **题库索引建立**
   * 解析AMC10\_v250722.csv中的题目数据
   * 建立topic\_code到题目的索引映射
2. **智能题目筛选**
   * 根据知识点标签匹配相关题目
   * 按难度、相关性等因素排序
   * 为每个知识点准备题目池

**注意事项**:

* 需要处理题目数据的结构和格式
* 考虑题目难度梯度，从易到难
* 建立题目质量评估机制

#### **步骤三：前端UI交互设计**

**目标**: 在Summary页面添加"相关题目"按钮和对话框

**具体任务**:

1. **按钮集成**
   * 在每个知识点板块添加"相关题目"按钮
   * 设计按钮样式和位置
2. **对话框界面**
   * 创建弹出式对话框组件
   * 设计对话界面布局（题目展示区、对话区、输入区）
   * 实现对话框的打开/关闭逻辑

**注意事项**:

* 确保UI响应式设计，适配不同屏幕
* 对话框要有良好的用户体验（加载状态、错误处理）
* 考虑多个对话框同时打开的情况

#### **步骤四：AI Tutor对话系统**

**目标**: 实现智能教学对话机器人

**具体任务**:

1. **Prompt设计**
   * 设计苏格拉底式提问的prompt模板
   * 实现scaffolding（脚手架式）引导策略
   * 设计hint提示机制
2. **对话流程控制**
   * 实现多轮对话状态管理
   * 设计学生理解度评估逻辑
   * 建立自适应难度调整机制
3. **教学策略实现**
   * **引导式提问**: 不直接给答案，通过问题引导思考
   * **分步解析**: 将复杂问题分解为小步骤
   * **概念检查**: 验证学生对基础概念的理解
   * **错误纠正**: 识别常见错误并给出针对性指导

**注意事项**:

* Prompt需要大量测试和调优
* 要设计对话轮次限制，防止无限循环
* 需要处理学生的各种回答类型（正确、错误、不知道等）
* 建立对话质量评估机制

#### **步骤五：后端API设计**

**目标**: 提供稳定的API接口支持前端功能

**具体任务**:

1. **知识点匹配API**

POST /api/match-questions- 输入: 知识点ID- 输出: 匹配的题目列表

1. **对话管理API**

POST /api/tutor/start-session- 输入: 题目ID, 学生ID- 输出: 会话ID, 初始问题POST /api/tutor/chat- 输入: 会话ID, 学生回答- 输出: AI回复, 会话状态

**注意事项**:

* 实现会话状态持久化
* 设计错误处理和重试机制
* 考虑并发用户的性能问题

#### **步骤六：测试与优化**

**目标**: 确保系统稳定性和教学效果

**具体任务**:

1. **单元测试**
   * 知识点标签匹配准确性测试
   * 题目筛选逻辑测试
   * 对话流程测试
2. **教学效果评估**
   * 设计学习效果指标
   * 收集用户反馈
   * A/B测试不同的prompt策略

**注意事项**:

* 建立自动化测试流程
* 设计用户行为分析机制
* 准备prompt调优的迭代流程

### 建议的开发顺序

1. **先完成步骤一和二**（后端核心逻辑）
2. **再实现步骤五**（API接口）
3. **然后开发步骤三**（前端界面）
4. **重点攻克步骤四**（AI Tutor，最复杂）
5. **最后进行步骤六**（测试优化）

首先：用已经生成的Summary文件进行知识点匹配的测试，测试匹配的准确性和相关性，没有问题了之后进行下一步开发

## 题目匹配

### 匹配逻辑：

多层验证：

*# 第一层：关键词匹配*

简单：关键词预测topic\_code → 遍历AMC题库 → 匹配Topica/Topicg/Topics/Topicc字段 → 按相关性排序 → 返回前N道题

*# 第二层：语义相似度*

语义理解匹配，Promptlike:

prompt = f""" 知识点: {knowledge\_point} 题目内容: {question\_content} 这个题目是否真的与知识点匹配？请分析： 1. 知识点讲的是什么概念？ 2. 题目考察的是什么技能？ 3. 匹配度 (0-10分) """

*# 第三层：LLM验证匹配合理性*

*# 第四层：用户反馈学习*

### 完整的测试流程

1. 人工标注预期结果 (基于Topic) → "标准答案"
2. 运行匹配算法 → "实际结果"
3. 对比预期vs实际 → "准确率评估"
4. 根据结果调优算法 → "改进匹配逻辑"

### 测试数据集

先生成一个测试数据集，JSON格式，包含2-3道真正的概率题 + 3-4道其他类型题目作为对比

实现快速匹配测试，验证四层匹配逻辑的效果

之后如果效果好，再考虑自动化PDF处理

0805匹配逻辑：

✅ 基础关键词匹配工作正常

✅ Topic标签匹配有效

#### **测试**结果

#### **准确率**评估

#### 下一步

##### ✅ ****Option 1****: 前端集成

* 在现有Summary页面添加按钮
* 实现题目推荐弹窗

##### ✅ ****Option 2****: ****测试****和优化

* 扩展PDF测试题库到20-30道题
* 添加更多知识点测试
* 完善准确率评估

##### ✅ ****Option 3****: 改进匹配算法

* 添加语义相似度匹配
* 实现简单的LLM验证

##### ✅ ****Option 4****: AI Tutor系统

* 设计对话流程
* 实现教学策略

## 从题目到推荐的整体流程分析

### ****您的策略优势****

* ✅ **渐进式学习**: 从简单到复杂，符合教育心理学
* ✅ **个性化推荐**: 根据学生表现动态调整
* ✅ **即时反馈**: 答对后立即推荐下一题

## 完整实施步骤

### ****阶段一：验证和优化匹配结果****

#### 1. ****评估当前匹配效果****

* **检查匹配的题目质量**：推荐的题目是否真的与知识点相关？
* **验证难度分布**：匹配的题目是否涵盖了E(简单)、M(中等)、H(困难)各个层次？
* **确认题目数量**：每个知识点是否有足够的题目供渐进式推荐？
* **记录问题**：哪些匹配结果需要调整？

#### 2. ****按难度重新整理匹配结果****

* **难度映射**：E→简单，M→中等，H→困难
* **同难度排序**：相同难度内按相关性分数排序
* **验证梯度**：确保每个知识点都有不同难度的题目可选
* **标记特殊情况**：某些知识点可能缺少简单题目或困难题目

### ****阶段二：设计前端展示逻辑****

#### 3. ****Summary页面改造****

* **位置设计**：在每个知识点块的什么位置放置"相关题目"按钮？
* **按钮样式**：如何显示题目数量和难度信息？
* **状态指示**：如何显示学生的完成进度？(如：✅ 简单题已完成，🔄 中等题进行中)

#### 4. ****题目展示界面设计****

* **弹窗 vs 新页面**：选择哪种展示方式？
* **题目呈现格式**：
  + 题目文本显示
  + 选项排列 (A/B/C/D/E)
  + 难度标识
  + 进度指示器
* **交互元素**：提交按钮、返回按钮、提示按钮

#### 5. ****渐进式推荐界面****

* **首次显示**：只显示最简单的一道题
* **答对后界面**：显示"恭喜！解锁下一题"+ 新题目
* **答错处理**：是否给提示？是否允许重试？
* **完成状态**：所有难度完成后显示什么？

### ****阶段三：后端API设计****

#### 6. ****题目推荐API设计****

* **初始推荐接口**：
  + 输入：知识点ID
  + 输出：最简单的相关题目 + 总题目数量
* **渐进推荐接口**：
  + 输入：知识点ID + 当前完成的题目ID + 答题结果
  + 输出：下一道题目 或 完成提示

#### 7. ****学习进度管理****

* **进度存储**：如何记录学生在每个知识点的完成情况？
* **状态管理**：
  + 未开始：显示"开始练习"
  + 进行中：显示当前难度和进度
  + 已完成：显示"已掌握"+ 复习选项
* **数据结构设计**：学生ID + 知识点ID + 完成的题目列表 + 当前难度

#### 8. ****答题验证逻辑****

* **答案检查**：正确答案匹配
* **进度更新**：答对后更新学生进度
* **下一题选择**：
  + 同难度还有题目 → 推荐同难度下一题
  + 同难度完成 → 推荐更高难度第一题
  + 全部完成 → 返回完成状态

### ****阶段四：数据准备和处理****

#### 9. ****题目数据预处理****

* **难度标准化**：确保E/M/H分类准确
* **相关性排序**：每个难度内按匹配分数排序
* **题目完整性检查**：选项、答案、解题思路是否完整
* **格式统一**：LaTeX公式、图片等特殊内容处理

#### 10. ****推荐策略配置****

* **难度进阶规则**：
  + 答对几题后进入下一难度？(建议1题，保持流畅性)
  + 答错后如何处理？(给提示 vs 推荐同难度其他题)
* **题目选择策略**：
  + 相关性优先 vs 多样性平衡
  + 是否避免重复推荐已做过的题目
* **完成标准**：每个难度做几题算"掌握"？

### ****阶段五：用户体验优化****

#### 11. ****加载和缓存策略****

* **预加载**：点击"相关题目"时预先加载所有相关题目
* **缓存机制**：避免重复请求相同知识点的题目
* **离线支持**：是否支持离线答题？

#### 12. ****反馈和引导****

* **即时反馈**：答对/答错的即时提示
* **进度可视化**：进度条、成就徽章等
* **学习建议**：完成后给出学习建议或相关资源推荐

#### 13. ****错误处理****

* **网络异常**：断网时如何处理？
* **数据异常**：某知识点没有匹配题目时显示什么？
* **用户异常**：长时间不答题的处理

### ****阶段六：测试和上线****

#### 14. ****功能测试****

* **完整流程测试**：从点击按钮到完成所有题目
* **边界情况测试**：只有一道题、全答错、网络中断等
* **多用户测试**：不同学生同时使用的情况

#### 15. ****用户体验测试****

* **界面友好性**：按钮位置、题目显示是否清晰？
* **操作流畅性**：从一题到下一题是否自然？
* **学习体验**：渐进式难度是否有助于学习？

#### 16. ****性能优化****

* **响应速度**：题目加载是否够快？
* **并发处理**：多个学生同时答题的性能
* **数据存储**：学习进度存储是否高效？

## 关键决策点

### ****需要您现在决定的问题****

1. **题目展示方式**：弹窗 vs 新页面？
2. **难度进阶策略**：一题通过就升级 vs 多题通过？
3. **答错处理**：给提示继续 vs 推荐其他同难度题？
4. **进度存储**：需要登录系统吗？还是用本地存储？

### ****优先级建议****

**高优先级 (先做)**：

* 验证匹配结果质量
* 设计基础的渐进推荐逻辑
* 实现简单的前端展示

**中优先级 (后做)**：

* 完善用户体验细节
* 添加进度可视化
* 优化性能和缓存

**低优先级 (可选)**：

* 离线支持
* 高级分析功能
* 社交分享功能

## 建议的实施顺序

## **今天和明天的开发目标**

### **核心功能模块：**

#### **1. 弹窗系统**

* **触发机制**：在Summary的可点击知识点旁边添加"相关题目"按钮
* **弹窗设计**：模态对话框，包含题目展示区域和对话界面
* **状态管理**：弹窗打开/关闭、题目加载状态

#### **2. 题目展示系统**

* **题目渲染**：显示题目内容、选项、难度信息
* **题目切换**：当前题目完成后自动加载下一题
* **进度显示**：显示当前是第几题，总共多少题

#### **3. 对话交互系统**

* **答案输入**：用户输入答案的界面
* **AI评估**：判断答案正确性并给出反馈
* **对话历史**：记录整个对话过程
* **提示系统**：当用户答错时提供解题提示

#### **4. 推荐逻辑**

* **初始推荐**：优先显示HIGH相关度且难度最低的题目
* **递进机制**：答对后推荐难度稍高的题目
* **题目池管理**：维护可推荐的题目列表

### **技术实现要点：**

#### **前端部分**：

* 在 script\_video\_player.js 中添加弹窗组件
* 在 style\_video\_player.css 中添加弹窗样式
* 实现题目展示和对话交互的UI

#### **后端部分**：

* 在 main\_v3\_1.py 中添加题目推荐API
* 实现对话状态管理和题目递进逻辑
* 集成现有的匹配算法

#### **数据流设计**：

text

用户点击"相关题目" → 获取知识点信息 → 匹配相关题目 → 返回题目数据 → 前端展示弹窗 → 用户答题 → AI评估 → 更新推荐

### **具体实现步骤：**

### **开发优先级调整**

#### **今天完成**：

1. **基础题目匹配**：使用现有规则匹配
2. **弹窗UI**：题目展示界面
3. **基础对话**：简单的答案评估

#### **明天完成**：

1. **LLM语义匹配**：改进题目匹配算法
2. **个性化反馈**：LLM答案评估和反馈
3. **推荐优化**：基于用户表现的推荐

### **文件修改计划：**

#### **需要修改的文件**：

* static/script\_video\_player.js：添加弹窗和对话逻辑
* static/style\_video\_player.css：添加弹窗样式
* main\_v3\_1.py：添加题目推荐API
* knowledge\_tagger.py：集成到现有系统中

#### **需要创建的文件**：

* templates/question\_dialog.html：弹窗模板
* static/script\_question\_dialog.js：弹窗交互逻辑
* static/style\_question\_dialog.css：弹窗样式

### **测试策略：**

1. **功能测试**：验证弹窗打开、题目显示、对话交互
2. **匹配测试**：确认推荐的题目确实相关
3. **用户体验测试**：确保对话流程顺畅

好的，我们今天专注于实现**题目展示页面**和**基础对话功能**。让我规划一下具体的实现步骤：

## **今天的目标：完整的题目解答对话流程**

### **功能范围：**

1. ✅ 题目展示页面（弹窗形式）
2. ✅ 用户输入答案
3. ✅ LLM评估答案
4. ✅ 对话式解题指导
5. ✅ 完成一道题目的完整解答流程

### **实现步骤：**

#### **第一步：创建弹窗组件**

* 在 script\_video\_player.js 中添加弹窗HTML结构
* 在 style\_video\_player.css 中添加弹窗样式
* 实现弹窗的打开/关闭功能

#### **第二步：题目展示界面**

* 显示题目内容、选项
* 添加答案输入框
* 添加提交按钮
* 显示对话历史区域

#### **第三步：后端API**

* 在 main\_v3\_1.py 中添加题目推荐API
* 添加答案评估API
* 集成现有的题目匹配逻辑

#### **第四步：对话功能**

* 用户提交答案
* LLM评估答案正确性
* 生成个性化反馈
* 提供解题指导
* 支持多轮对话直到完成

#### **第五步：集成到现有系统**

* 在Summary的可点击知识点旁添加"相关题目"按钮
* 点击后打开弹窗并加载相关题目
* 完成对话后可以关闭弹窗

### **技术要点：**

#### **前端**：

* 模态弹窗设计
* 实时对话界面
* 答案输入和提交
* 对话历史显示

#### **后端**：

* 题目推荐API
* 答案评估API
* LLM集成
* 对话状态管理

#### **数据流**：

text

用户点击"相关题目" → 获取题目 → 显示弹窗 → 用户答题 → LLM评估 → 显示反馈 → 继续对话 → 完成解答

### **测试场景：**

1. 用户点击知识点旁的"相关题目"按钮
2. 弹窗打开，显示匹配的题目
3. 用户输入答案并提交
4. LLM评估答案并给出反馈
5. 用户继续对话直到完全理解解题思路
6. 完成一道题目的完整解答流程

### **成功标准：**

* ✅ 弹窗能正常打开和关闭
* ✅ 题目内容正确显示
* ✅ 用户能输入答案
* ✅ LLM能正确评估答案
* ✅ 对话流程顺畅
* ✅ 能完成一道题目的完整解答

## **UI设计计划**

### ✅ **第一步：在Summary的可点击知识点旁添加"相关题目"按钮**

我们需要修改 script\_video\_player.js 中的 makeInteractiveSummary 函数，在知识点旁边添加一个按钮。

### ✅ **第二步：创建弹窗HTML结构**

在 video\_player.html 模板中添加弹窗的HTML结构。

### ✅ **第三步：添加弹窗样式**

在 style\_video\_player.css 中添加弹窗的CSS样式。

### ✅ **第四步：实现弹窗交互逻辑**

在 script\_video\_player.js 中添加弹窗的打开/关闭和题目展示逻辑。

### **第一阶段：保存当前UI设计到main分支**

1. **当前UI功能分析**

* ✅ 弹窗对话界面已设计完成
* ✅ 题目显示和选项选择功能
* ✅ 答案评估和反馈系统
* ✅ 对话历史记录
* ✅ 提示和帮助功能
* ❌ 目前使用模拟数据，需要连接真实API

## ✅ 第二阶段：真实API调用系统

1. **题目推荐API集成**

* 连接 match\_knowledge\_points API
* 根据知识点获取真实题目
* 动态加载题目内容

1. **LLM对话系统设计**

* 创建智能对话接口
* 支持多轮对话
* 个性化学习建议

## ✅ 第三阶段：LLM对话系统架构

## ✅ 第四阶段 调整这个版本的功能

done

✅ 弹窗页面高一点

✅ 题目选择题数字显示成时间

✅ 弹窗和浏览器页面的问题

function closePracticeDialog() {

// 🆕 修复：直接关闭当前窗口

window.close();

}

## 后期-优化计划：

### 题目匹配：# 第二层：调用大模型，进行语义相似度匹配

语义理解匹配，Promptlike:

prompt = f""" 知识点: {knowledge\_point} 题目内容: {question\_content} 这个题目是否真的与知识点匹配？请分析： 1. 知识点讲的是什么概念？ 2. 题目考察的是什么技能？ 3. 匹配度 (0-10分) """

1. 数据埋点：记录学生的数据，对LLM的回复进行支持或者反对，记录下这些数据为以后做研究用
2. **多语言支持**

* 中英文界面切换
* 动态内容本地化

1. **智能对话界面**

* 实时对话流
* 个性化学习路径
* 进度跟踪

1. **自适应学习**

* 根据答题表现调整难度
* 个性化学习建议
* 知识点掌握度分析